

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Л.С. Глоба, С.В. Суліма, М.А. Скулиш

РОБОТА З БАЗАМИ ДАНИХ

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Інформаційно-комунікаційні технології»
спеціальності 172 Електронні комунікації та радіотехніка

Електронне мережеве навчальне видання

2-е видання

Київ
КПІ ім. Ігоря Сікорського
2024

Автори: *Глоба Лариса Сергіївна*, д-р техн. наук, проф.
Суліма Світлана Валеріївна, канд. техн. наук, доц.
Скулиш Марія Анатоліївна, д-р техн. наук, проф.

Рецензент *Кравчук Сергій Олександрович*,
д. т. н., проф., зав. каф. телекомунікацій НН ІТС КПІ ім. Ігоря
Сікорського

Відповідальний д.т.н *Скулиш Марія Анатоліївна*
редактор

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 5 від 06.03.2025 р.)
за поданням вченої ради навчально-наукового інституту телекомунікаційних систем
(протокол № 8 від 23.09.2024 р.)*

Глоба Л. С.

В19 Робота з базами даних [Електронний ресурс] : навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Інформаційно-комунікаційні технології» спец. 172 Електронні комунікації та радіотехніка / Л.С. Глоба, С.В. Суліма, М.А. Скулиш; КПІ ім. Ігоря Сікорського. – 2-е вид. – Електрон. текст. дані (1 файл: 8.7 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2024. – 532 с.

Починаючи з вивчення основних понять, читачі знайомляться з сутністю баз даних, заглиблюються в реляційну модель, реляційну алгебру та реляційні обчислення, висвітлюючи основні принципи, які лежать в основі сучасних систем баз даних. Посібник представляє методологію проектування бази даних, озброюючи читачів основними стратегіями для створення надійних і масштабованих архітектур баз даних. Розглядається управління транзакціями, звертаючись до критичних аспектів забезпечення узгодженості та надійності даних. Також висвітлюється тема нереляційних СУБД, досліджуючи альтернативні підходи до управління даними в епоху великих даних і розподілених систем. Завдяки повному висвітленню та глибокому аналізу посібник слугує дороговказом у величезному просторі управління інформацією..

Навчальний посібник призначений для здобувачів ступеня бакалавра за освіт. програмою «Інформаційно-комунікаційні технології» за спеціальністю 172 Електронні комунікації та радіотехніка, буде також корисним для усіх, хто працює з базами даних.

УДК 621.39 (XXX)

Реєстр. № НП 23/24-317. Обсяг 6,2 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
<https://kpi.ua>

Свідectво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© Л.С. Глоба, С.В. Суліма, М.А. Скулиш, 2024
© КПІ ім. Ігоря Сікорського, 2024

Зміст

Список скорочень	4
ТЕМА 1	5
Вступ до БД.....	5
1. 1 Історія розвитку СКБД.....	5
1.2 Файлові системи	8
1.3 Системи з базами даних	10
1.4 Система керування базами даних - СКБД.....	13
1.5 Моделі даних.....	14
ТЕМА 2	28
Реляційна модель.	28
Реляційна алгебра. Реляційне числення.	28
2.1 Реляційна модель	28
2.2 Реляційні мови	32
2.3 Реляційна алгебра	33
Унарні операції реляційної алгебри.....	34
2.4 Реляційне числення	47
2.5 Інші мови	49
ТЕМА 3	54
Моделювання даних	54
Модель «сутність-зв'язок»	54
ТЕМА 4.....	107
Нормалізація.....	107
4.1 Класи відношень	107
4.2 Нормальні форми від першої до п'ятої	108
4.2 Аналіз критеріїв для нормалізованих і ненормалізованих моделей даних.....	137
4.3 Доменно-ключова нормальна форма (ДКНФ).....	138
4.4 Синтез відношень	141
4.5 Правила побудови відношень.....	145
4.6 Оптимізація	146
ТЕМА 5	149
Методологія проектування баз даних.....	149
5.1 Введення в методологію проектування баз даних	149
5.2 Загальний огляд процедури проектування бази даних	150
5.3 Методологія концептуального проектування бази даних	152
5.4 Методологія логічного проектування реляційних баз даних	162
5.4 Методологія фізичного проектування бази даних	187
ТЕМА 6.....	209
Архітектура систем БД. СУБД	209
6.1 Трирівнева архітектура ANSI-SPARC	209

6.2 Система управління Базами Даних	217
ТЕМА 7	251
Мова SQL.....	251
7.1 Мова SQL: Маніпулювання даними	251
7.2 Мова SQL: Визначення даних	295
7.3 Збережені процедури і функції. Тригери.	338
Тема 8.....	350
Управління транзакціями.....	350
8.1 Підтримка транзакцій.....	350
8.2 Управління паралельним доступом	352
8.3 Відновлення бази даних	380
8.4 Покращені моделі транзакцій.....	390
Тема 9.....	399
Оптимізація запитів.....	399
9.1 Загальний огляд методів обробки запитів.....	399
9.2 Евристичний підхід до оптимізації запитів.....	406
9.3 Оцінка вартості операцій реляційної алгебри.....	410
ТЕМА 10	428
Захист баз даних	428
10.1 Основні типи загроз.....	429
9.2 Привілеї	433
ТЕМА 11	440
Напрямки розвитку баз даних і СУБД.....	440
11.1 Розподілені системи управління базами даних (СУБД)	440
11.2 Об'єктні системи управління базами даних (СУБД)	445
11.3 Об'єктно-реляційні системи управління базами даних СУБД	447
11.4 Документно-орієнтовані СУБД.....	448
11.5 Тенденції розвитку СУБД.....	450
ТЕМА 12	459
Нереляційні СУБД.....	459
12.1 Що таке бази даних NoSQL?	459
12.2 Типи моделей даних з базами даних NoSQL	459
12.3 Нереляційні дані і бази даних NoSQL	460
12.4 Порівняння SQL та NoSQL баз даних	469
12.5 Що таке MongoDB.....	478
Список рекомендованої літератури	531

Список скорочень

АБД - адміністратор бази даних

БД – база даних

ДОБД - документо-орієнтована база даних

ДОСУБД - документо-орієнтована система управління базами даних

ООБД - об'єктно-орієнтована база даних

ООМД – об'єктно-орієнтована модель даних

ООСУБД - об'єктно-орієнтована система управління базою даних

ОРБД - об'єктно-реляційна база даних

СКБД, СУБД – система керування (управління) базами даних

СУРБД – розподілена система управління базами даних

DBTG (Data Base Task Group) - цільова група з бази даних

DDL (Data Definition Language) - мова визначення даних

DML (Data Manipulation Language) - мова маніпулювання даними

GIS (Geographic Information System) - геоінформаційна система

GUAM (Generalized Update Access Method) - узагальнений спосіб доступу до оновлень

IBM (International Business Machines Corporation) – компанія-розробник програмного забезпечення

IMS (Information Management System) – система управління інформацією

OLTP (Online Transaction Processing) - обробка онлайн-транзакцій

RFID (Radio-Frequency Identification) - радіочастотна ідентифікація

SGML (Standard Generalized Markup Language) - стандартна узагальнена мова розмітки

SQL (Structured Query Language) - мова структурованих запитів

XML (Extensible Markup Language) – розширювана мова розмітки

ТЕМА 1

Вступ до БД

Історія досліджень систем баз даних – це, по суті, *історія розвитку прикладного програмування або інформаційних систем*, які досягли виняткової продуктивності і зробили все більш істотний вплив на економіку.

Якщо ще 20 років тому ця сфера була всього лише галуззю фундаментальних наукових досліджень, то тепер на дослідженнях баз даних заснована ціла індустрія інформаційних послуг, щорічний бюджет якої тільки в США становить 10 мільярдів доларів. Досягнення в дослідженнях баз даних стали основою фундаментальних розробок комунікаційних систем, транспорту і логістики, фінансового менеджменту, систем з базами знань, методів доступу до наукової літератури, а також великої кількості цивільних і військових програм. Вони також послужили фундаментом значного прогресу в провідних галузях науки - від інформатики до біології, - Зільбершцац (Silberschatzetal., 1991).

Можна стверджувати, що поява баз даних стала найважливішим досягненням у галузі програмного забезпечення. Бази даних лежать в основі інформаційних систем, і це докорінно змінило характер роботи багатьох організацій. Із моменту своєї появи технологія баз даних стала каталізатором багатьох значних досягнень в галузі програмного забезпечення.

Сьогодні людство знаходиться тільки в кінці початку їх розвитку. *Прикладне програмне забезпечення, яким доведеться користуватися в майбутньому, виявиться настільки складним, що потрібно буде переосмислити багато алгоритмів - наприклад, використовувані в даний час алгоритми зберігання файлів, а також доступу до них.*

1. 1 Історія розвитку СКБД

Як уже згадувалося вище, попередницями СКБД були файлові системи. Однак поява СКБД не призвела до повного зникнення файлових систем. Для виконання деяких спеціалізованих завдань файлові системи використовуються й досі. Вважається, що розвиток СКБД почався ще в 60-і роки 20-го сторіччя, коли був у розробці проект польоту корабля Apollo на Місяць. Цей проект був розпочатий з ініціативи президента США Дж. Ф. Кеннеді, який поставив завдання висадити людину на Місяць до кінця десятиліття. У той час не існувало ніяких систем, здатних обробляти або будь-яким чином керувати тією величезною кількістю даних, яка була необхідною для реалізації цього проекту. У результаті, фахівці основного підрядника - фірми North American Aviation (NAA, тепер ця фірма називається Rockwell International) - розробили програмне забезпечення під назвою GUAM (Generalized Update Access Method). Основна ідея GUAM полягала в тому, що малі компоненти об'єднуються разом як частини більших компонентів до тих пір, поки не буде зібраний воедино весь проект. Ця відповідна інвертованому дереву структура часто називається ієрархічною структурою (hierarchical structure). У середині 60-х років корпорація IBM приєдналася до фірми NAA для спільної роботи над GUAM, у результаті чого була створена система IMS (Information Management System). Причина, за якої корпорація IBM обмежила функціональні можливості IMS тільки керуванням ієрархіями записів, полягала в тому, що необхідно було забезпечити роботу з пристроями зберігання з послідовним доступом, а саме з магнітними стрічками, які були основним типом носія

у той час. Через деякий час це обмеження вдалося подолати. Незважаючи на те, що IMS є найпершою з усіх комерційних СКБД, вона до сих пір залишається основною ієрархічною СКБД, що використовується на більшості великих мейнфреймів.

Іншим помітним досягненням середини 60-х років була поява системи IDS (Integrated Data Store) фірми General Electric. Роботу над нею очолював один з піонерів досліджень в області систем керування базами даних - Чарльз Бачман. Розвиток цієї системи дозволив створити новий тип систем керування базами даних - мережеву (network) СКБД, - що мало істотний вплив на інформаційні системи того покоління. Мережева СКБД створювалася для представлення більш складних взаємозв'язків між даними, ніж ті, які можна було моделювати за допомогою ієрархічних структур, а також для формування стандарту баз даних. Для створення таких стандартів в 1965 році на конференції організації CODASYL (Conference on Data System Languages), що проходила за участю представників уряду США і бізнесменів, було сформовано робочу групу List Processing Task Force, перейменована в 1967 році в групу Data Base Task Group (DBTG). До компетенції групи DBTG входило визначення специфікацій середовища, яке допускало б розробку баз даних і керування даними. Попередній варіант звіту цієї групи був опублікований в 1969 році, а перший повний варіант - в 1971 році. Пропозиції групи DBTG містили три компоненти:

- Мережева схема - це логічна організація всієї бази даних в цілому (з точки зору адміністратора даних бази даних-АДБ), яка включає визначення імені бази даних, типу кожного запису і компонентів записів кожного типу.

- Підсхема - це частина бази даних у вигляді, як її бачать користувачі або прикладні програми.

- Мова керування даними - інструмент для визначення характеристик і структури даних, а також для керування ними.

Група DBTG також запропонувала стандартизувати три різні мови:

- Мова визначення даних (Data Definition Language) для схеми, яка дозволить адміністратору бази даних (АБД) описати її.

- Мова визначення даних (також DDL) для підсхеми, яка дозволить визначати в прикладних програмах ті частини бази даних, доступ до яких буде необхідний.

- Мова маніпулювання даними (Data Manipulation Language - DML), призначена для керування даними.

Незважаючи на те, що цей звіт офіційно не був схвалений Національним Інститутом Стандартизації США (American National Standards Institute - ANSI), була розроблена велика кількість систем у повній відповідності з цими пропозиціями групи DBTG. Тепер ці системи називають CODASYL-системами, або DBTG-системами. CODASYL-системи і системи на основі ієрархічних підходів є СКБД першого покоління.

Однак цим двом моделям притаманні наведені нижче недоліки:

- навіть для виконання простих запитів із використанням переходів і доступом до певних записів необхідно створювати досить складні програми;
- незалежність від даних існує лише в мінімальному ступені;
- відсутність загальновизнаних теоретичних основ.

У 1970 році Е. Ф. Кодд, який працював у дослідницькій лабораторії корпорації IBM, опублікував дуже важливу і дуже своєчасну статтю про реляційну модель даних,

що дозволяла усунути недоліки колишніх моделей. Слідом за цим з'явилася безліч експериментальних реляційних СКБД, а перші комерційні продукти з'явилися в кінці 70-х - початку 80-х років. Особливо слід зазначити проект System R, розроблений в дослідницькій лабораторії корпорації IBM, яка розташована в місті Сан-Хосе, штат Каліфорнія, та створений в кінці 70-х років (Astrakhan et al., 1976). Цей проект був задуманий з метою довести практичність реляційної моделі, яка досягалась за допомогою реалізації передбачених нею структур даних і необхідних функціональних можливостей. На основі цього проекту були отримані найважливіші результати:

- Була розроблена структурована мова запитів SQL, яка з тих пір стала стандартною мовою будь-яких реляційних СКБД.
- У 80-х роках були створені різні комерційні реляційні СКБД - наприклад, DB2 або SQL/DS корпорації IBM та Oracle корпорації Oracle Corporation.

У наш час існує декілька сотень різних реляційних СКБД для мейнфреймів і мікрокомп'ютерів, хоча для багатьох із них визначення реляційної моделі має дещо перебільшений характер. Прикладом багатокористувацьких СКБД може служити система CA-OpenIngress фірми Computers Associates і система Informix фірми Informix Software Inc.

Прикладами реляційних СКБД для персональних комп'ютерів є Access і FoxPro фірми Microsoft, Paradox і Visual dBase фірми Borland.

Реляційні СКБД відносяться до СКБД другого покоління.

Однак реляційна модель також має деякі недоліки - зокрема, обмежені можливості моделювання. Для вирішення цієї проблеми було виконано великий обсяг дослідницької роботи. У 1976 році Чен запропонував модель "сутність-зв'язок" (Entity-relationship model - ER-модель), яка в даний час стала найпоширенішою технологією проектування баз даних і є основою методології моделювання даних.

У 1979 році Кодд зробив спробу усунути недоліки власної основної роботи і опублікував розширену версію реляційної моделі - RM/T (1979), потім ще одну версію - RM/V2 (1990). Спроби створення моделі даних, яка дозволяє більш точно описувати реальний світ, нестрого називають семантичним моделюванням даних (semantic data modeling).

У відповідь на всезростаючу складність прикладного програмного забезпечення баз даних з'явилися дві нові системи: об'єктно-орієнтовані СКБД, або ОО СКБД (Object-Oriented DBMS-OODBMS), і об'єктно-реляційні СКБД, або ОР-СКБД (Object-Relational DBMS-ORDBMS). Однак, на відміну від попередніх моделей, справжня структура цих моделей не зовсім ясна.

Система баз даних— це комп'ютеризована система зберігання записів, тобто комп'ютеризована система, головне призначення якої - зберігати інформацію, надаючи користувачам засоби її вилучення та модифікації. До інформації може відноситися все, що заслуговує на увагу окремого користувача або організації.

База даних: Спільно використовуваний набір логічно пов'язаних даних (разом з описом цих даних), призначений для задоволення інформаційних потреб, передбачених якоюсь інформаційною системою.

База даних – це єдине, велике сховище даних, яке одноразово визначається, а потім використовується одночасно багатьма користувачами через різні прикладні програми, завдяки чому база даних є загальним корпоративним ресурсом.

Система керування базами даних, або СКБД (Database Management System – DBMS) – програмне забезпечення, яке керує доступом до цієї бази даних.

Наприклад, доступ до бази даних необхідний при покупці продуктів у найближчому супермаркеті, коли касир зчитує з покупок штрих-код. При цьому ручний сканер, пов'язаний з прикладним програмним компонентом бази даних, використовує штрих-код для пошуку ціни даного предмета в базі даних всіх товарів. Потім програма відніме кількість всіх щойно проданих товарів з товарних запасів і відобразить на касовому апараті їх вартість. Якщо запаси складу опустяться нижче деякого заздалегідь визначеного рівня, то в такому випадку система зможе автоматично направити замовлення на поставку додаткової кількості даного товару. Коли клієнт робить покупки телефоном, касир може перевірити наявність того чи іншого товару на складі, також запустивши відповідний компонент баз даних.

Якщо під час покупки використовується кредитна картка, касир має перевірити наявність кредитних коштів. Це можна зробити або за номером телефону, або автоматично, за допомогою спеціального пристрою, який зчитує та є пов'язаним із комп'ютером. У будь-якому випадку при цьому використовується якась база даних, яка містить відомості про покупки, що здійснюються за допомогою кредитної картки. На основі номеру кредитної картки спеціальний прикладний програмний компонент звіряє ціну товарів, що купуються в даний момент і товарів, куплених протягом цього місяця з кредитним лімітом. Після підтвердження допустимості такої покупки, усі відомості щодо придбаних товарів вводяться в базу даних. Однак, ще до отримання підтвердження допустимості покупки, прикладний програмний компонент бази даних має переконатися в тому, що дана картка не перебуває у списку вкрадених або загублених. Крім того, має існувати ще один самостійний прикладний програмний компонент баз даних, який буде оплачувати рахунки після отримання суми платежу, а також щомісяця надсилати кожному власнику кредитної картки повний звіт.

Можна розглядати й інші прикладні програми, які працюють з БД.

1.2 Файлові системи

Зберігання даних як інформаційного ресурсу раніше здійснювалося безпосередньо в тілі програми, що було вкрай неефективно. Наступним кроком розвитку в області інформаційних технологій стали традиційні **файлові системи (file-based system)**.

Файлові системи – це набір програм, які виконують для користувачів деякі операції, наприклад, створення звітів, причому кожна програма визначає свої власні дані і керує ними.

Файлові системи було розроблено у відповідь на потребу в отриманні більш ефективних способів доступу до даних.

Суть підходу, що використовують у файлових системах, полягає в наступному. Замість організації централізованого сховища всіх даних, які використовуються прикладними програмами, було використано децентралізований підхід, при якому кожна програма працювала зі своїми власними даними і зберігала їх тільки одним їй відомим способом.

Файловим системам властиві такі обмеження:

- Розділ та ізоляція даних.
- Дублювання даних.
- Залежність від даних.
- Несумісність файлів.
- Фіксовані запити/швидке збільшення кількості прикладних програмних компонентів, які забезпечують доступ до даних.

Розділ та ізоляція даних: дані ізолювані в окремих файлах, доступ до них дуже ускладнений, виконувати подібну обробку даних у файлових системах досить складно. Для вилучення відповідної поставленим умовам інформації, програміст має організувати відповідну обробку всіх необхідних файлів даних.

Дублювання даних: через децентралізовану роботу з даними у файловій системі, фактично виконується безконтрольне дублювання даних. Безконтрольне дублювання даних є небажаним через такі дві причини:

1. *Дублювання даних супроводжується неекономною витратою ресурсів, оскільки на введення надлишкових даних потрібно витрачати додаткові час і гроші. Більш того, для їх зберігання необхідно додаткове місце у зовнішній пам'яті, що пов'язано з додатковими витратами. У багатьох випадках дублювання даних можна уникнути за рахунок спільного використання файлів.*

2. *Ще більш важливим є той факт, що дублювання даних може призвести до порушення їх цілісності.*

Наприклад, розглянемо описаний вище випадок дублювання даних в розрахунковому секторі бухгалтерії та відділі кадрів. Якщо співробітник переїде в інший будинок, і зміну адреси буде зафіксовано тільки в відділі кадрів, то повідомлення про зарплату буде послано йому старою, тобто помилковою, адресою. Більш серйозна проблема може виникнути, якщо якийсь співробітник отримає підвищення по службі з відповідним збільшенням заробітної плати. І знову ж таки, якщо ця зміна буде зафіксована тільки в інформації відділу кадрів, залишившись непроведеною в файлах розрахункового сектору, то даний співробітник помилково буде отримувати колишню заробітну плату. При виявленні подібної помилки для її виправлення потрібно затратити додатковий час і кошти.

Обидва приклади демонструють протиріччя, які можуть виникнути при дублюванні даних. Оскільки не існує ніякого автоматичного способу поновлення даних одночасно і в файлах відділу кадрів, і в файлах розрахункового сектору, неважко передбачити, що подібні протиріччя час від часу обов'язково виникатимуть. Навіть якщо співробітники розрахункового сектору після отримання повідомлень про подібні зміни будуть негайно їх вносити, все одно існує ймовірність декількох невдалих спроб вводу змінених даних.

Залежність від даних: фізична структура і спосіб зберігання записів файлів даних жорстко зафіксовані в коді прикладних програм. Це означає, що змінити існуючу структуру даних досить складно.

Наприклад, збільшення в файлі довжини поля адреси будь-якого даного з 40 до 41 символу здається абсолютно незначною зміною його структури, але для втілення цієї зміни буде потрібно, як мінімум, створити одноразову програму спеціального призначення (тобто програму, яка виконується тільки один раз), перетворюючи вже існуючий файл в новий формат. Вона має виконувати такі дії:

- відкрити вихідний файл для читання;
- відкрити тимчасовий файл із новою структурою запису;
- зчитати запис з вихідного файлу, перетворити дані в новий формат і записати їх у тимчасовий файл. Ці дії слід виконати для всіх записів вихідного файлу;
- видалити вихідний файл;
- присвоїти тимчасовому файлу необхідне ім'я.

Таким чином, виконання всіх цих дій вимагає великих витрат часу і може стати причиною появи помилок.

Така особливість файлових систем називається залежністю від програм даних (program-data dependence).

Несумісність форматів файлів: структура файлів визначається кодом прикладних програмних компонент, вона також залежить від мови програмування цієї прикладної програми.

Наприклад, структура файлу, створеного програмою мовою *COBOL*, може абсолютно відрізнятися від структури файлу, що створюється програмою мовою *C*. *Пряма несумісність таких файлів ускладнює процес їхньої спільної обробки.*

Фіксовані запити/швидке збільшення кількості прикладних програм. Із точки зору користувача постійно зростають вимоги до реалізації нових або модифікованих запитів. Однак файлові системи багато в чому залежать від програміста, тому що всі необхідні запити і звіти мають бути створеними саме ним.

У результаті події зазвичай розвивалися за одним із таких двох сценаріїв. У багатьох випадках типи створюваних запитів і звітів мали фіксовану форму, і не було ніяких інструментів створення незапланованих або довільних (ad hoc) запитів, як до самих даних, так і до відомостей про те, які типи даних доступні.

В інших випадках спостерігалось швидке збільшення кількості файлів і прикладних програм. У кінцевому рахунку, наступав момент, коли ставало просто неможливо впоратися з усією цією роботою за допомогою наявних ресурсів. У цьому випадку навантаження настільки зростало, що неминуче наступав момент, коли програмне забезпечення було не здатне адекватно відповідати запитам користувачів, ефективність його падала, а недостатність документування служила додатковим ускладненням супроводу програм. При цьому часто ігнорувалися питання підтримки функціональності системи: не передбачалися заходи щодо забезпечення безпеки або цілісності даних; засоби відновлення в разі збою апаратного або програмного забезпечення були вкрай обмеженими або взагалі відсутніми. Доступ до файлів часто обмежувався одним користувачем.

1.3 Системи з базами даних

Усі зазначені вище обмеження файлових систем є наслідком **двох факторів:**

1. Визначення даних міститься всередині прикладних програм, а не зберігається окремо і незалежно від них.

2. Крім прикладних програм *не передбачено ніяких інших інструментів доступу до даних і їх обробки.*

Для підвищення ефективності роботи необхідно використовувати новий підхід, а саме базу даних (database) і систему керування базами даних, або СКБД (Database management System-DBMS).

База даних: Набір логічно пов'язаних даних (з описом цих даних), що використовують спільно, призначений для задоволення інформаційних потреб, передбачених якоюсь інформаційною системою.

База даних – це єдине, велике сховище даних, яке одноразово визначається, а потім використовується одночасно багатьма користувачами через різні прикладні програми, завдяки чому база даних є загальним корпоративним ресурсом.

Замість розрізнених файлів із надлишковими даними, тут всі дані зібрані разом із мінімальною часткою надмірності.

База даних зберігає не тільки робочі дані, але і їх описи. Із цієї причини базу даних ще називають набором інтегрованих записів із *самоописом*.

У сукупності, **опис даних** називається **системним каталогом** (system catalog), або **словником даних** (data dictionary), а самі **елементи опису** прийнято називати **метаданими** (meta-data), тобто “даними про дані”.

Саме наявність самоопису даних у базі даних забезпечує незалежність між програмами і даними (program-data independence).

Підхід, заснований на застосуванні баз даних, де визначення даних відокремлено від прикладних програм, дуже схожий на підхід, який використовують при розробці сучасного програмного забезпечення, коли поряд з внутрішнім визначенням об'єкту існує його зовнішнє визначення. Користувачі об'єкту бачать тільки його зовнішнє визначення і не піклуються про те, як він визначається і як функціонує.

Одна з переваг такого підходу, а саме абстрагування даних (data abstraction), полягає в тому, що можна змінити внутрішнє визначення об'єкта без будь-яких наслідків для його користувачів, за умови, що зовнішнє визначення об'єкта залишається незмінним.

Аналогічним чином, у підході з використанням баз даних, структура даних відокремлена від прикладних програм і зберігається в базі даних. Додавання нових структур даних або зміна існуючих ніяк не впливає на прикладні програми, за умови, що вони не залежать безпосередньо від змінюваних компонентів.

Наприклад, додавання нового поля в запис або створення нового файлу ніяк не вплине на роботу наявних прикладних програм. Проте видалення поля з використовуваного прикладною програмою файлу вплине на цю прикладну програму, а тому його також потрібно відповідним чином модифікувати.

У базах даних використовуються *логічно пов'язані дані*. Термін "*логічно пов'язаний*" використовується, коли при аналізі інформаційних потреб організації слід виділити *сутності, атрибути та зв'язки*.

Сутністю (entity) називається окремий тип об'єкта (людина, місце або річ, поняття або подія), який потрібно представити в базі даних.

Атрибутом (attribute) називається властивість, яка описує деяку характеристику об'єкта, який описується;

Зв'язок (relationship) – це те, що об'єднує кілька сутностей.

Діаграму "сутність-зв'язок", або ER-діаграму (Entity relationship – ER) розглянемо на прикладі абстрактного навчального процесу. Він складається з таких компонентів:

- Сутностей (позначені прямокутниками): група, студент, викладач, кафедра, предмет, вивчення, успішність.
- Зв'язків (позначені ромбами), які об'єднують основні сутності. Наприклад, ГС (скорочення Група/Студент): кожен студент знаходиться в певній групі, і кожна група містить певних студентів. Аналогічно, кожна група належить до певної кафедри, а кафедра складається з певних груп (зв'язок ГК – скорочення від Група/Кафедра); викладач працює на певній кафедрі, а кафедра має певну кількість викладачів (зв'язок КВ – скорочення Кафедра/Викладач) і т.д.

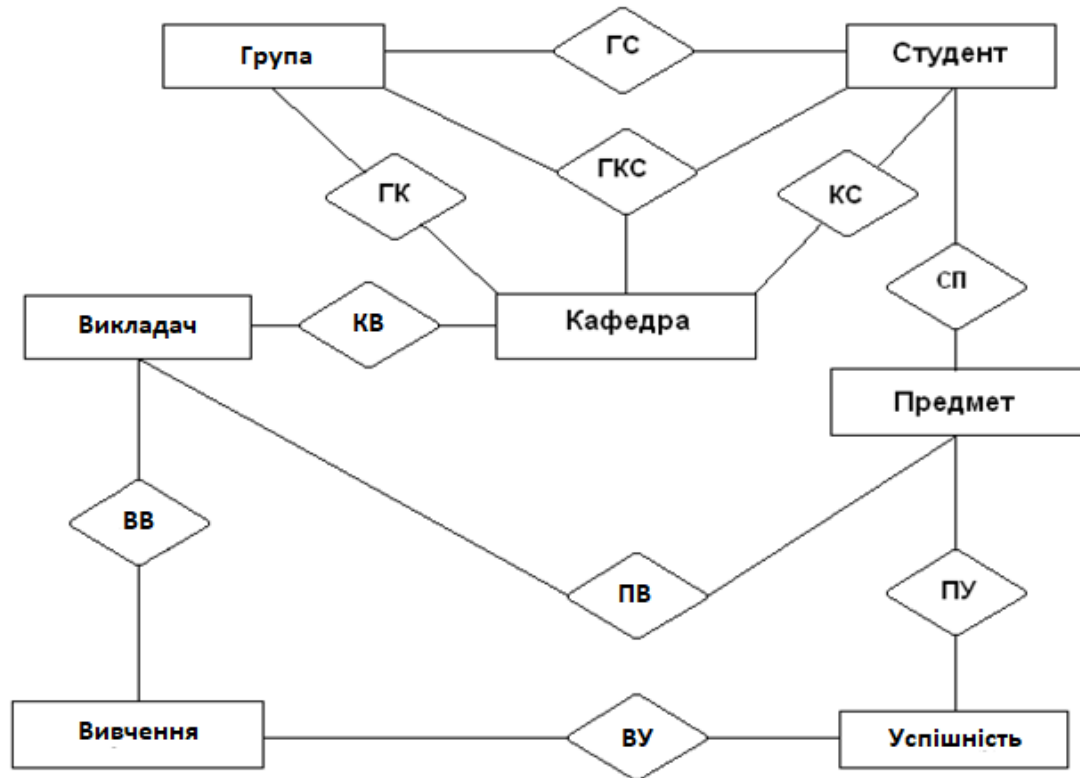


Рис. 1.1. Приклад діаграми «сутність-зв'язок»

Подібна база даних представляє сутності, атрибути й логічні зв'язки між об'єктами. Інакше кажучи, база даних містить логічно пов'язані дані.

Зазвичай дані в базі даних називають *перманентними* або *такими, що постійно зберігаються* (хоча іноді насправді вони недовго такими залишаються). Під словом **перманентні** (*persistent*) мають на увазі дані, які відрізняються від інших, більш мінливих даних, таких як проміжні результати, вхідні і вихідні дані, керуючі оператори, робочі черги, програмні керуючі блоки й взагалі всі дані, *тимчасові* (*transient*) за своєю суттю. Можна стверджувати, що дані в базі є *перманентними*, оскільки після того як вони були прийняті засобами СКБД для розміщення в базу, їх подальше видалення можливе лише при використанні відповідного явного запиту до бази даних, але не в результаті будь-якого побічного ефекту від виконання деякої програми. Подібний погляд на поняття перманентність дозволяє точніше визначити термін база даних.

База даних (БД) – це певний набір перманентних даних (тих, що постійно зберігаються), які використовують прикладні програмні системи будь-якого підприємства.

Примітка. Спочатку замість *перманентних* даних використовували термін *операційні дані*. Старий термін акцентував увагу на особливому значенні **оперативного**, або **виробничого** прикладного програмного забезпечення баз даних, тобто рутинного

прикладного програмного забезпечення, яке часто використовують. Останнім часом використовують термін **оперативна обробка транзакцій** (*On-Line Transaction Processing - OLTP*). Однак тепер бази даних все частіше застосовують в прикладних програмних системах **підтримки прийняття рішень** (*decision support*), і термін *операційні дані* для них уже не підходить. На сьогоднішній день на практиці використовують дві окремі бази даних - з операційними даними і з даними для підтримки прийняття рішень; останню зазвичай називають **сховищем даних** (*data warehouse*). У сховищах даних часто міститься агрегована інформація, яка, в свою чергу, періодично вилучається із операційної бази даних.

Типи інформації в базі даних:

1. **Дані користувачів.** На сьогоднішній день більшість БД представляють дані користувачів у вигляді *відношень* (*relations*).

Відношення – це *таблиця даних*. Стовпці таблиці містять *поля* або *атрибути*, а рядки містять *записи* про конкретні об'єкти ділового світу.

2. **Метадані.** Як відомо база даних є самодокументованою, тобто однією з її складових є *опис власної структури*. Такий опис називається **метаданими**. Так як СКБД призначені для зберігання таблиць і маніпуляції ними, більшість з них зберігають *метадані у формі таблиць*, які іноді називають *системними таблицями*. Зберігання метаданих в таблицях не лише є ефективним для СКБД, але і зручним для розробників, оскільки для запиту метаданих вони можуть використовувати ті ж самі засоби, що й для запиту даних, які призначені користувачу.

3. **Індекси.** Ще один тип даних, що зберігають в базі даних, введено для покращення її продуктивності й доступності. Ці дані, що називають іноді надлишковими даними, складаються головним чином з індексів, хоча в деяких випадках використовуються й інші структури даних, такі як пов'язані списки. Індекси використовують не тільки для сортування, а й для швидкого доступу до даних.

4. **Метадані прикладного програмного забезпечення.** Останній тип інформації в базі даних – це метадані прикладного програмного забезпечення, які описують структуру та формат форм користувача, звітів, запитів й інших компонентів прикладного програмного забезпечення. Не всі СКБД підтримують компоненти прикладного програмного забезпечення, а в тих СКБД, де таку можливість передбачено, не всі зберігають структуру цих компонентів у вигляді метаданих прикладного програмного забезпечення в базі даних. Однак більшість сучасних СКБД зберігають цю інформацію в базі даних. Взагалі кажучи, ні розробники баз даних, ні користувачі не звертаються до метаданих прикладного програмного забезпечення безпосередньо, а користуються відповідними програмними засобами, які надає СКБД.

1.4 Система керування базами даних - СКБД

СКБД – це програмне забезпечення, за допомогою якого користувачі можуть визначати, створювати й підтримувати базу даних, а також здійснювати до неї контрольований доступ.

СКБД можна також визначати, як програмне забезпечення, що взаємодіє з прикладними програмами користувача і базою даних і володіє наведеними нижче можливостями:

1. Дає можливість визначати базу даних, що зазвичай здійснюється за допомогою мови визначення даних (**DDL - Data Definition Language**). DDL

призначений для надання користувачам засобів опису типу даних і їх структури, а також засобів завдання обмежень для інформації, яку зберігають в базі даних.

2. Дає можливість вставляти, оновлювати, видаляти й витягувати інформацію з бази даних, що зазвичай здійснюють за допомогою мови керування даними (Data Manipulation Language). Наявність централізованого сховища всіх даних і їх описів дозволяє використовувати мову DML як загальний інструмент організації запитів, який іноді називають мовою запитів (Query Language). Мова запитів дозволяє усунути властиві файловим системам обмеження, за яких користувачам доводиться мати справу тільки з фіксованим набором запитів, а не постійно зростаючою кількістю програм, що породжує інші, більш складні проблеми керування програмним забезпеченням.

Існують дві різновиди мов DML – *процедурні (procedural)* та *непроцедурні (non-procedural)* мови, які вирізняють одна від одної за способом вилучення даних. **Процедурні мови** зазвичай обробляють інформацію в базі даних послідовно, запис за записом, а **непроцедурні** оперують одразу цілими наборами записів. За допомогою процедурних мов DML зазвичай вказують, як можна отримати бажаний результат, тоді як непроцедурні мови DML використовують для опису того, що слід отримати.

Найбільш поширеним типом *непроцедурної мови* є **мова структурованих запитів** (*Structured Query Language – SQL*), котру в наш час визначають спеціальним стандартом і яка фактично є обов'язковою мовою для будь-яких реляційних СКБД.

3. **СКБД** надає контрольований доступ до бази даних за допомогою перерахованих нижче засобів:

- *системи забезпечення безпеки*, яка запобігає несанкціонованому доступу до бази даних із боку користувачів;
- *системи підтримки цілісності даних*, яка забезпечує несуперечливий стан даних, які зберігаються;
- *системи керування паралельною роботою прикладних програм*, яка контролює процеси їх сумісного доступу до бази даних;
- *системи відновлення*, яка дозволяє відновити базу даних до попереднього несуперечливого стану, порушеного в результаті збою апаратного або програмного забезпечення;
- *доступного користувачам каталогу*, який містить опис інформації, що зберігається в базі даних.

1.5 Моделі даних

Модель даних – це інтегрований набір понять для опису даних, зв'язків між ними та обмежень, що накладаються на дані в деякій організації.

Модель є представленням об'єктів і подій "реального світу", а також існуючих між ними зв'язків. Це деяка абстракція, в якій акцент робиться на найважливіших невід'ємних аспектах діяльності організації, а всі другорядні властивості ігноруються. Модель має відображати основні концепції, представлені в такому вигляді, який дозволить проектувальникам і користувачам бази даних обмінюватися конкретними і недвозначними думками про своє розуміння ролі тих або інших даних у цій організації.

Модель даних розглядається як поєднання трьох компонентів:

- **Структурна частина** – набір правил, за допомогою яких може бути побудована база даних.
- **Керуюча частина**, що визначає типи допустимих операцій із даними (сюди відносяться операції оновлення й вилучення даних, а також операції зміни структури бази даних).
- **Набір обмежень підтримки цілісності даних** (не є обов'язковим), що гарантують правильність використовуваних даних.

Метою побудови моделі даних є подання даних у зрозумілому вигляді. Якщо таке подання можливе, то модель даних можна буде легко застосувати при проектуванні бази даних.

Всі моделі даних діляться на три категорії:

- об'єктні (object-based) моделі даних;
- моделі даних на основі записів(record-based);
- фізичні моделі даних.

Об'єктні моделі даних

При побудові об'єктних моделей даних використовують поняття *сутність, атрибути та зв'язки*.

Найбільш застосовувані типи об'єктних моделей даних:

- *Модель типу “сутність-зв'язок” або ER-модель (Entity-Relationship model);*
- *Семантична модель;*
- *Функціональна модель;*
- *Об'єктно-орієнтована модель.*

В даний час ER-модель стала одним із основних методів концептуального проектування баз даних.

Об'єктно-орієнтована модель розширює визначення сутності з метою включення в нього не тільки атрибутів, які описують стан об'єкта, але й дій, які з ним пов'язані, тобто його поведінку.

У такому випадку говорять, що *об'єкт інкапсулює стан і поведінку*.

Моделі даних на основі записів

У моделі даних на основі записів база даних складається з декількох записів фіксованого формату, які можуть мати різні типи.

Кожен тип запису визначає фіксовану кількість полів, кожне з яких має фіксовану довжину.

Існує *три основних типи логічних моделей даних на основі записів*:

- *реляційна модель даних (relational data model);*
- *мережна модель даних (network data model);*
- *ієрархічна модель даних (hierarchical data model).*

Ієрархічна і мережна моделі даних були створені майже на десять років раніше реляційної моделі даних, а тому їх зв'язок з концепціями традиційної обробки файлів більш очевидний.

Реляційна модель даних

Короткий огляд історії реляційно моделі

Реляційна модель вперше була запропонована Е.Ф.Коддом в 1970 році у його основоположній статті "Реляційна модель даних для великих спільно використовуваних банків даних". У наш час публікацію цієї статті прийнято вважати поворотним пунктом в історії розвитку систем баз даних, хоча слід зауважити, що ще раніше було запропоновано модель, засновану на множинах (Chils, 1968). Цілі створення реляційної моделі формувалися таким чином:

- Забезпечення більш високого ступеня незалежності від даних. Прикладні програми не мають залежати від змін внутрішнього представлення даних, зокрема від змін організації файлів, перевпорядкування записів і шляхів доступу.
- Створення міцного фундаменту для вирішення семантичних питань, а також проблем несуперечності й надмірності даних. Зокрема, в статті Кодда вводиться поняття нормалізованих відносин, тобто відношень без повторюваних груп.
- Розширення мов керування даними за рахунок включення операцій над множинами.

Хоча інтерес до реляційної моделі зумовлений декількома різними причинами, все ж найбільш значні дослідження були проведені в рамках трьох проектів:

Перший з них розроблявся в кінці 70-х років у дослідницькій лабораторії корпорації IBM в місті Сан-Хосе, штат Каліфорнія, під керівництвом Астрахана, результатом якого стало створення системи під назвою "System R", яка стала прототипом справжньої реляційної СКБД. Цей проект задумано з метою отримати реальні докази практичності використання реляційної моделі за допомогою реалізації передбачених нею структур даних і операцій. Цей проект також зарекомендував себе як найважливіше джерело інформації про такі проблеми реалізації, як керування паралельністю, оптимізація запитів, керування транзакціями, забезпечення безпеки й цілісності даних, технологію відновлення, врахування людського фактора та розробку інтерфейсу користувача. Виконання проекту стимулювало публікацію багатьох науково-дослідних статей і створення інших прототипів реляційних СКБД. Зокрема, робота над проектом System R дала поштовх проведенню таких найважливіших розробок, як:

- створення мови структурованих запитів SQL, який набув статусу формального стандарту ISO (International Organization for Standardization) і в даний час є фактичним галузевим стандартом мови реляційних СКБД;
- створення різних комерційних реляційних СКБД, які вперше з'явилися на ринку на початку 80-х років 20-го сторіччя, таких як DB2 та SQL/DS корпорації IBM, а також ORACLE корпорації ORACLE Corporation.

Другим проектом, який зіграв помітну роль в розробці реляційної моделі даних, був проект INGRES (Interactive Graphics Retrieval System), робота над яким проводилася в Каліфорнійському університеті (місто Берклі) майже в той же самий час, що і над проектом System R. СКБД INGRES концентрувалась на тих же загальних цілях, що й проект System R. Результатом цих досліджень стала поява академічної версії INGRES, яка зробила значний внесок у загальне визнання реляційної моделі даних. Пізніше результатом досліджень даного проекту стала поява комерційних

продуктів INGRES фірми Relational Technology Inc. (зараз CA-OpenIngress фірми Computers Associates) і Intelligent Database Machine фірми Britton Lee Inc.

Третім проектом була система Peterlee Relational Test Vehicle наукового центру корпорації IBM, розташованого в місті Потерли, Великобританія. Цей проект був більш теоретичним, ніж проекти System R і INGRES. Його результати мали принципове значення, особливо в таких питаннях як обробка запитів і оптимізація, а також функціональне розширення системи.

Комерційні системи на основі реляційної моделі даних почали з'являтися в кінці 70-х - початку 80-х років. У наш час існує декілька сотень типів різних реляційних СКБД як для мейнфреймів, так і для мікрокомп'ютерів, хоча багато з них не повністю задовольняють точному визначенню реляційної моделі даних. Прикладами реляційних СКБД для мікрокомп'ютерів є СКБД Access і FoxPro фірми Microsoft, Paradox та Visual dBase фірми Borland.

Завдяки популярності реляційної моделі даних багато не реляційних систем використовують реляційний, призначений для користувача інтерфейс, незалежно від використовуваної базової моделі. Основна мережева СКБД, система IDMS (або IDMS/R, або IDMS/SQL) підтримує реляційне представлення даних. Іншими СКБД для мейнфреймів, в яких підтримуються деякі реляційні компоненти, є Model 204 фірми Computer Corporation of America і ADABAS D фірми Software AG.

Крім того, пізніше було запропоновано деякі розширення реляційної моделі даних, призначені для найбільш повного і точного вираження сенсу даних (Codd, 1979), об'єктно-орієнтованих понять (Stonebraker and Row, 1986), а також підтримки процесів прийняття рішень (Gardarin and Valduries, 1989).

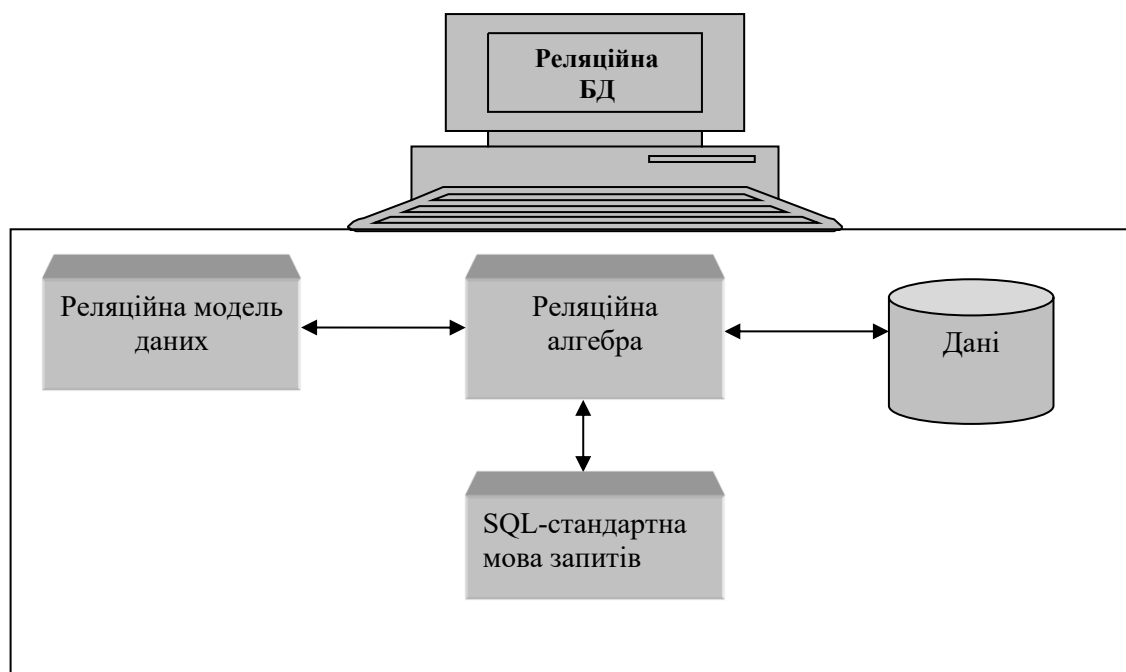


Рис. 1.2. Структура обробки інформації в реляційній БД

Використана термінологія:

Реляційна модель заснована на математичному понятті відношення, фізичним представленням якого є *таблиця*.

Кодд використав математичну термінологію, зокрема з теорії множин і логіки предикатів (рис. 1.3).



Рис. 1.3. Структура реляційних даних

Відношення – це плоска таблиця, що складається зі стовпців та рядків.

У будь-якої реляційній СКБД передбачається, що користувач сприймає базу даних як набір таблиць. Однак слід підкреслити, що це сприйняття стосується тільки логічної структури бази даних. Подібне представлення не відноситься до фізичної структури бази даних, яку реалізовано за допомогою різних структур зберігання.

Атрибут – це поименований стовпець відношення.

У реляційній моделі відношення використовують для зберігання інформації про об'єкти, представлені в базі даних. Відношення зазвичай має вигляд двовимірної таблиці, в якій рядки відповідають окремим записам, а стовпці - атрибутам. При цьому атрибути можуть розташовуватися у будь-якому порядку, незалежно від їх перевпорядкування, відношення буде залишатися одним і тим самим, а тому мати таке ж значення.

Наприклад, інформація про студентів може бути представлена відношенням **Студент**, що включає стовпці з атрибутами *Номер заліковки*, *Номер групи*, *Прізвище*, *Середній бал*. Аналогічно, інформація про викладачів може бути представлена відношенням **Викладач**, що включає стовпці з атрибутами *Табельний номер*, *Прізвище*, *Робочий телефон\факс*. На рис. 1.4. показано приклади відношень **Студент** і **Викладач**.

Студент

Номер заліковки	Номер групи	Прізвище	Середній бал
T33106	T331	Іванов	5
T33217	T332	Петров	4

TM3115	TM31	Пушников	3,5
TM3220	TM32	Сидоров	4,5

Викладач

Табельний номер	Прізвище	Роб. телефон	Факс
4587	Бондаренко	456-37-85	456-86-27
2136	Воронін	474-96-35	474-85-12
5496	Анісімова	468-05-38	468-28-01

Рис. 1.4. Приклади відношень **Студент і Викладач**.

Домен – набір допустимих значень для одного або декількох атрибутів.

Домени є надзвичайно потужними компонентами реляційної моделі. Кожен атрибут реляційної бази даних визначається на деякому домені, домени можуть відрізнятися для кожного з атрибутів, але два і більше атрибутів можуть визначатися на одному й тому ж домені.

У табл. 1.1 представлені домени для атрибутів відношень **Студент і Викладач**. Хоча у відношенні **Викладач** є чотири атрибути, тут показані тільки три, тому що два атрибути, *Робочий телефон і Факс*, визначені на одному й тому ж домені.

У будь-який момент часу в доменах можуть існувати значення, які не будуть реально представлені значеннями відповідного атрибута.

Таблиця 1.1. Домени атрибутів відношень **Студент і Викладач**

Атрибут	Ім'я домену	Вміст домену	Визначення домену
Номер заліковки	НОМЕР_ЗАЛІКОВКИ	Множина всіх допустимих номерів заліковок студентів	Символьний: розмір 6, діапазон 'T33101'-'T33111'
Номер групи	НОМЕР_ГРУПИ	Множина всіх наявних груп	Символьний: розмір 4, діапазон 'T331'-'T331'
Прізвище	ПРИЗВИЩЕ_СТУДЕНТА	Множина всіх прізвищ студентів	Символьний: розмір 10
Прізвище	ПРИЗВИЩЕ_ВИКЛАДАЧА	Множина всіх прізвищ викладачів	Символьний: розмір 10
Середній бал	СЕРЕДНІЙ_БАЛ	Множина всіх допустимих середніх балів студентів	Символьний: розмір 3
Табельний номер	ТАБЕЛЬНИЙ_НОМЕР	Множина всіх табельних номерів викладачів	Символьний: розмір 4

Робочий телефон	РОБОЧИЙ_ТЕЛЕФОН_ФАКС	Множина всіх номерів телефонів і факсів викладачів	Символьний: розмір 7
Факс	РОБОЧИЙ_ТЕЛЕФОН_ФАКС	Множина всіх номерів телефонів і факсів викладачів	Символьний: розмір 7

Поняття домену має суттєве значення, оскільки завдяки йому користувач може централізовано визначати зміст і джерело значень, які можуть отримувати атрибути.

У результаті *при виконанні реляційної операції системі доступно більше інформації, що дозволяє їй уникнути семантично некоректних операцій.*

Наприклад, безглуздо порівнювати назву вулиці з номером телефону, навіть якщо для обох цих атрибутів визначеннями доменів є символьні рядки. Однак множення значень з цих доменів є допустимою операцією.

Як впливає з цих двох прикладів, забезпечити повну реалізацію поняття домену зовсім непросто, а тому в багатьох реляційних СКБД вони підтримуються не в повному обсязі, а лише частково.

Кортеж – це рядок відношення.

Елементами відношення є кортежи або рядки таблиці.

Кортежі можуть розташовуватися у будь-якому порядку, при цьому відношення залишатиметься тим же самим, а значить і матиме таке ж саме значення.

Опис структури відношення разом зі специфікацією доменів та будь-якими іншими обмеженнями можливих значень атрибутів іноді називають його заголовком (або змістом (intension)). Зазвичай воно є фіксованим до тих пір доки сенс відношення не змінюється за рахунок додавання в нього додаткових атрибутів.

Кортежі називаються розширенням (extension), станом (state) або тілом відношення, яке постійно змінюється.

Ступінь відношення визначається кількістю атрибутів, яку вона містить.

Відношення Студент, показане на рис. 1.4, має чотири атрибути і, отже, його ступінь дорівнює чотирьом. Це означає, що кожен рядок таблиці є 4-арним кортежем, тобто кортежем, що містить чотири значення.

Відношення тільки з одним атрибутом має ступінь 1 і називається *унарним відношенням (або 1-арним кортежем)*. Відношення з двома атрибутами називається *бінарним*, відношення з трьома атрибутами – *тернарним*, а для відношень із більшою кількістю атрибутів використовується термін *n-арний*.

Визначення ступеня відношення є частиною заголовка відношення.

Кардинальність – це кількість кортежів, яка містить відношення.

Ця характеристика змінюється при кожному додаванні або видаленні кортежів. *Кардинальність є властивістю тіла відношення й визначається поточним станом відношення у довільно взятий момент.*

Реляційна база даних – це набір нормалізованих відношень.

Реляційна база даних складається з відношень, структура яких визначається за допомогою особливих методів, які називаються **нормалізацією**.

Альтернативна термінологія

Термінологія, яка використовується в реляційній моделі, часом може призвести до плутанини, оскільки крім запропонованих двох наборів термінів існує ще один - третій.

Відношення у ньому називається *файлом*, *кортежі* – *записами*, а *атрибути* – *полями*.

Ця термінологія заснована на тому факті, що фізично реляційна СКБД може зберігати кожне відношення в окремому файлі. У табл. 1.2 зображені відповідності, що існують між трьома згадуваними вище групами термінів.

Таблиця 1.2. Альтернативні варіанти термінів у реляційній моделі

Офіційні терміни	Альтернативний варіант 1	Альтернативний варіант 2
Відношення	Таблиця	Файл
Кортеж	Рядок	Запис
Атрибут	Стовпчик	Поле

Відношення і їх властивості в базі даних

Реляційна схема – це ім'я відношення, за яким слідує множина пар атрибутів і доменів.

Наприклад, для атрибутів $A_1, A_2, \dots, A_n$ з доменами $D_1, D_2, \dots, D_n$ реляційною схемою буде множина $\{A_1:D_1 \dots A_n:D_n\}$.

Відношення R , задане реляційною схемою S , є множиною відображень імен атрибутів на відповідні їм домени, тобто відношення R є множиною таких n -арних кортежів.

Кожен елемент n -арного кортежу складається з атрибута й значення цього атрибута.

Зазвичай при записі відношення у вигляді таблиці імена атрибутів перераховуються в заголовках стовпчиків, а кортежі утворюють рядки формату $(d_1, d_2, \dots, d_n)$, де кожне значення береться з відповідного домена.

Таким чином, в реляційній моделі відношення можна представити як довільну підмножину декартового добутку множин, тоді як таблиця - це всього лише фізичне уявлення такого відношення.

У прикладі, показаному на рис. 1.4. відношення Студент має атрибути Номер заліковки, Номер групи, Прізвище, Середній бал із відповідними їм доменами. Ставлення Студент є довільною підмножиною декартового добутку доменів або довільно множиною 4-арних кортежів, в яких першим йде елемент із домену НОМЕР_ЗАЛІКОВКИ, другим - елемент з домену НОМЕР_ГРУПИ і т.д. Наприклад, один з 4-арних кортежів може мати такий вигляд:

$\{(T33106, T331, \text{Іванов}, 5)\}$

Цей самий кортеж можна записати у більш коректній формі:

$\{(\text{Номер заліковки: 'T33106', Номер групи: 'T331', Прізвище: 'Іванов', Середній бал: '5'})\}$

Таблиця Студент представляє собою зручний спосіб запису всіх 4-арних кортежів, що утворюють відношення в певний заданий момент часу.

Відношення має такі *характеристики*:

- Відношення має ім'я, яке відрізняється від імен усіх інших відношень.

- Кожна комірка відношення містить тільки атомарне (неподільне) значення.
- Кожен атрибут має унікальне ім'я.
- Значення атрибута береться з одного й того самого домена.
- Порядок слідування атрибутів не має ніякого значення.
- Кожен кортеж є унікальним, тобто дублікатів кортежів не може бути.
- Теоретично порядок слідування кортежів у відношенні не має ніякого значення. (Однак, практично цей порядок може істотно вплинути на ефективність доступу до них).

Відношення не можуть містити груп, що повторюються.

Розглянемо відношення Студент (мал. 1.4.). Імена стовпців, зазначені в їх верхньому рядку, відповідають іменам атрибутів відношення. Значення атрибута Номер заліковки беруться з домену НОМЕР_ЗАЛІКОВКИ, не допускається розміщення в цьому стовпці інших значень, наприклад середнього бала.

Стовпці можна міняти місцями за умови, що ім'я атрибута переміщується разом з його значеннями.

Відношення не може містити кортежі-дублікати.

Наприклад, рядок ('ТЗ3106', 'ТЗ31', 'Іванов', '5') може бути представленим у відношенні тільки один раз.

При необхідності рядки можна міняти місцями довільним способом (наприклад, перемістити весь рядок, який відноситься до студента 'Іванов' на місце рядка студента 'Сидоров'), саме відношення при цьому залишиться попереднім.

Більша частина властивостей відношень походить від властивостей математичних відношень:

- Так як відношення є множиною, то порядок елементів не має значення. Із цього випливає, що порядок кортежів у відношенні не є важливим.
- У множині немає елементів, що повторюються. Аналогічно, відношення не може містити кортежі-дублікати.
- При обчисленні декартового добутку множин із простими однозначними елементами (наприклад, цілочисельними значеннями), кожний елемент в кожному кортежі має єдине значення. Так само, кожна комірка відношення містить тільки одне значення. Однак математичне відношення не потребує нормалізації. Кодд запропонував заборонити груп, що повторюються з метою спрощення реляційної моделі даних.
- Набір можливих значень для даної позиції відношення визначається множиною, або доменом, на якому визначається ця позиція. У таблиці всі значення у кожному стовпчику повинні походити від одного й того ж самого домену, відповідного для даного атрибута. Хоча у математичному відношенні порядок слідування елементів не має значення, у кортежі має значення.

Наприклад, допустима пара значень (1,2) абсолютно відмінна від допустимої пари (2,1). Це твердження не є правильним для відношень у реляційній моделі, де спеціально зазначається, що порядок атрибутів є несуттєвим. Справа в тому, що заголовки стовпців однозначно визначають, до якого саме атрибуту відноситься дане значення. Наслідком цього факту є положення про те, що порядок проходження заголовків стовпців у заголовку відношення є несуттєвим. Однак, якщо структура відношення вже визначена, то порядок елементів в кортежі тіла відношення повинен відповідати порядку імен атрибутів.

Реляційні ключі

Необхідно мати можливість унікальної ідентифікації кожного окремого кортежу відношення за значенням його атрибутів.

Суперключ (*superkey*) – атрибут або множина атрибутів, що єдиним образом ідентифікує кортеж даного відношення.

Розглянемо відношення Студент, показане на мал. 1.4. Конкретне значення атрибута Номер групи не може однозначно визначити студента, оскільки в групі вчиться певна кількість студентів. Даний атрибут не може бути обраний як суперключ, з іншого боку, оскільки в навчальному проекті для кожного студента задається номер його залікової книжки, кожне значення цього номера (атрибут Номер заліковки) може визначати не більш одного кортежу (щодо Студент), а тому Номер заліковки є суперключем як окремо, так і в поєднанні з іншими атрибутами.

Оскільки суперключ може містити додаткові атрибути, які є обов'язковими для унікальної ідентифікації кортежу, інтерес представляють суперключі, що складаються тільки з тих атрибутів, які дійсно необхідні для унікальної ідентифікації кортежів.

Потенційний ключ – це суперключ, який не містить підмножини, що також є суперключем даного відношення.

Таким чином, *потенційним* ключем для відношення Студент є Номер заліковки студента, що визначає його однозначно.

Потенційний ключ К для даного відношення R має дві властивості:

- **Унікальність.** У кожному кортежі відношення R значення ключа К єдиним способом ідентифікує цей кортеж.
- **Незвідність.** Ніяка допустима підмножина ключа К не має властивості унікальності.

Відношення може мати декілька потенційних ключів.

Якщо ключ складається з декількох атрибутів, то він називається первинним ключем.

Будь-який конкретний набір кортежів відношення не можна використовувати як доказ того, що якийсь атрибут або комбінація атрибутів є потенційним ключем.

Той факт, що в деякий момент часу не існує значень-дублікатів, зовсім не означає, що їх не може бути взагалі.

Наявність значень-дублікатів у конкретному існуючому наборі кортежів підтверджує той факт, що деяка комбінація атрибутів не може бути потенційним ключем.

Для ідентифікації потенційного ключа потрібно знати зміст використовуваних атрибутів в "реальному світі" тільки це дозволить обґрунтовано прийняти рішення про можливість існування значень-дублікатів. Тільки виходячи з подібної семантичної інформації можна гарантувати, що деяка комбінація атрибутів є потенційним ключем відношення.

На підставі представлених на рис. 1.4. даних, можна вирішити, що відповідним потенційним ключем для відношення Викладач цілком може бути атрибут Табельний номер викладача. Наприклад, Прізвище викладача не можна використовувати в якості потенційного ключа, оскільки можна зустріти кількох викладачів з таким прізвищем.

Первинний ключ – це потенційний ключ, що є обраним для унікальної ідентифікації кортежів усередині відношення.

Оскільки відношення не містить кортежі-дублікати, завжди можна унікальним чином ідентифікувати кожен його рядок.

Це означає, що відношення завжди має первинний ключ.

У гіршому випадку вся множина атрибутів може використовуватися як первинний ключ, але, зазвичай, щоб розрізнити кортежі, досить використовувати меншу підмножину атрибутів.

Потенційні ключі, які є обраними як первинний ключ, називаються **альтернативними ключами**.

Зовнішній ключ – це атрибут або множина атрибутів всередині відносини, що відповідає потенційному ключу деякого (може бути, того ж самого) відношення.

Якщо якийсь атрибут присутній в декількох аспектах, то його наявність зазвичай відображає певний зв'язок між кортежами цих відношень.

Розглянемо приклад. Нехай інформація про студентів представлена відношенням Студент, що містить стовпці з атрибутами Номер заліковки, Номер групи, Прізвище, Код предмету, який вивчає студент. Аналогічно, інформація про предмети може бути представлена відношенням Предмет, що містить стовпці з атрибутами Код предмету, Назва предмета, Лекційні години, Практичні години. На мал. 1.5. показані приклади відношень Студент і Предмет.

Студент

Номер заліковки	Номер групи	Прізвище	Код предмету
T33106	T331	Іванов	101
T33217	T332	Петров	102
TM3115	TM31	Пушніков	101
TM3220	TM32	Сидоров	103

Предмет

Код предмету	Назва предмету	Лекційні години	Практичні години
101	Фізика	18	26
102	Математика	25	30
103	Хімія	15	20

Рис. 1.5. Приклад відношень Студент і Предмет

Атрибут Код предмету навмисно міститься у відношенні Студент і Предмет для встановлення зв'язку між відомостями про студентів і предмети. У відношенні Предмет атрибут Код предмету є *первинним ключем*, а в відношенні Студент він введений для визначення вивчення студентами предметів. У відношенні Студент атрибут Код предмету є *зовнішнім ключем*. У такому випадку говорять, що атрибут Код предмета у відношенні Студент посилається на первинний ключ, тобто на атрибут Код предмета в базовому відношенні Предмет.

Базове відношення іноді називають **цільовим відношенням**.

Мережева модель даних

У мережевій моделі дані представлені у вигляді колекцій записів, а зв'язки – у вигляді наборів.

На відміну від реляційної моделі, зв'язки тут явним чином моделюються наборами, які реалізуються з допомогою вказівників.

Мережеву модель можна представити як граф із записами у вигляді вузлів графа і наборами у вигляді його ребер.

Ієрархічна модель даних

Ієрархічна модель є обмеженим підтипом мережевої моделі. У ній дані також представлені як колекції записів, а зв'язки – як набори.

Однак в ієрархічній моделі вузол може мати тільки один шлях утворення.

Ієрархічна модель може бути представлена як деревовидний граф із записами у вигляді вузлів (які також називаються сегментами) і множинами у вигляді ребер.

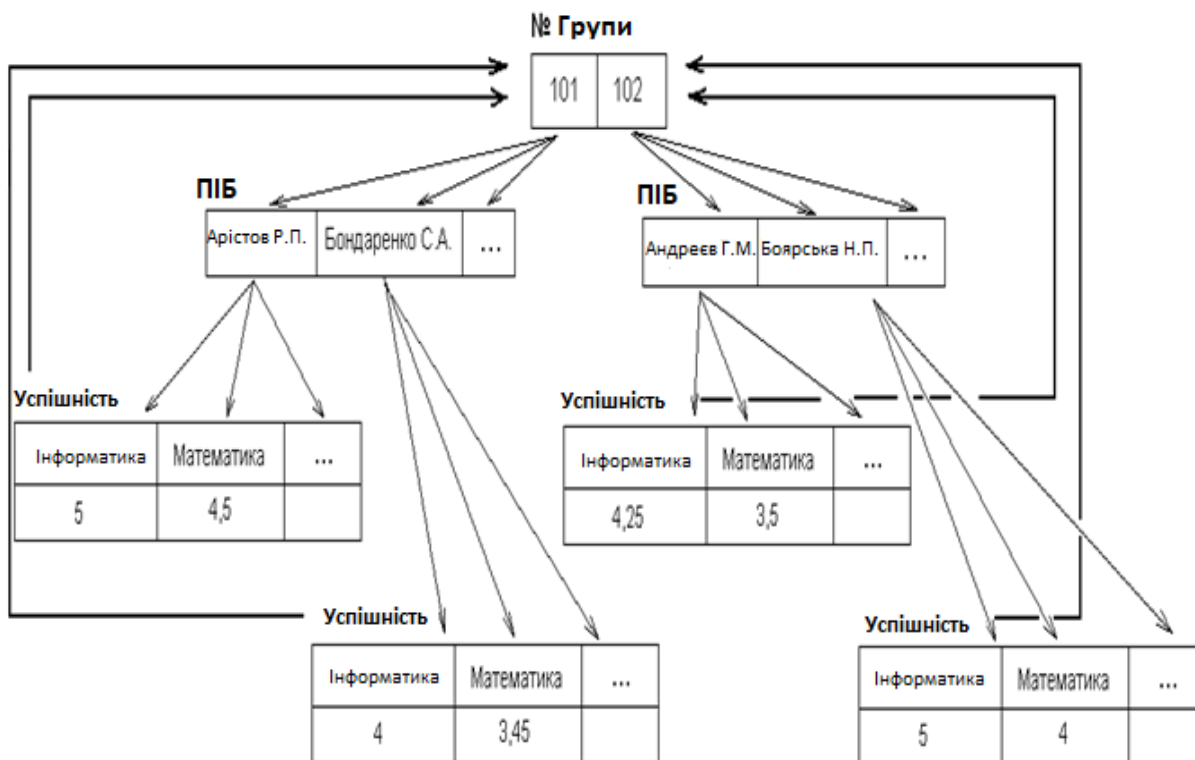


Рис. 1.6. Приклад мережевої моделі даних, що описана за допомогою СКБД

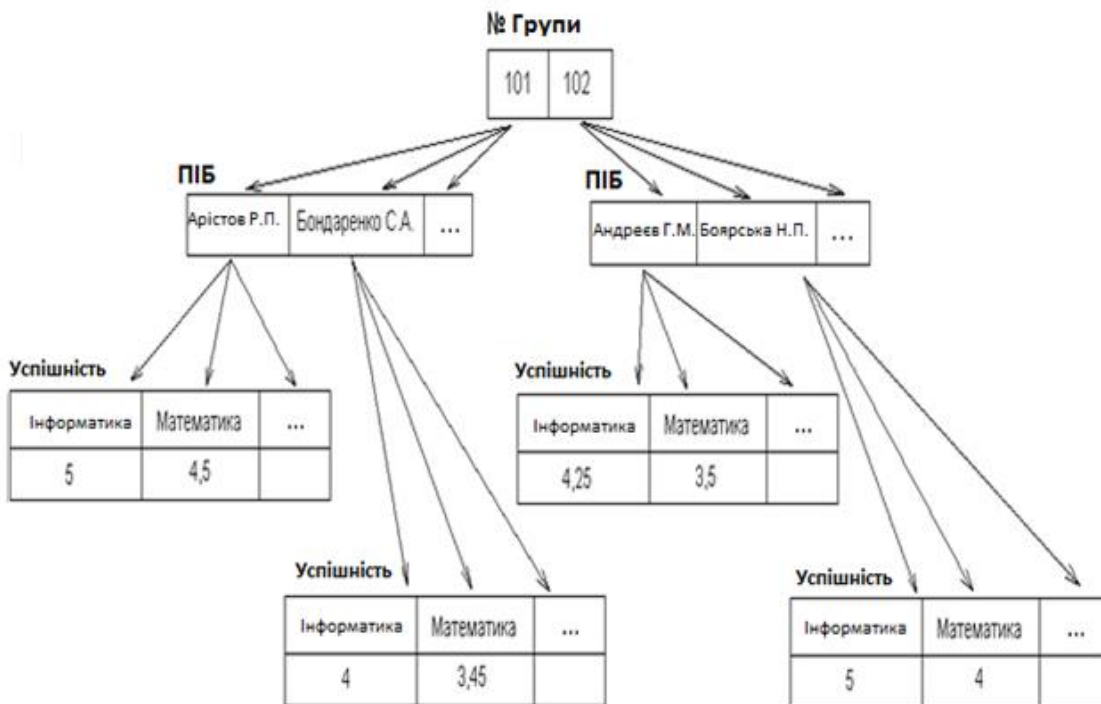


Рис. 1.7. Приклад мережевої моделі даних, що описана за допомогою СКБД

Засновані на записях (логічні) моделі даних використовуються для визначення загальної структури бази даних і високорівневого опису її реалізації.

Їх головний *недолік* полягає в тому, що вони *не дають адекватних засобів для явного вказання на обмеження, що накладаються на дані*.

У той же час в *об'єктних моделях даних відсутні засоби вказання на логічної структури*, але за рахунок надання користувачеві дається можливість вказати обмеження для даних, вони *дозволяють більшою мірою представити семантичну суть інформації, що зберігається*.

Більшість сучасних комерційних систем засновані на реляційній парадигмі, тоді як найперші системи баз даних будувалися на основі мережевої або ієрархічної моделі.

При використанні мережевої або ієрархічної моделі від користувача *вимагається знання фізичної організації бази даних*, до якої він повинен здійснювати доступ, у той час як при роботі з *реляційною моделлю незалежність від даних забезпечується значно більшою мірою*.

Отже, якщо в реляційних системах для обробки інформації в базі даних прийнятий *декларативний підхід* (тобто вони вказують, які дані слід вилучити), то в мережевих і ієрархічних системах - *навігаційний підхід* (тобто вони вказують, як їх слід вилучити).

Фізичні моделі даних

Фізичні моделі даних описують те, як дані зберігаються в комп'ютері, представляючи інформацію про структуру записів, їх впорядкованість та існуючих шляхах доступу.

Фізичних моделей даних не так багато, як логічних, а найпопулярнішими серед них є *узагальнююча модель (unifying model)* і *модель пам'яті кадрів (frame model)*.

Запитання:

- 1.1. Дайте визначення файловим системам.
- 1.2. Опишіть підхід, що використовується для обробки даних у файлових системах. Укажіть присутні обмеження.
- 1.3. Визначте термін база даних.
- 1.4. Опишіть основні характеристики підходу, що базується на використанні бази даних, і порівняйте їх із характеристиками звичайних файлових систем.
- 1.5. Визначте терміни сутність, атрибут, зв'язок.
- 1.6. Що тає перманентні дані? Сформулюйте термін база даних із допомогою поняття перманентності.
- 1.7. Опишіть типи інформації в базі даних.
- 1.8. Дайте визначення терміну СКБД. Опишіть основні можливості СКБД.
- 1.9. Що таке модель даних? Дайте визначення і поясніть сутність основних типів моделей даних.
- 1.10. Сформулюйте основні цілі створення реляційної моделі.
- 1.11. Опишіть структуру реляційних даних. Дайте визначення основним термінам: відношення, атрибут, домен, кортеж, ступінь відношення і кардинальність. Наведіть приклади.
- 1.12. У чому сутність альтернативної термінології?
- 1.13. Дайте визначення реляційній схемі та наведіть приклад.
- 1.14. Опишіть характеристики відношень.
- 1.15. Наведіть приклади класифікації реляційних ключів і дайте відповідні визначення. Поясніть сутність на прикладах.

Вправи:

- 1.1. Припустимо, що у вас є колекція записів класичної музики, що міститься на компакт-дисках, міні-дисках, довгограючих пластинках і аудіокасетах, і ви хочете створити базу даних, яка дозволить знаходити записи певного композитора (наприклад, Сібеліуса), диригента (наприклад, Симона Ратлам), соліста (наприклад, Артура Грюмікса), твори (наприклад, П'ятої симфонії Бетховена), оркестру (наприклад, Нью-Йоркського філармонічного), виду твору (Наприклад, концерту для скрипки) або оркестрової групи (наприклад, квартету Кронус). Накресліть діаграму «сутність-зв'язок» для цієї бази даних за зразком, поданим на мал. 1.1.
- 1.2. Створіть реляційну базу даних на будь-яку обрану вами тему, крім "Навчального процесу". За допомогою цієї бази проаналізуйте вивчений матеріал і визначте реляційні ключі.

ТЕМА 2

Реляційна модель.

Реляційна алгебра. Реляційне числення.

2.1 Реляційна модель

На сьогоднішній день реляційні СУБД стали домінуючим типом програмного забезпечення для обробки даних. Щорічний обсяг продажів в цьому секторі ринку оцінюється в 8-10 мільярдів доларів (або 2.5 мільярдів доларів разом з інструментами розробки), причому щорічний приріст цього обсягу становить 25%. Це програмне забезпечення являє собою друге покоління СУБД, засноване на використанні реляційної моделі даних, запропонованої Е.Ф.Коддом в 1970 році. У реляційній моделі всі дані логічно структуровані всередині відношень (таблиць). Кожне відношення має ім'я і складається з поіменованих атрибутів (стовпців) даних. Кожен кортеж (рядок) даних містить по одному значенню кожного з атрибутів. Велика перевага реляційної моделі полягає саме в цій простоті логічної структури. Хоча, звичайно ж, за цією простотою ховається серйозний теоретичний фундамент, якого не було у першого покоління СУБД (тобто у мережевих і ієрархічних СУБД).

Представлення схем в реляційній базі даних

Як було визначено в темі 1 **реляційна схема** - це ім'я відносини, за яким слідує безліч пар атрибутів і доменів, а **реляційна база даних** - це набір нормалізованих відносин.

Представлення - це динамічний результат однієї або декількох реляційних операцій над базовими відносинами з метою створення деякого іншого відношення.

У зв'язку з наявністю зазначених в темі 1 функціональних можливостей, СУБД стає надзвичайно корисним інструментом. Але оскільки для кінцевих користувачів не має значення, наскільки проста або складна внутрішня організація системи, можна почути заперечення, що СУБД ускладнює роботу, надаючи користувачам набагато більшу кількість даних, ніж їм дійсно потрібно. Як показано на малюнку, в підході, заснованому на використанні баз даних, необхідні співробітникам деканату докладні відомості про студентів (ПІБ, рік народження, адресу і т.д.).

Для вирішення проблеми надмірності даних, що представляються користувачам при роботі з ними, в СУБД пропонується інший механізм створення представлень (view), який дозволяє будь-якому користувачеві мати свій власний погляд на базу даних.

Мова DDL включає засоби визначення представлень, кожне з яких є деякою підмножиною бази даних.

Наприклад, можна організувати представлення, в якому співробітникам деканату будуть доступні тільки ті дані, які необхідні для ведення даних про успішність студентів.

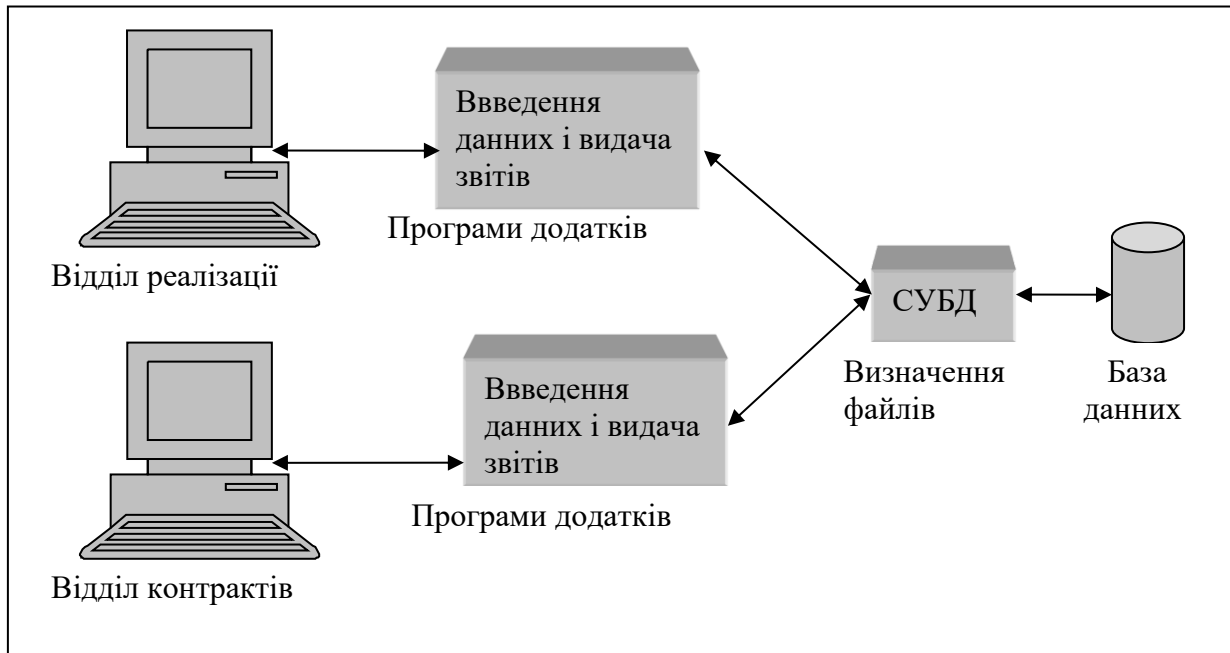


Рис. 2.1. Приклад організації доступу до інформації за допомогою СУБД

Крім спрощення роботи за рахунок надання користувачам тільки дійсно потрібних їм даних, уявлення мають деякі інші переваги:

- *Забезпечують додатковий рівень безпеки. Уявлення можуть створюватися з метою виключення тих даних, які не повинні бачити деякі користувачі. Наприклад, можна створити певне уявлення, яке дозволить ректорату переглядати всі дані про викладачів кафедри, включаючи відомості про їх зарплати. У той же час для організації доступу до даних інших користувачів, наприклад студентів, можна створити ще одне представлення, з якого всі відомості про зарплату будуть виключені.*
- *Надають механізм налаштування зовнішнього інтерфейсу бази даних. Наприклад, якщо мова йде про кваліфікацію викладачів, ректорат може працювати з полем «Вчений ступінь викладача кафедри», використовуючи для нього коротше і просте ім'я - «Вчений ступінь», вся інша інформація про викладача буде прихована.*
- *Дозволяють зберігати зовнішній інтерфейс бази даних не суперечним і незмінним навіть при внесенні змін до її структури. Наприклад, при додаванні або видаленні полів, зміні зв'язків, розбитті файлів, їх реорганізації або перейменування. Якщо в файл додаються або з нього видаляються поля, невикористовувані в деякому розумінні, то всі ці зміни на даному представлення ніяк не позначаються.*
- *Представлення забезпечує повну незалежність програм від реальної структури даних, що дозволяє усунути найважливіший недолік файлових систем.*

Реляційна цілісність

Модель даних має дві частини: керуючу частину, яка визначає типи допустимих операцій з даними, і набір обмежень цілісності, які гарантують коректність даних.

Ці дві частини моделі даних дозволяють встановлювати реляційні обмеження цілісності, а також виконувати - реляційні операції управління даними.

Існує три види цілісності даних:

- цілісність сутностей;
- посилальна цілісність;
- корпоративна цілісність (з точки зору сенсу предметної області)

Оскільки кожен атрибут пов'язаний з деяким доменом, для безлічі допустимих значень кожного атрибуту відносини визначаються так звані *обмеження домену*.

Два важливих правила цілісності (два правила реляційної моделі), які, по суті, є обмеженнями для всіх допустимих станів бази даних:

- *цілісність сутностей;*
- *посилальна цілісність.*

Визначник NULL

Визначник NULL вводить у зв'язку з підтриманням правил цілісності і вказує, що значення атрибута зараз невідомо чи неприйнятно для цього кортежу.

Визначник NULL слід сприймати як логічний величину "невідомо". Іншими словами, або це значення не входить в область визначення деякого кортежу, або ніяке значення ще не задано. Ключове слово NULL являє собою спосіб обробки неповних або незвичайних даних.

Однак визначник NULL не слід розуміти як нульове чисельне значення або заповнену пробілами текстовий рядок.

Нулі і пробіли являють собою деякі значення, тоді як ключове слово NULL покликане позначати відсутність будь-якого значення.

Застосування визначника NULL може викликати проблеми на етапі реалізації. Труднощі виникають через те, що реляційна модель заснована на обчисленні предикатів першого порядку, яке володіє двозначною чи булевою логікою, тобто допустимими є тільки два значення: істина і брехня.

Застосування визначника NULL означає, що доведеться вести роботу з логікою більш високого порядку, наприклад тризначною або навіть чотиризначною (Codd, 1986, 1987, 1990).

Використання поняття NULL в реляційній моделі є спірним питанням. Кодд (1990) розглядає поняття NULL як складову частину цієї моделі, а інші фахівці вважають цей підхід неправильним, вважаючи що проблема відсутньої інформації ще не до кінця зрозуміла, задовільне її рішення не знайдено, а тому включення визначника NULL в реляційну модель є передчасним (Date, 1995). Слід зазначити, що не у всіх реляційних системах підтримується робота з визначником NULL.

Цілісність сутностей

Перше обмеження цілісності стосується первинних ключів базових відносин.

Базове ставлення визначається як відношення, яке відповідає певній суті в концептуальній схемі.

Цілісність сутностей означає, що в базовому відношенні жоден атрибут первинного ключа не може містити відсутніх значень, які охоплюють визначником NULL.

За визначенням, первинний ключ - це мінімальний ідентифікатор, який використовується для унікальної ідентифікації кортежів. Це означає, що ніяка підмножина первинного ключа не може бути достатньою для унікальної ідентифікації кортежів.

Якщо допустити присутність визначника NULL в будь-якій частині первинного ключа, це рівноцінно твердженню, що не всі його атрибути необхідні для унікальної

ідентифікації кортежів, що суперечить визначенню первинного ключа. Наприклад, розглянемо стосунки Студент і Викладач.

Студент

Номер заліковки	Номер групи	Прізвище	Середній бал
T33106	T331	Іванов	5
T33217	T332	Петров	4
TM3115	TM31	Пушніков	3,5
TM3220	TM32	Сидоров	4,5

Викладач

Табельний номер	Прізвище	Роб. телефон	Факс
4587	Бондаренко	456-37-85	456-86-27
2136	Воронін	474-96-35	474-85-12
5496	Анісімова	468-05-38	468-28-01

Рис. 2.2. Приклади відношень Студент і Викладач.

Оскільки Номер заліковки є первинним ключем відношення Студент, то не можна допустити вставку в відношення Студент кортежу з визначником NULL в атрибуті Номер заліковки.

Якщо розглянути це правило більш уважно, то можна помітити кілька його незвичайних властивостей.

По-перше, чому воно застосовується тільки до первинних ключів, але не застосовується до альтернативних ключів?

По-друге, чому воно обмежується тільки базовими відносинами?

Наприклад, використовуючи дані відношення Викладач (рис. 2.2), розглянемо виконання наступного запиту: "Створити список табельних номерів всіх викладачів". Результатом цього запиту є унарне ставлення з єдиного атрибута Табельний номер. За визначенням, цей атрибут повинен бути первинним ключем. Припустимо, що серед його значень міститься визначник NULL. Оскільки це відношення не є базовим, реляційна модель допускає присутність визначника NULL в його первинному ключі.

Нещодавно було зроблено декілька спроб перевизначення цього правила (Codd, 1988; Date, 1990).

Цілісність за посиланням

Друге обмеження цілісності стосується зовнішніх ключів.

Якщо в відношенні існує зовнішній ключ, то значення зовнішнього ключа повинно або відповідати значенням потенційного ключа деякого кортежу в його базовому відношенні або задаватися визначником NULL.

Розглянемо відношення Студент і Предмет.

Студент

Номер заліковки	Номер групи	Прізвище	Код предмета
T33106	T331	Іванов	101
T33217	T332	Петров	102
TM3115	TM31	Пушніков	101
TM3220	TM32	Сидоров	103

Предмет

Код предмета	Назва Предмета	Лекційні години	Практичні години
101	Фізика	18	26
102	Математика	25	30
103	Хімія	15	20

Рис. 2.3. Приклад відношень Студент і Предмет

Атрибут Код предмета щодо Студент є зовнішнім ключем, який посилається на атрибут Код предмета базового відносини Предмет. Система повинна запобігати будь-яким спробам створити запис з інформацією про предмет з кодом '104' щодо Студент до тих пір, поки в відношенні Предмет не буде створена запис, що містить відомості про предмет з кодом '104'. Однак вважається допустимим створення запису з інформацією про новий предмет із зазначенням визначника NULL замість коду предмета, який вивчається студентом. Така ситуація може мати місце в тому випадку, коли введений новий предмет для вивчення, але ще інформація про даний предмет не визначена остаточно.

Корпоративні обмеження цілісності

Корпоративні обмеження цілісності даних – це додаткові правила підтримки цілісності, які визначаються користувачами або адміністраторами бази даних.

Користувачі самі можуть вказувати додаткові обмеження, якими повинні задовольняти дані. Наприклад, якщо викладач може викладати лише на одній кафедрі, то користувач може вказати це правило, а СУБД стежити за його виконанням. В цьому випадку щодо Викладач необхідно кожного викладача відносити тільки до однієї кафедри.

На жаль, рівень підтримки реляційної цілісності в різних системах істотно варіює.

2.2 Реляційні мови

Реляційна модель даних побудована таким чином, що одна з частин моделі даних є керуючою, тобто вона визначає типи допустимих операцій з даними,

включаючи операції оновлення та видалення даних, а також операції зміни структури бази даних.

Для управління відносинами в реляційних СУБД використовуються найрізноманітніші мови. Деякі з них є процедурними, тобто з їх допомогою користувач точно вказує системі, як слід маніпулювати даними. Інші мови є непроцедурного, тобто користувач вказує, які дані йому потрібні, а не як їх слід вилучати.

Розглянемо основи реляційної алгебри і реляційного числення, які запропоновані Коддом (1971) в якості основи для створення реляційних мов.

Реляційну алгебру можна описати як (високорівнева) процедурна мова, тобто та, яка може бути використана для того, щоб повідомити СУБД про те, як слід побудувати необхідне відношення на базі одного або декількох існуючих в базі даних відносин.

Реляційне числення являє собою непроцедурну мову, яку можна використовувати для визначення того, яким буде певне відношення, створене на основі одного або кількох інших відношень бази даних.

Реляційна алгебра і реляційне числення еквівалентні один одному, тобто для кожного виразу алгебри існує еквівалентний вираз в реляційному численні (і навпаки).

Реляційне числення використовується для оцінки виборчої потужності реляційних мов.

Реляційно-повна мова - мова, що дозволяє отримати будь-яке відношення, яке можна вивести за допомогою реляційного числення.

Більшість реляційних мов запитів є реляційно-повними. Однак у порівнянні з реляційною алгеброю і реляційним обчисленням реляційні мови запитів мають і інші, більш широкі функціональні можливості, оскільки в них передбачені додаткові операції, здатні виконувати обчислювальні, узагальнюючі і впорядковуючі функції.

Реляційна алгебра і реляційне числення є формальні, а не дружні користувачеві мови.

У реляційних базах даних реляційна алгебра і реляційне числення використовувалися в якості основи для розробки інших мов управління даними, більш високого рівня.

2.3 Реляційна алгебра

Реляційна алгебра - це теоретична мова операцій, яка на основі одного або декількох відношень дозволяє створювати інше відношення без зміни самих вихідних відносин.

Таким чином, обидві операнди і результат є відносинами, а тому результати однієї операції можуть стати вихідними даними для іншої операції. Це дозволяє створювати вкладені вирази реляційної алгебри точно так само, як створюються вкладені арифметичні вирази (властивість). Ця властивість називається замкнутістю, тобто відношення покриваються реляційною алгеброю так само, як числа - арифметичними операціями.

Реляційна алгебра є мовою послідовного використання відносин, в якому всі кортежі, можливо, взяті навіть з різних відносин, обробляються однією командою, без організації циклів. Для команд реляційної алгебри запропоновано кілька варіантів

синтаксису. Скористаємося загальноприйнятими символічними позначеннями для цих команд і представимо їх у неформальному вигляді.

Існує кілька варіантів вибору операцій, які включаються в реляційну алгебру. Початково Кодд запропонував вісім операторів, але згодом до них були додані і деякі інші.

П'ять основних операцій реляційної алгебри (виконують більшість операцій вилучення даних):

- вибірка (selection),
- проекція (projection),
- декартовий добуток (cartesian product),
- об'єднання (union),
- різниця (set difference).

На підставі п'яти основних операцій можна також вивести додаткові операції, такі як операції:

- з'єднання (join),
- перетину (intersection),
- ділення (division).

Функції цих операцій схематично показані на рис. 5.1.

Операції вибірки і проекції є унарними, оскільки вони працюють з одним ставленням. Інші операції працюють з парами відносин, і тому їх називають бінарними операціями.

У наведених нижче визначеннях S і R - це два відношення, визначені над атрибутами $A = (a_1, a_2, \dots, a_N)$; $B = (b_1, b_2, \dots, b_M)$ відповідно

Унарні операції реляційної алгебри

Операція вибірки (або обмеження) $\sigma_{\text{предикат}}(R)$ працює з одним відношенням R і визначає результуюче відношення з тим самим заголовком, що і відношення R , і тілом, що складається з кортежів, значення атрибутів яких при підстановці в умову (предикат σ) дають значення істина.

$\sigma_{\text{предикат}}(R)$ являє собою логічне вираження, до якого можуть входити атрибути відношення R і (або) скалярні вирази.

У найпростішому випадку умова (предикат) σ має вигляд $X \Theta Y$, де Θ – один з операторів порівняння ($=, \neq, <, >$ і т.д.), а X і Y – атрибути відношення R або скалярні значення. Такі вибірки називаються

Θ -вибірки (тета-вибірки) або Θ -обмеження, Θ -селекції.

Синтаксис операції вибірки: R where σ , или R where $(X \Theta Y)$.

Синтаксис на мові SQL: `select * from R where (X  $\Theta$  Y).`

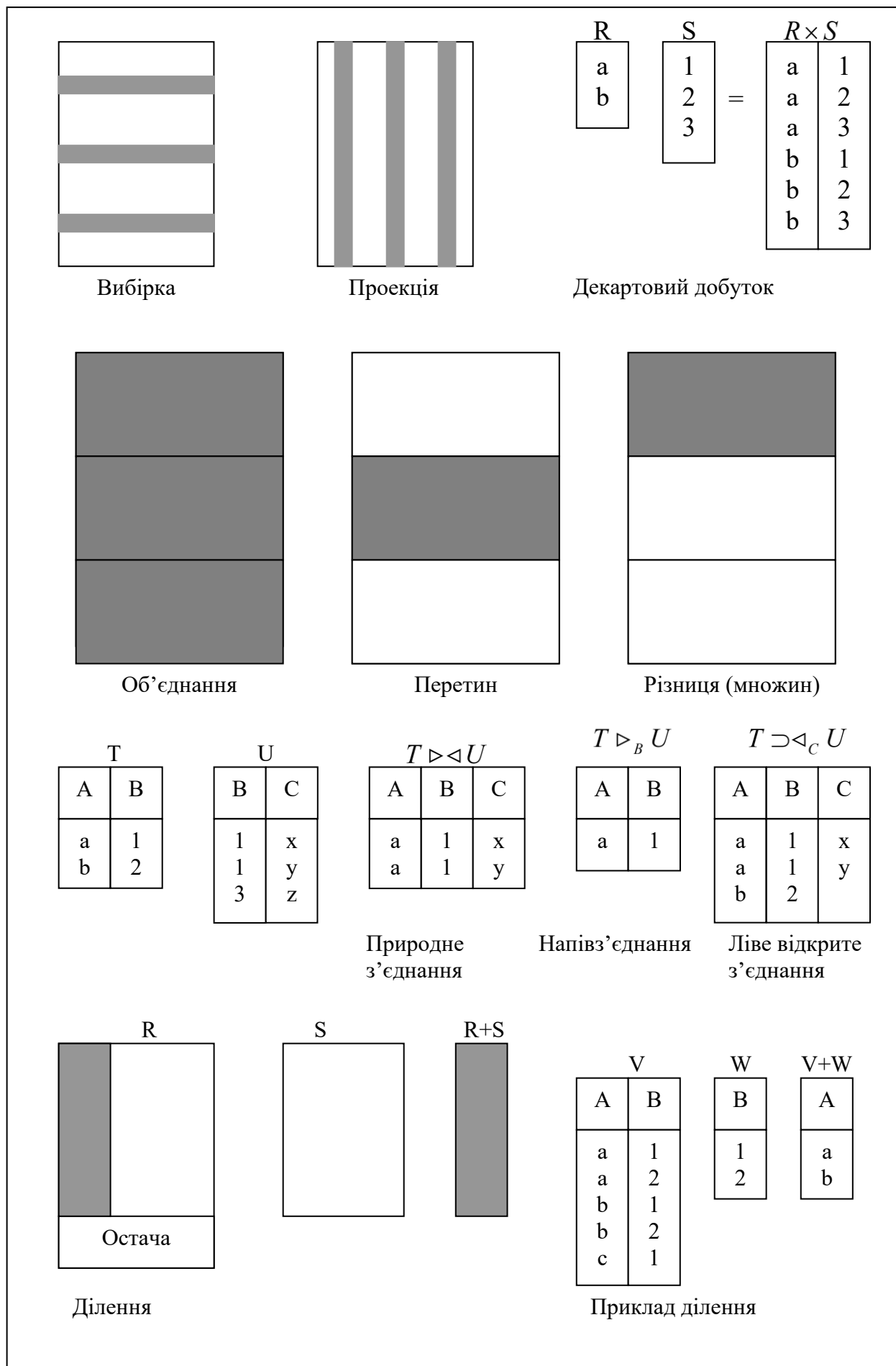


Рис 2.33.Схематичне представлення функцій операторів реляційної алгебри.

Нехай дано відношення R з інформацією про студентів:

<i>Номер студента</i>	Прізвище	Середній бал
6	Іванов	5
17	Петров	4
19	Сидоров	4,5

Результат вибірки R where Середній бал <5 буде мати вигляд:

<i>Номер студента</i>	Прізвище	Середній бал
17	Петров	4
19	Сидоров	4,5

Рис. 2.34. Приклад операції вибірки

Операція проєкції *Патр.1, ..., атр.n (R)* працює з одним відношенням R і визначає нове відношення, що містить вертикальну підмножину відношення R, створюване за допомогою вилучення значень зазначених атрибутів і виключення з результату рядків-дублікатів.

Синтаксис операції проєкції: R [X,...,Z]

Синтаксис на мові SQL: Select X, Y,..., Z from R.

Нехай дано відношення R з інформацією про викладачів:

Табельний номер	Прізвище	Предмет
4587	Бондаренко	Географія
2136	Воронін	Історія
5496	Анісімова	Історія
5496	Анісімова	Філософія

Проекція R [Предмет] матиме вигляд:

Предмет
Географія
Історія
Філософія

Рис. 2.35. Приклад операції проєкції

Операції з множинами

Декартовий добуток $R \times S$ визначає нове відношення, яке є результатом конкатенації (тобто зчеплення) кожного кортежу з відношення R з кожним кортежем з відношення S.

Тобто це таке відношення, заголовок якого є зчепленням заголовків відношень R і S : $(R_1, \dots, R_n, S_1, \dots, S_m)$, а тіло складається з кортежів, які є зчепленням кортежів відношень R і S : $(r_1, \dots, r_n, s_1, \dots, s_m)$, таких, що $(r_1, \dots, r_n) \in R$, $(s_1, \dots, s_m) \in S$.

Синтаксис операції декартового добутку: $R \text{ times } S$.

SQL запит буде виглядати дуже просто: `Select * from R, S.`

Потужність добутку $R \text{ times } S$ дорівнює добутку потужностей відношень R і S , тому що кожен кортеж відношення R з'єднується з кожним кортежем відношення S .

Оператор декартового добутку множить два відношення, що в результаті призводить до створення іншого відношення, що складається з усіх можливих пар кортежів обох відношень. Отже, якщо одне відношення має I кортежів і N атрибутів, а інше - J кортежів і M атрибутів, то відношення з їх декартових твором буде містити $(I \times J)$ кортежів і $(N \times M)$ атрибутів.

Вихідні відношення можуть містити атрибути з однаковими іменами. В такому випадку імена атрибутів будуть містити назви відношень у вигляді префіксів для забезпечення унікальності імен атрибутів в відношенні, отриманому як результат виконання операції декартового добутку.

Якщо у відношенні R і S є атрибути з однаковими найменуваннями, то перед виконанням операції декартового добутку такі атрибути необхідно перейменувати.

Нехай дано два відношення R і S з інформацією про студентів і предметах:

Відношення R (Студенти)

Номер студента	Прізвище
6	Іванов
17	Петров
19	Сидоров

Відношення S (Предмети)

Код предмета	Назва
101	Фізика
102	Математика
103	Інформатика

Декартовий добуток відношень R і S буде мати вигляд:

Відношення $R \text{ TIMES } S$

Номер студента	Прізвище	Код предмета	Назва
6	Іванов	101	Фізика
6	Іванов	102	Математика
6	Іванов	103	Інформатика
17	Петров	101	Фізика
17	Петров	102	Математика

17	Петров	103	Інформатика
19	Сидоров	101	Фізика
19	Сидоров	102	Математика
19	Сидоров	103	Інформатика

Рис. 2.36. Приклад операції декартового добутку

Операція об'єднання $R \cup S$ виходить в результаті конкатенації R і S , з утворенням одного відношення з тим же заголовком, що і у відношень R і S і тілом, що складається з кортежів, що належать або R , або S , або обом відношенням (з максимальною кількістю кортежів), якщо кортежі-дублікати виключені. При цьому відношення R і S повинні бути сумісними по об'єднанню.

Сумісність по об'єднанню: об'єднання відношень можливо тільки в тому випадку, якщо збігаються їх схеми, тобто якщо вони мають однакову кількість атрибутів з однаковими доменами. У деяких випадках для отримання двох сумісних по об'єднанню відношень може бути використана операція проєкції.

Синтаксис операції об'єднання: $R \text{ union } S$.

Синтаксис на мові SQL: $(\text{Select } * \text{ from } R) \text{ union } (\text{select } * \text{ from } S)$

Нехай дано два відношення R і S з інформацією про студентів: відношення R

Номер студента	Прізвище	Середній бал
6	Іванов	5
17	Петров	4
19	Сидоров	4,5

Відношення S

Номер студента	Прізвище	Середній бал
6	Іванов	5
18	Пушников	3,5
19	Сидоров	4,5

Об'єднання відношень R і S буде мати вигляд:

Номер студента	Прізвище	Середній бал
6	Іванов	5
17	Петров	4
19	Сидоров	4,5
18	Пушников	3,5

Рис. 2.37. Приклад операції об'єднання

Операція різниці RS визначає відношення з тим же заголовком, що і у відношень R і S , і тілом, що складається з кортежів, що належать відношенню R і не належать відношенню S , таких, які є у відношенні R , але відсутні в відношенні S . Причому відношення R і S повинні бути сумісними по об'єднанню.

Синтаксис операції віднімання: $R \text{ minus } S$.

Синтаксис на мові SQL: `(select * from R) except (select * from S)` (Деякі версії використовують оператор `minus` замість `except`).

Для двох відношень R і S віднімання має вигляд:

Відношення R

Номер студента	Прізвище	Середній бал
6	Іванов	5
17	Петров	4
19	Сидоров	4,5

Відношення S

Номер студента	Прізвище	Середній бал
6	Іванов	5
18	Пушников	3,5
20	Сидоров	4,5

Відношення $R \text{ MINUS } S$

Номер студента	Прізвище	Середній бал
17	Петров	4
19	Сидоров	4,5

Рис. 2.38. Приклад операції різниці

Перетин $R \cap S$ визначає відношення, яке містить кортежі, присутні як у відношенні R , так і у відношенні до S . Відношення R і S повинні бути сумісними по об'єднанню.

Іншими словами, перетином двох сумісних по типу відношень R і S називається відношення з тим же заголовком, що і у відношень R і S , і тілом, що складається з кортежів, що належать одночасно обома відношенням R і S .

Перетин можна сформулювати і на основі оператора різниці множин:

$$R \cap S = R - (R - S)$$

Синтаксис операції перетину: $R \text{ intersect } S$.

Синтаксис на мові SQL: `(Select * from R) intersect (select * from S)`.

Нехай дано два відношення R і S з інформацією про студентів:

відношення R

Номер студента	Прізвище	Середній бал

6	Іванов	5
17	Петров	4
19	Сидоров	4,5

Відношення S

Номер студента	Прізвище	Середній бал
6	Іванов	5
18	Пушников	3,5
20	Сидоров	4,5

Для відношень R і S перетин має вигляд:

Відношення R INTERSECT S

Номер студента	Прізвище	Середній бал
6	Іванов	5

Рис. 2.39. Приклад операції перетину

Операції з'єднання

Як правило, користувачів цікавить лише деяка частина всіх комбінацій кортежів декартового добутку, яка задовольняє заданій умові. Тому замість декартового добутку зазвичай використовується одна з найважливіших операцій реляційної алгебри - операція з'єднання.

Операція з'єднання, в результаті виконання якої на базі двох вихідних відношень створюється деяке нове відношення; вона еквівалентна операції вибірки з декартового добутку двох операндів-відношень тих кортежів, які задовольняють умові, вказаною в предикаті з'єднання як формули вибірки.

Предикат - це формула, що описує деякі логічні зв'язки і міркування.

З'єднанням відношень R і S за умовою (задовольняє предикату) F називається відношення (R times S) where F, де F є логічний вираз, до якого можуть входити атрибути відношень R і S і (або) скалярні вирази.

Синтаксис на мові SQL: Select R.*, S. from R, S where f.

Таким чином, операція з'єднання є результат послідовного застосування операцій декартового добутку і вибірки. Якщо у відношеннях R і S є атрибути з однаковими найменуваннями, то перед виконанням з'єднання такі атрибути необхідно перейменувати.

С точки зору ефективної реалізації в СУБД, ця операція є однією з найважчих і часто виявляється однією з основних причин, що викликають проблеми з продуктивністю, властиві всім реляційним системам.

Типи операцій з'єднання:

- тета-з'єднання;
- з'єднання по еквівалентності, яке є окремим видом тета-з'єднання;
- природне з'єднання;
- зовнішнє з'єднання;

- напівз'єднання.

Операція тета-з'єднання (Θ – join) $R \bowtie_F S$ визначає відношення, яке містить кортежі з декартового добутку відношень R і S , що задовольняють предикату F .

Предикат F має вигляд $R.a_i \Theta S.b_j$, де замість b може бути вказана одна з операцій порівняння ($<$, $\leq$, $>$, $\geq$, или $\sim$).

Позначення тета-з'єднання можна переписати на основі базових операцій вибірки і декартового добутку наступним чином: $R \bowtie_F S = \sigma_F(R \times S)$

Іншими словами операцію тета-з'єднання можна визначити: Нехай відношення R містить атрибут X , відношення S містить атрибут Y , а Θ - один з операторів порівняння ($=$, $<$, $>$ і т.д.). Тоді Θ - з'єднанням відношення R по атрибуту X зі ставленням S по атрибуту Y називають відношення $(R \text{ times } S) \text{ where } (X \Theta Y)$.

Синтаксис на мові SQL: `Select * from R, S where (R.X  $\Theta$  S.Y).`

Іноді, для операції $\square$ -з'єднання застосовують наступний, більш короткий синтаксис: $R[X \Theta Y]S$

Розглянемо деяку базу даних, яка містить інформацію про викладачів і предмети. Нехай викладачам і предметам присвоєно якийсь статус. Нехай навчальний процес організований таким чином, що викладачі мають право викладати тільки ті предмети, статус яких не вище статусу викладача (сенса цього може бути в тому, що хороший викладач з високим статусом може викладати більшу кількість предметів).

Відношення K (Викладачі)

Табельний номер	Прізвище	X (Статус викладача)
4587	Бондаренко	4
2136	Воронін	1
5496	Анісимова	2

Відношення S (Предмети)

Код предмета	Назва	Y(Статус предмета)
101	Історія	3
102	Географія	2
103	Філософія	1

Відповідь на питання: "Які викладачі мають право викладати якісь предмети?", Дає Θ - з'єднання $R[X \geq Y]S$:

Відношення "Які викладачі викладають які предмети"

Табельний номер	Прізвище	X (Статус викладача)	Код предмета	Назва	Y (Статус предмета)
4587	Бондаренко	4	101	Історія	3
4587	Бондаренко	4	102	Географія	2
4587	Бондаренко	4	103	Філософія	1

2136	Воронін	1	103	Філософія	1
5496	Анісімова	2	102	Географія	2
5496	Анісімова	2	103	Філософія	1

Рис. 2.40. Приклад операції тета-з'єднання

З'єднання по еквівалентності (equi-join) - окремий випадок

Θ-з'єднання, коли предикат F містить тільки операцію порівняння з рівності (=).

Синтаксис екві-з'єднання: $R[X=Y]S$.

Синтаксис на мові SQL: `select R.*, S. from R, S where (R.X = S.Y).`

Нехай є відношення S - студенти, P - предмети і SP - вивчення, що зберігають інформацію про студентів, предмети і про вивчення предметів даними студентами (для зручності введемо короткі найменування атрибутів):

Відношення S (Студенти)

Номер студента SNUM	Прізвище студента SNAME
6	Іванов
17	Петров
19	Сидоров

Відношення P (Предмет)

Код предмета PCOD	Назва предмета PNAME
101	Фізика
102	Математика
103	Інформатика

Відношення SP (Вивчення)

Номер студента SNUM	Код предмета PCOD	Середній бал по предмету SRBALL
6	101	4,5
6	102	4

6	103	5
17	101	3,5
17	102	4
19	101	4,5

Відповідь на питання: "Які предмети вивчаються студентами?", дає еквівалентне $S[SNUM = SNUM]SP$. Насправді, тому що у відношеннях є однакові атрибути, то потрібно спочатку перейменувати атрибути, а потім виконати еквівалентні-з'єднання. Запис стає більш громіздким: $(S \text{ rename } SNUM \text{ as } SNUM1)[SNUM1 = SNUM2](SP \text{ rename } SNUM \text{ as } SNUM2)$. Зазвичай, такою складною формою запису не користуються. В результаті маємо відношення:

Відношення "Які предмети вивчаються якими студентами?"

Номер студента SNUM1	Прізвище студента SNAME	Номер студента SNUM2	Код предмета PCOD	Середній бал по предмету SRBALL
6	Іванов	6	101	4,5
6	Іванов	6	102	4
6	Іванов	6	103	5
17	Петров	17	101	3,5
17	Петров	17	102	4
19	Сидоров	19	101	4,5

Рис. 2.41. Приклад операції з'єднання по еквівалентності

Недоліком еквівалентні-з'єднання є те, що якщо з'єднання відбувається по атрибутам з однаковими найменуваннями (а так найчастіше і відбувається!), То в результаті відношенні з'являється два атрибути з однаковими значеннями.

Позбутися від цього недоліку можна, взявши проекцію по всіх атрибутів, крім одного з дублюючих. Саме так діє природне з'єднання.

Природне з'єднання $R \bowtie S$ – з'єднання по еквівалентності двох відношень R і S , виконане за всіма спільними атрибутам, з результатів якого виключається по одному примірникові кожного загального атрибута.

Таким чином, нехай дано відношення $R(R_1, \dots, R_n, X_1, \dots, X_p)$ і $S(X_1, \dots, X_p, S_1, \dots, S_m)$, мають однакові атрибути $X_1, \dots, X_p$ (тобто атрибути з однаковими іменами і визначені на однакових доменах). Тоді природним з'єднанням відношень R і S називається відношення з заголовком $R(R_1, \dots, R_n, X_1, \dots, X_p, S_1, \dots, S_m)$, і тілом, що містить безліч кортежів $(r_1, \dots, r_n, x_1, \dots, x_p, s_1, \dots, s_m)$, таких, що $(r_1, \dots, r_n, x_1, \dots, x_p) \in R$ і $(x_1, \dots, x_p, s_1, \dots, s_m) \in S$.

Синтаксис: $R \text{ join } S$.

Ступенем природного з'єднання називається сума ступенів операндів відношень R і S мінус кількість атрибутів x .

У синтаксисі природного з'єднання не вказуються, за якими атрибутам здійснюється з'єднання. Природне з'єднання проводиться за всіма однаковим атрибутам.

Природне з'єднання еквівалентно такій послідовності реляційних операцій:

- перейменувати однакові атрибути у відношеннях;
- виконати декартовий добуток відносин;
- виконати вибірку за співпадаючими значеннями атрибутів, які мали однакові імена;
- виконати проекцію, видаливши повторювані атрибути;
- перейменувати атрибути, повернувши їм початкові імена.

У попередньому прикладі відповідь на питання: "Які предмети вивчаються, якими студентами?", Більш просто записується у вигляді природного з'єднання трьох відношень $S \text{ join } SP \text{ join } P$ (для зручності перегляду порядок атрибутів змінений, це є допустимим за властивостями відносин):

Відношення $S \text{ JOIN } SP \text{ JOIN } P$

Номер студента SNUM	Прізвище студента SNAME	Код предмета PCOD	Назва предмета PNAME	Середній бал по предмету SRBALL
6	Іванов	101	Фізика	4,5
6	Іванов	102	Математика	4
6	Іванов	103	Інформатика	5
17	Петров	101	Фізика	3,5
17	Петров	102	Математика	4
19	Сидоров	101	Фізика	4,5

Рис. 2.42. Приклад природного з'єднання

Найчастіше при з'єднанні двох відношень кортеж з одного відношення не знаходить відповідного кортежу в іншому відношенні. Інакше кажучи, в стовпцях з'єднання виявляються неспівпадаючі значення. Може знадобитися, щоб рядок з одного відношення був представлена в результаті з'єднання, навіть якщо в іншому відношенні немає збігається значення. Ця мета може бути досягнута за допомогою зовнішнього з'єднання.

Зовнішнє з'єднання буває ліве і праве.

Лівим зовнішнім з'єднанням $R \supset \triangleleft S$ називається з'єднання, при якому кортежі відношення R , що не мають співпадаючих значень в загальних стовпцях відношення S , також включаються в результуюче відношення.

Для позначення відсутніх значень у другому відношенні використовується визначник NULL.

Перевагою зовнішнього з'єднання є те, що при такому з'єднанні зберігається вихідна інформація, тобто зовнішнє з'єднання зберігає кортежі, які були б втрачені при використанні інших типів з'єднання.

Студент може брати участь в олімпіадах з предметів, встановлений загальний бал яких не більше середнього бала студента. На основі відношень R і S створити список, в якому вказані студенти і предмети, за якими вони беруть участь в олімпіадах.

$(\Pi(R)) \supset \triangleleft S$

Результат цієї операції показаний в табл. А.

Відношення R

Номер студента	Прізвище	Середній бал
6	Іванов	5
17	Петров	3
19	Сидоров	4

Відношення S

Код предмета	Назва	Загальний бал
101	Фізика	4,5
102	Хімія	4

Таблиця А

Номер студента	Прізвище	Середній балл	Код предмета	Назва	Загальний бал
6	Іванов	5	101	Фізика	4,5
6	Іванов	5	102	Хімія	4
17	Петров	3	NULL	NULL	NULL
19	Сидоров	4	102	Хімія	4

Рис. 2.43. Приклад лівого зовнішнього з'єднання

Праве зовнішнє з'єднання - з'єднання, в результуючому відношенні якого містяться всі кортежі правого відношення.

Повне зовнішнє з'єднання - з'єднання, в результуюче відношення якого поміщаються всі кортежі з обох відношень і в якому для позначення незбіжних значень кортежів використовуються визначники NULL.

Напівз'єднання $R \supset_F S$ визначає відношення, яке містить ті кортежі відношення R, які входять в з'єднання відношень R і S.

Перевага напівз'єднання полягає в тому, що воно дозволяє скоротити кількість кортежів, які потрібно обробити для отримання з'єднання. Ця особливо корисно при обчисленні з'єднань в розподілених системах. Операцію напівз'єднання можна сформулювати і з допомогою операторів проекції і з'єднання:

$$R \supset_F S = \Pi_A(R \supset \triangleleft_F S)$$

Тут A - це набір всіх атрибутів по відношенню R.

Номер студента	Прізвище	Середній бал	Код предмета	Назв	Загальний бал
6	Іванов	5	101	Фізика	4,5
6	Іванов	5	102	Хімія	4
19	Сидоров	4	102	Хімія	4

Рис. 2.44. Приклад напівз'єднання

Ділення

Нехай відношення R визначено на безлічі атрибутів A , а відношення S - на безлічі атрибутів B , причому $B \subseteq A$ (Тобто B є підмножиною A). Нехай $C = A - B$, тобто C є безліччю атрибутів відношення R , які не є атрибутами відношення S .

Ділення $R \div S$ – результатом є набір кортежів відношення R (визначено на безлічі атрибутів A), визначених на безлічі атрибутів C ($C = A - B$), які відповідають комбінації всіх кортежів відношення S (визначено на безлічі атрибутів B).

Цей оператор можна сформулювати і на основі інших основних операторів:

$$T_1 = \Pi_c(R)$$

$$T_2 = \Pi_c((S \times T_1) - R)$$

$$T = T_1 - T_2$$

Приклад.

Відношення R

Номер групи	Кількість студентів	ПІБ куратора	Код предмета	Назва предмета
ТМ-31	20	Іванов	01	Математика
ТМ-32	22	Петров	01	Математика
ТІ-31	13	Сидоров	01	Математика
ТМ-31	20	Іванов	02	Фізика
ТМ-32	22	Петров	02	Фізика

Відношення S

Номер групи	Кількість студентів	ПІБ куратора
ТМ-31	20	Іванов
ТМ-32	22	Петров
ТІ-31	13	Сидоров

T_1 : Select 'Код предмета', 'Назва предмету' from R

Код предмета	Назва предмета
01	Математика
02	Фізика

T_{21} : Select * from T_1, S

Код предмета	Назва предмета	Номер групи	Кількість студентів	ПІБ куратора
01	Математика	ТМ-31	20	Іванов
01	Математика	ТМ-32	22	Петров
01	Математика	ТІ-31	13	Сидоров
02	Фізика	ТМ-31	20	Іванов
02	Фізика	ТМ-32	22	Петров
02	Фізика	ТІ-31	13	Сидоров

T_{22} : (Select * from T_{21}) except (Select * from R)

Код предмета	Назва предмета	Номер групи	Кількість студентів	ПІБ куратора
02	Фізика	ТІ-31	13	Сидоров

T_2 : Select 'Код предмета', 'Название предмета' from T_{22}

Код предмета	Назва предмета
02	Фізика

$$P=T_1-T_2$$

Код предмета	Назва предмета
01	Математика

Рис. 2.45. Приклад операції ділення

2.4 Реляційне числення

У виразах реляційної алгебри завжди явно задається якийсь порядок, а також мається на увазі якась стратегія оцінки запиту. У реляційному численні не існує ніякого опису оцінки запиту, оскільки в запиті реляційного числення вказується, що слід витягти, а не як.

Реляційне числення ніяк не пов'язане з диференціальним і інтегральним численнями в математиці, а його назва походить від частини символічної логіки, яка називається обчисленням предикатів.

В контексті баз даних воно існує в двох формах:

1. У формі запропонованого Коддом реляційного числення кортежів.
2. У формі запропонованого Лакруа та Піро реляційного числення доменів.

Предикат в логіці першого порядку (або теорії числення предикатів) - істиннісна функція з параметрами.

Судження - вираз, який приймає функція після підстановки значень замість параметрів. Може бути істинним або хибним.

Наприклад, пропозиції "Іванов є студентом даного інституту" і "Стипендія Іванова вище, ніж у Петрова" є судженнями, оскільки можна визначити їх істинність або хибність. У першому випадку функція "є студентом даного інституту" має один аргумент ("Іванов"), а в другому випадку функція "стипендія вище" має, два аргументи ("Іванов" і "Петров").

Якщо предикат містить змінну, то у цієї змінної повинна бути відповідна область визначення.

При підстановці замість змінної x одних значень з її області визначення судження може виявитися істинним, а при підстановці інших - хибним.

Якщо областю визначення є всі люди, і ми підставимо замість змінної x значення "Іванов", то судження "Іванов є студентом даного інституту" буде істинним. Якщо ж замість змінної x підставити ім'я іншої людини, яка не є студентом даного інституту, то судження буде хибним.

Якщо P - предикат, то безліч всіх значень змінної x , при яких судження P стає істинним, можна символічно записати наступним чином:

$$\{x \mid P(x)\}$$

Предикати можуть з'єднуватися за допомогою логічних операторів: (AND), (OR) і $\neg$ (NOT) з утворенням складових предикатів.

Реляційне числення кортежів

У реляційному численні кортежів завдання полягає в знаходженні таких кортежів, для яких предикат є істинним.

Це обчислення засноване на змінних кортежу.

Змінними кортежу є такі змінні, областю визначення яких служить вказане відношення, тобто змінні, для яких допустимими значеннями можуть бути тільки кортежі даного відношення.

Для вказівки відношення R в якості області визначення змінної кортежу S використовується наступна форма запису: $R(S)$.

Крім того, запит "знайти безліч всіх кортежів S ", для яких $R(S)$ є істинним" можна записати в такий спосіб:

$\{S \mid R(S)\}$.

Тут предикат R називається **формулою** (правильно побудованою формулою).

Запит "вибрати атрибути Код предмета, Назва предмета, Прізвище викладача для предметів, кількість лекційних годин яких > 22 " можна записати в такий спосіб: $\{S \mid R(S) \wedge S.\text{кількість} > 22\}$.

Для вказівки кількості екземплярів, до яких повинен бути застосований предикат, в формулах можуть використовуватися два типи кванторів.

Квантор існування ($\exists$) використовується у формулі, яка повинна бути істинною хоча б для одного екземпляра.

Наприклад, студент $(S) \wedge \{Зв\} (Рік народження (В) \wedge (В.ім'я = S.ім'я) \wedge В.група =$

'ТИ-31'). Цей вислів означає, що по відношенню Рік народження існує кортеж, який має таке ж значення атрибута ім'я, що і значення атрибута ім'я в поточному кортежі S з відношення Студент, а атрибут група з кортежу V має значення

'ТИ-31'.

Квантор спільності ($\forall$) використовується в виразах, які відносяться до всіх екземплярів.

Приклад. $(В) (В.група * 'ТИ-31')$.

Цей вислів означає, що ні в одному кортежі відношення Рік народження значення атрибута група не дорівнює 'ТМ-31'.

Вільні змінні - змінні кортежу, що не кваліфікуються кванторами $\forall$ або $\exists$; в іншому випадку вони називаються пов'язаними змінними.

Аналогічно тому, як не всі можливі послідовності букв алфавіту утворюють правильно побудовані слова, так і в реляційному численні не кожна послідовність формул є допустимою.

Допустимими формулами можуть бути тільки недвозначні і не безглузді послідовності.

Формула (або правильно побудована формула) в обчисленні предикатів визначається наступними правилами:

- Якщо P є n -арною формулою (предикатом з n аргументами), а $t_1, t_2, \dots, t_n$ - це константи або змінні, то вираз $P(t_1, t_2, \dots, t_n)$ є правильно побудованою формулою.
- Якщо P є n -арною формулою (предикатом з n аргументами), а Θ являє собою один з операторів порівняння ($<, <=, >, >=$, в або $=$), то вираз $t_1 \Theta t_2$ є правильно побудованою формулою.

- Якщо вирази F_1, F_2 є формулами, то їх кон'юнкція позначається як $F_1 \wedge F_2$, диз'юнкція – як $F_1 \vee F_2$, а заперечення – як $\overline{F}$.
- Якщо вираз F_1 є формулою з вільною змінною X , то вирази $\exists F(X)$ і $\forall F(X)$ також є формулами.

Вираз реляційного числення може генерувати нескінченну послідовність кортежів. Уникнути цього можливо завдяки використанню змінних області визначення, які задаються за допомогою оператора RANGE.

Замість використання цього оператора можна явно вказувати область визначення всередині формули, в такому випадку задається безліч кортежів наступним чином: $\{S | \sim (S \in \text{Staff})\}$. Цей вираз означає набір кортежів, які не входять до відношення Staff. Вирази подібного типу називаються небезпечними. Щоб уникнути їх виникнення, слід додати обмеження, що вимагає, щоб всі результуючі значення входили в область визначення формули. Наприклад, областю визначення наведеної вище формули є безліч всіх значень, які представлені в відношенні Staff.

Реляційне числення доменів

У реляційному численні доменів використовуються змінні, значення яких беруться з доменів, а не з кортежів відносин.

Якщо $P(d_1, d_2, \dots, d_n)$ позначає предикат зі змінними $d_1, d_2, \dots, d_n$, то безліч всіх змінних домену $d_1, d_2, \dots, d_n$, для яких предикат або формула $P(d_1, d_2, \dots, d_n)$ істинні, позначається наступним виразом: $\{d_1, d_2, \dots, d_n | P(d_1, d_2, \dots, d_n)\}$

У реляційному численні доменів найчастіше потрібна перевірка умови приналежності, щоб визначити, чи належать значення зазначеного відношенню.

Вираз $R(x, y)$ вважається дійсним тоді і тільки тоді, коли у відношенні R є кортеж зі значеннями x і y в двох його атрибутах.

Приклад. Знайдіть імена всіх викладачів, зарплата яких перевищує 2500 гривень.

$\{\text{Ім'я, Фамілія} \mid \exists \text{ посада, } \exists \text{ зарплата } ((\text{фамілія, посада, зарплата}) \wedge \text{посада} = \text{'викладач'} \wedge \text{зарплата} > 2500)\}$

Для кожного виразу реляційної алгебри існує еквівалентний вираз реляційного числення, а для кожного виразу реляційного числення доменів або кортежів існує еквівалентний вираз реляційної алгебри.

2.5 Інші мови

Незважаючи на те, що реляційне числення є досить складним з точки зору освоєння і використання, проте, його непроцедурна природа вважається досить перспективною і це стимулює пошук інших, більш простих у вживанні непроцедурних методів.

Подібні дослідження викликали появу двох інших категорій реляційних мов:

- на основі перетворень,
- графічних мов.

Мови на основі перетворень є класом непроцедурних мов, які використовують відношення для перетворення вихідних даних до необхідного виду.

Прикладами є SQUARE, SEQUEL і його версії, а також мова SQL.

Графічні мови надають користувачеві малюнок або інші графічне відображення структури відношення; користувач створює якийсь зразок бажаного результату, і система повертає викликані дані в зазначеному форматі.

Прикладом подібного мови є мова QBE.

Ще однією категорією мов є мови **четвертого покоління**, які дозволяють створювати повністю готовий і який відповідає вимогам замовника прикладний додаток за допомогою обмеженого набору команд і в той же час надають дружнє по відношенню до користувача середовище розробки.

Питання:

- 2.1. Як структуровані дані в реляційній моделі?
- 2.2. Розкрийте поняття представлення.
- 2.3. Опишіть основні положення реляційної цілісності. Що таке цілісність сутностей і посиальна цілісність?
- 2.4. Опишіть різницю між процедурним і непроцедурного мовами? До яких з них відносяться реляційна алгебра і реляційне числення?
- 2.5. Поясніть сенс наступних термінів: реляційно-повний; замикання реляційних операцій.
- 2.5. Які операції називаються унарними і бінарними?
- 2.6. Дайте визначення п'яти основних операцій реляційної алгебри. Визначте за допомогою цих п'яти операцій операції з'єднання, перетину і ділення.
- 2.7. Поясніть, в чому полягають відмінності між п'ятьма операціями з'єднання (тета-з'єднання, з'єднання по еквівалентності, природне з'єднання, зовнішнє з'єднання і напівз'єднання). Наведіть приклади, що ілюструють вашу відповідь.
- 2.8. Покажіть, в чому полягають подібність і відмінність реляційного обчислення кортежів і реляційного числення доменів. детально опишіть відмінність між змінними кортежу і домену.
- 2.9. Визначте структуру (правильно побудованої) формули в реляційному численні кортежів і реляційному численні доменів.
- 2.10. Дайте визначення: мови на основі перетворень і графічних мов.

Вправи:

- 2.1. Задано відношення R

Код	Номер семінара	Дата семінара	ПІБ відповідального
1	183	23.01.2006	Іванов К.А.
2	457	11.04.2006	Петров Л.И
3	754	10.08.2006	Сидоров М.А.

Використовуйте операцію вибірки для відношення S, за умови, що семінар проходить не раніше 18.05.2006.

$\sigma_{\text{Дата семінара} \geq 18.05.2006}$

- 2.2. Задано відношення

Код	Дата	Назва запчастини	Вартість
1	25.11.2005	Передній бампер	145
2	17.02.2005	Заднє крило	175
3	18.11.2005	Передній бампер	145
4	03.12.2005	Передня права фара	87

Використовуйте операцію проєкції для відношення R по атрибутам Назва запчастини.

$$\Pi_{\text{Назва запчастини}}(R)$$

2.3. Задано два відношення.

Відношення R

Назва товару	Вартість	Номер склада
Tide	5.7 грн	5
Ariel	6.4 грн	7
Асе	5.3 грн	4

Відношення S

Назва банку	Плата за перерахування грошей (%)	ПІБ касира
Аваль	0.4	Фролов Н.І.
УкрСоцБанк	0.5	Іванов М.О.

Побудуйте відношення R, що дорівнює декартовому добутку відношень R і S.

$$P = R \times S$$

2.4. Задані два відношення.

Відношення R

Номер рейса	Час вильоту	Маршрут
1245	15.20	Київ-Одеса
7894	20.45	Київ-Лондон
4589	9.00	Київ-Рим

Відношення S

Номер рейса	Час вильоту	Маршрут
7814	21.15	Київ-Мілан
7894	20.45	Київ-Лондон

Побудуйте відношення R, що об'єднує відношення R і S.

$$P = R \cup S$$

2.5. Задані два відношення.

Відношення R

Код	Номер виставки	Дата виставки	ПІБ відповідального
1	183	23.01.2006	Степаненко А.А.
2	457	10.04.2006	Рудников В.И.
3	754	10.08.2006	Гордієнко А.М.

Відношення S

Код	Номер виставки	Дата виставки	ПІБ відповідального
1	183	23.01.2006	Степаненко А.А.
4	597	21.05.2006	Василенко И.А.

Побудуйте відношення R, що є різницею відношень R і S.

$$P = R - S$$

2.6. Задані два відношення.

Відношення R

Номер виставки	Дата виставки
101	22.01.2006
165	25.02.2006
358	10.05.2006

Відношення S

Код картини	Назва картини	Номер виставки	ПІБ відповідального
1001	Квіти	101	Степаненко А.А.
1002	Річка	101	Степаненко А.А.
1003	Ніч	365	Рудников В.И.
1004	Море	358	Гордієнко А.М.

Побудуйте відношення R, що є природним з'єднанням відношень R і S.

$$P = R \bowtie S$$

2.7. Задані два відношення.

Відношення R

Номер групи	Кількість студентів	ПІБ куратора
ТМ-31	20	Іванов
ТМ-32	22	Петров
ТІ-31	13	Сидоров

Відношення S

Номер групи	Кількість студентів	ПІБ куратора
ТМ-31	20	Іванов
ТЗ-32	25	Круглов
ТІ-31	13	Сидоров

Побудуйте відношення P , що дорівнює перетину відношень R і S .

2.8. Задані два відношення.

Відношення R

№ склада	Адреса	Зав.склад у	Ідент.код	Кількість	Дата	ПІБ замовника
5	Воровського 1	Зиков	140770	78	15.09	Петров
7	Воровського 10	Сидоров	140765	56	14.11	Іванов
7	Воровського 10	Сидоров	140765	78	15.09	Петров

Відношення S

Ідент.ко д	Кількість	Дата	ПІБ замовника
140770	78	15.09	Петров
140765	56	14.11	Іванов

Побудуйте відношення P , що дорівнює частці від ділення відношень R і S .

$$P = R \div S$$

ТЕМА 3

Моделювання даних

Модель «сутність-зв'язок»

У цій частині описується і ілюструється використання моделі «сутність - зв'язок» (entity-relationship model), введеної Пітером Ченом (Peter Chen) в 1976 р. Чен заклав основу моделі, яка відтоді розширювалася і модифікувалася самим Ченом і багатьма іншими. Крім того, модель «сутність-зв'язок» увійшла до складу множини CASE-інструментів, які також внесли свій внесок в її еволюцію. На сьогоднішній день не існує єдиного загальноприйнятого стандарту для моделі «сутність-зв'язок», зате є набір загальних конструкцій, які лежать в основі більшості варіантів цієї моделі. Символи, що застосовуються для графічного представлення моделі «сутність-зв'язок», досить різні. Ми обговоримо не тільки традиційні символи, а й символи мови UML (Unified Model Language, уніфікована мова моделювання) - засобу проектування, який завойовує все більшу популярність серед програмістів ООП і включає в себе модель «сутність-зв'язок».

Елементи моделі «сутність-зв'язок»

Ключовими елементами моделі «сутність-зв'язок» є сутності, атрибути, ідентифікатори та зв'язки. Розглянемо кожен з них по черзі.

Сутність

Сутність (entity) - *це деякий об'єкт, ідентифікований в робочому середовищі користувача (деякий об'єкт реального світу).*

Позначення засобами в UML-діаграмах:

Сутність позначається:



Прикладами сутностей можуть бути **СТУДЕНТ** Бойко І.Р., **ВИКЛАДАЧ** Ткачук П.Р. або **ПРЕДМЕТ** математика.

Сутності одного і того ж типу групуються в *класи сутностей*.

Так, клас сутностей **СТУДЕНТ** є сукупністю всіх сутностей **СТУДЕНТ**.

Важливо усвідомити різницю між класом сутностей і екземпляром сутності.

Клас сутностей (entity classes) - *це сукупність сутностей, яка описується структурою або форматом сутностей, що становлять цей клас.*

Екземпляр сутності (an instance) являє конкретну сутність, таку як *СТУДЕНТ Бойко І.Р.*; він відрізняється значеннями атрибутів даної сутності. Зазвичай клас сутностей містить множину екземплярів сутності.

Наприклад, клас **СТУДЕНТ** містить множину екземплярів - по одному на кожного студента, для якого існує запис у базі даних. Приклад класу сутностей і двох екземплярів сутностей зображений на рис. 3.1.

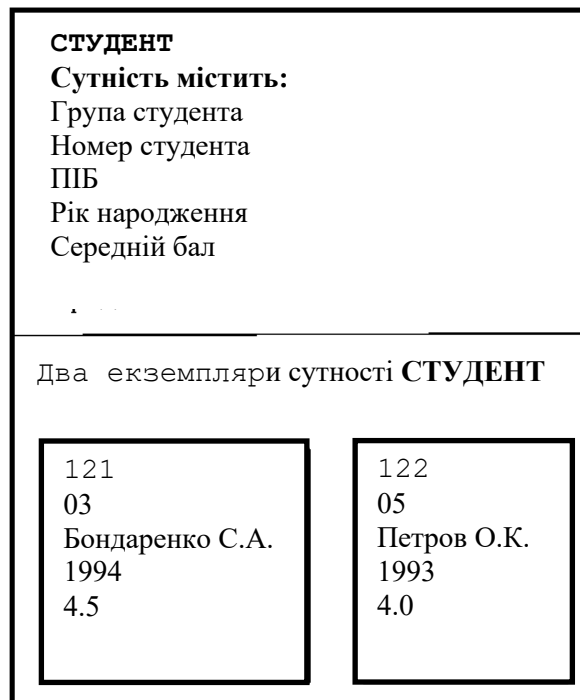


Рис. 3.1. Приклад сутності студент

Атрибути

Атрибути (властивості) - описують характеристики сутності.

Наприклад, атрибутами можуть бути ПІБ студента, Група та Номер студента.

Вихідне визначення моделі «сутність-зв'язок» включає в себе **композитні атрибути** (*composite attributes*) і **багатозначні атрибути** (*multival attributes*).

Наприклад, композитним атрибутом може бути Адреса, що складається з групи атрибутів {Вулиця, Місто, Індекс}.

Багатозначним атрибутом може бути атрибут Ім'я студента сутності **ВИКЛАДАЧ**, який може містити імена декількох студентів, у яких він викладає.

Атрибут може бути одночасно і композитним, і багатозначним, наприклад, композитний атрибут Телефон, що складається з групи атрибутів {Код Міста, Місцевий Номер}, може бути багатозначним, що дозволить мати в базі даних декілька телефонних номерів однієї і тієї ж особи.

У більшості реалізацій моделі «сутність-зв'язок» однозначні композитні

атрибути ігноруються, і потрібно, щоб багатозначні атрибути (чи складові вони, чи ні) перетворювалися на сутності, як буде показано нижче.

Ідентифікатори

Ідентифікатори (identifiers) - *атрибути, за допомогою яких екземпляри сутностей іменуються, або ідентифікуються.*

Екземпляри сутностей класу **СТУДЕНТ** можуть ідентифікуватися за атрибутами Група студента, Номер студента чи ПІБ. Такі атрибути, як Рік народження або Середній бал, навряд чи можуть служити ідентифікаторами екземплярів сутностей класу **СТУДЕНТ**, оскільки зазвичай ці атрибути не використовуються для однозначного визначення конкретного студента. Аналогічно цьому, сутності класу **ВИКЛАДАЧ** можуть ідентифікуватися за атрибутами Табельний номер викладача або ПІБ викладача, а сутності класу **ПРЕДМЕТ** - за атрибутами Код предмета або Назва предмета.

Ідентифікатор екземпляра сутності складається з одного або більше атрибутів сутності. Ідентифікатор може бути *унікальним (unique)* або *неунікальним (nonunique)*. Якщо ідентифікатор є **унікальним**, його значення буде вказувати на один і тільки один екземпляр сутності. Якщо ідентифікатор є **неунікальним**, його значення буде вказувати на деяку множину екземплярів. ПІБ студента є, швидше за все, унікальним ідентифікатором, а Номер студента - неунікальним (наприклад, в іншій групі номер студента може повторюватися).

Ідентифікатори, які з кількох атрибутів, називаються *композитними ідентифікаторами (composite identifiers)*. Прикладами можуть бути сукупності вигляду {Група студента, Номер студента}, {Код Міста, Місцевий Номер}.

Зв'язки

Взаємовідношення сутностей виражаються зв'язками.

Класи зв'язків (relationship classes) - *це взаємовідношення між класами сутностей, а екземпляри зв'язку (relationship instances) - взаємовідношення між екземплярами сутностей.*

У зв'язків можуть бути атрибути.

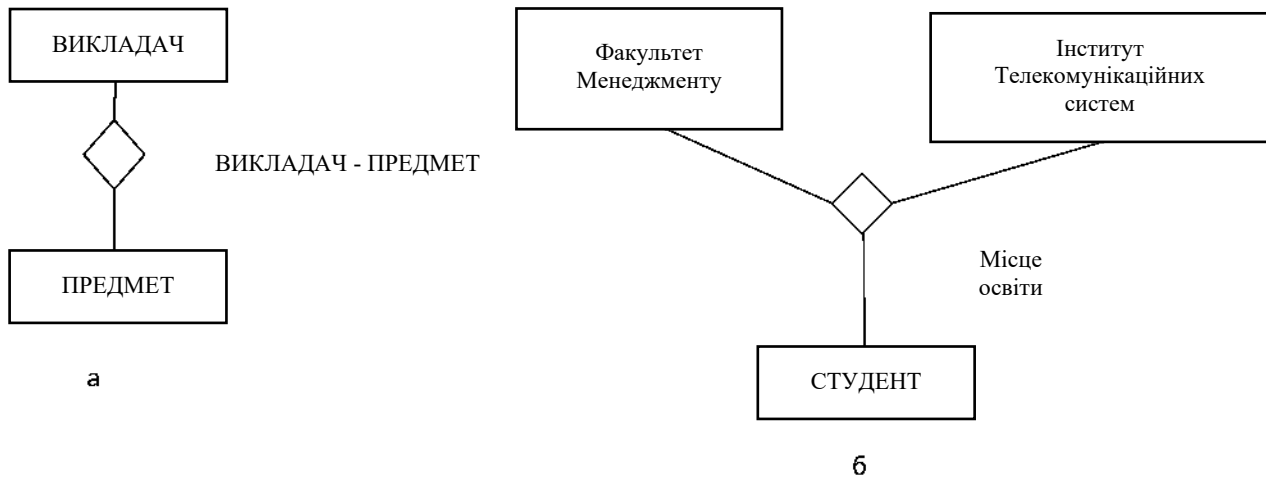
Модель «сутність-зв'язок» включає в себе класи зв'язків і екземпляри зв'язків. Клас зв'язків може стосуватися декількох класів сутностей.

Степінь зв'язку (*relationship degree*) - число класів сутностей, що беруть участь у зв'язку.

Позначення засобами в UML-діаграмах:

Зв'язок позначається 

Зображений на рис. 3.2а зв'язок ВИКЛАДАЧ-ПРЕДМЕТ має степінь 2, оскільки в ньому беруть участь два класи сутностей: ВИКЛАДАЧ і ПРЕДМЕТ. Зв'язок Місце навчання на рис. 3.2б має степінь 3, так як в ньому беруть участь три класи сутностей: Факультет Менеджменту, Інститут Телекомунікаційних Систем і СТУДЕНТ, оскільки студент в НТУУ «КПІ» одночасно може отримувати дві вищі освіти. Зв'язки степеня 2 дуже поширені, їх часто називають ще *бінарними зв'язками* (binary relationships).



Три типи бінарних зв'язків

Позначення засобами в UML -діаграмах:Зв'язок 1: 1 («один до одного»)

позначається 

Зв'язок 1: N («один до №» або «один до багатьох») - 

Зв'язок № M (читається «N до M» або «багато до багатьох») - 

Зв'язок володіння в узагальненому вигляді, коли не вказаний

конкретний тип зв'язку - 

На рис. 3.3 показані три типи бінарних зв'язків.

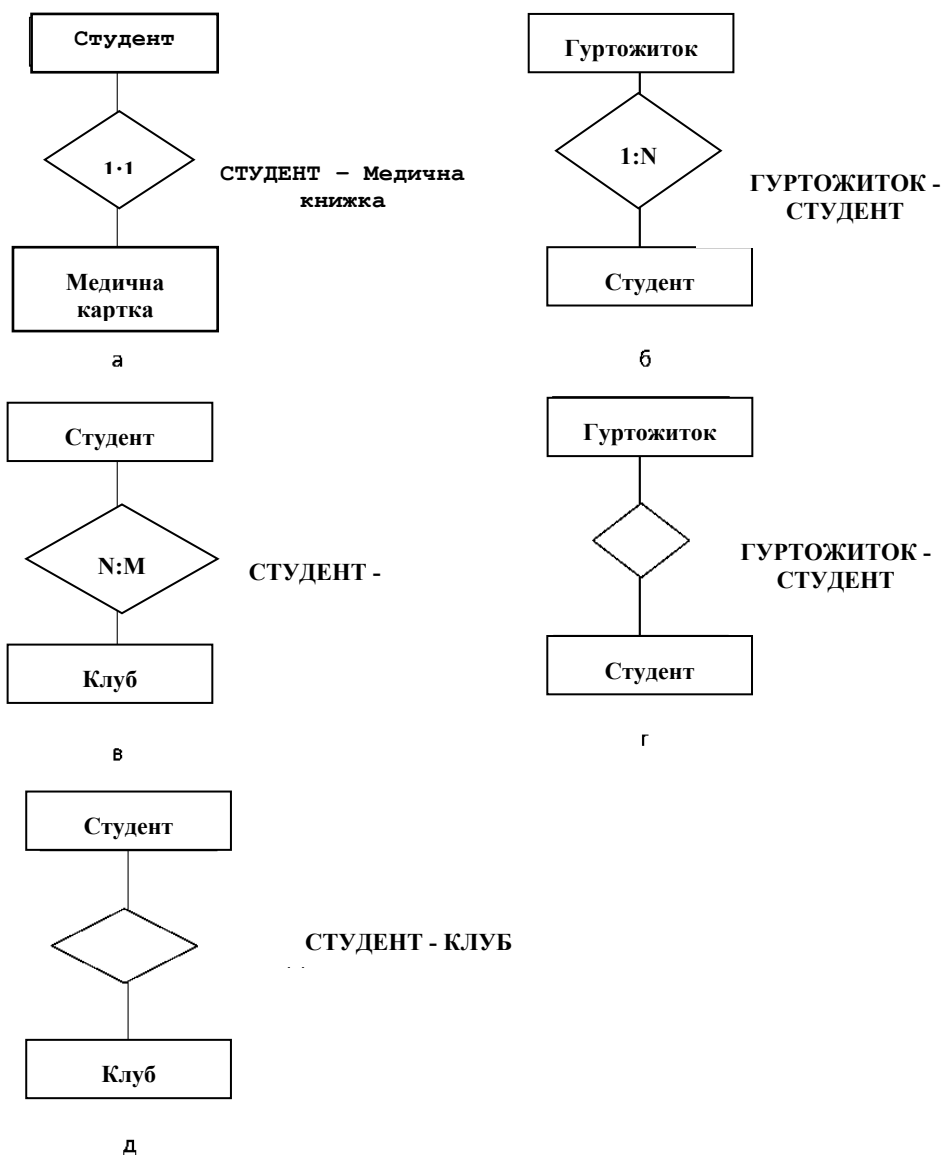


Рис. 3.3. Три типи бінарних зв'язків: а - бінарний зв'язок 1:1, б - бінарний зв'язок 1: N в - бінарний зв'язок N M, г,д - представлення зв'язку володіння в узагальненому вигляді, коли не вказаний конкретний тип зв'язку.

У зв'язку 1:1 одиночний екземпляр сутності одного типу пов'язаний з одиночним екземпляром сутності іншого типу. На рис. 3.3а зв'язок СТУДЕНТ-Медична, картка пов'язує одиночну сутність класу СТУДЕНТ з одиночною сутністю класу Медична картка. Відповідно з цієї діаграмою, ні за одним студентом не закріплено більше однієї медичної картки, і жодна медична картка не закріплена більш ніж за одним студентом.

На рис. 3.3б зображений другий тип зв'язку, 1: N. У цьому зв'язку, який називається ГУРТОЖИТОК-СТУДЕНТ, одиничний екземпляр сутності класу ГУРТОЖИТОК пов'язаний з багатьма екземплярами сутності класу СТУДЕНТ. Відповідно з цим рисунком, в гуртожитку проживає багато студентів, але кожен студент живе тільки в одному гуртожитку.

Позиція, в якій стоять 1 і N, має значення. Одиниця стоїть на тій стороні зв'язки, де розташовується ГУРТОЖИТОК, а N стоїть на тій стороні зв'язку, де розташовується СТУДЕНТ. Якби 1 і N розташовувалися навпаки, і зв'язок записувалася б як N: 1, вийшло б, що в гуртожитку живе один студент, причому кожен студент проживає в кількох гуртожитках. Це, зрозуміло, не так.

На рис. 3.3в показаний третій тип бінарного зв'язку, N:M. Цей зв'язок називається СТУДЕНТ-КЛУБ, і він пов'язує екземплярів сутностей класу СТУДЕНТ з екземплярами сутностей класу КЛУБ. Один студент може бути членом кількох клубів, а в одному клубі може бути багато студентів.

Числа всередині ромба, яке символізує зв'язок, позначають максимальну кількість сутностей на кожній стороні зв'язку. Ці обмеження називаються **максимальними кардинальними числами**, а сукупність з двох таких обмежень для обох сторін зв'язку називається **максимальною кардинальністю** (*maximum cardinality*) зв'язку.

Про зв'язок, зображений на рис. 3.3б, кажуть, що він має максимальну кардинальність 1: N. Кардинальні числа можуть мати й інші значення, а не тільки 1 і N. Наприклад, зв'язок між сутностями Баскетбольна команда і ГРАВЕЦЬ може мати кардинальність 1:5, що говорить нам про те, що у баскетбольній команді може бути не більше п'яти гравців.

Зв'язки трьох типів, представлених на рис. 3.3, називаються іноді зв'язками типу «МАЄ», або **зв'язками володіння** (HAS-A relationships).

Такий термін використовується тому, що одна сутність має (has) зв'язок з іншою сутністю. Наприклад, студент має медичну картку, студент має гуртожиток, клуб має студентів.

На рис. 3.3г показано представлення зв'язку володіння в узагальненому вигляді, суть якого полягає в тому, що має місце багатозначний зв'язок між двома сутностями і конкретно не вказано який тип зв'язку присутній.

Діаграми «сутність-зв'язок»

Схеми, зображені на рис. 3.3, називаються **діаграмами «сутність-зв'язок»**, або **ER-діаграмами** (*entity-relationship diagrams, ER-diagrams*). Такі діаграми стандартизовані, але не дуже жорстко. Відповідно до цього стандарту, *класи сутностей позначаються прямокутниками, зв'язки позначаються ромбами, а максимальне кардинальне число кожного зв'язку вказується всередині ромба. Назва сутності вказується усередині прямокутника, а ім'я зв'язку вказується поруч з ромбом.*

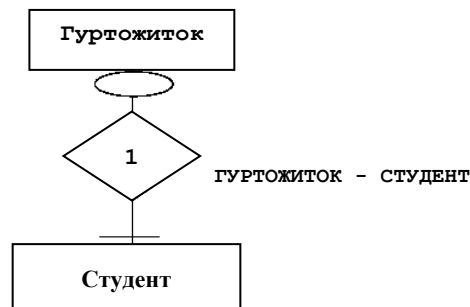


Рис. 3.4. Зв'язок з вказаною мінімальною кардинальністю

Як вже говорилося, максимальна кардинальність показує максимальне число сутностей, які можуть брати участь у зв'язку. Яке ж мінімальне число таких сутностей, наведені діаграми не повідомляють. Наприклад, рис. 3.3б показує, що студент може проживати максимум в одному гуртожитку, однак з нього не ясно, чи зобов'язаний студент проживати в якомусь гуртожитку.

Для вказівки **мінімальної кардинальності** (*minimum cardinality*) існує кілька способів. Один з них, продемонстрований на рис. 3.4, полягає в наступному: щоб показати, що *сутність зобов'язана брати участь у зв'язку*, на лінію зв'язку поміщають *перпендикулярну риску*, а щоб показати, що *сутність може (але не зобов'язана) брати участь у зв'язку*, на лінію зв'язку поміщають *овал*.

Відповідно, рис. 3.4 показує, що сутність ГУРТОЖИТОК повинна бути пов'язана як мінімум з однією сутністю СТУДЕНТ, однак сутність СТУДЕНТ не зобов'язана мати зв'язок з сутністю ГУРТОЖИТОК. Повний набір накладених на зв'язок обмежень полягає в тому, що ГУРТОЖИТОК має мінімальне кардинальне число, рівне одиниці, і максимальне кардинальне число, рівне «багатьом» сутностям СТУДЕНТ. СТУДЕНТ має мінімальне кардинальне число, рівне нулю, і максимальне кардинальне число, рівне одному екземпляру сутності ГУРТОЖИТОК.

Може існувати зв'язок між сутностями одного і того ж класу. Наприклад, для сутностей класу СТУДЕНТ може бути визначена зв'язок СУСІД_ПО_КІМНАТІ. Такий зв'язок показан на рис. 3.5. Зв'язки між сутностями одного і того ж класу називаються іноді **рекурсивними зв'язками** (*recursive relationships*).

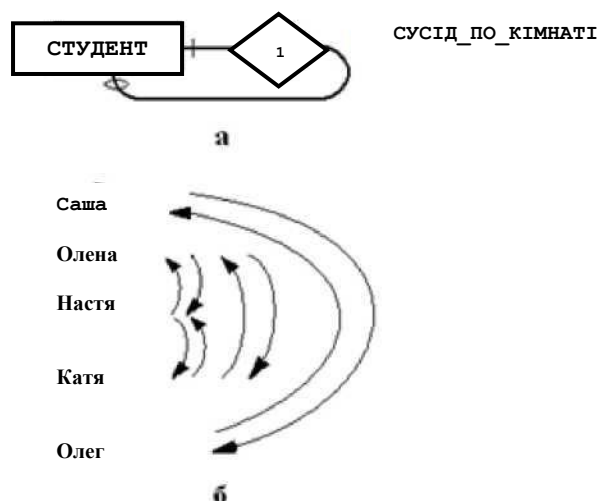
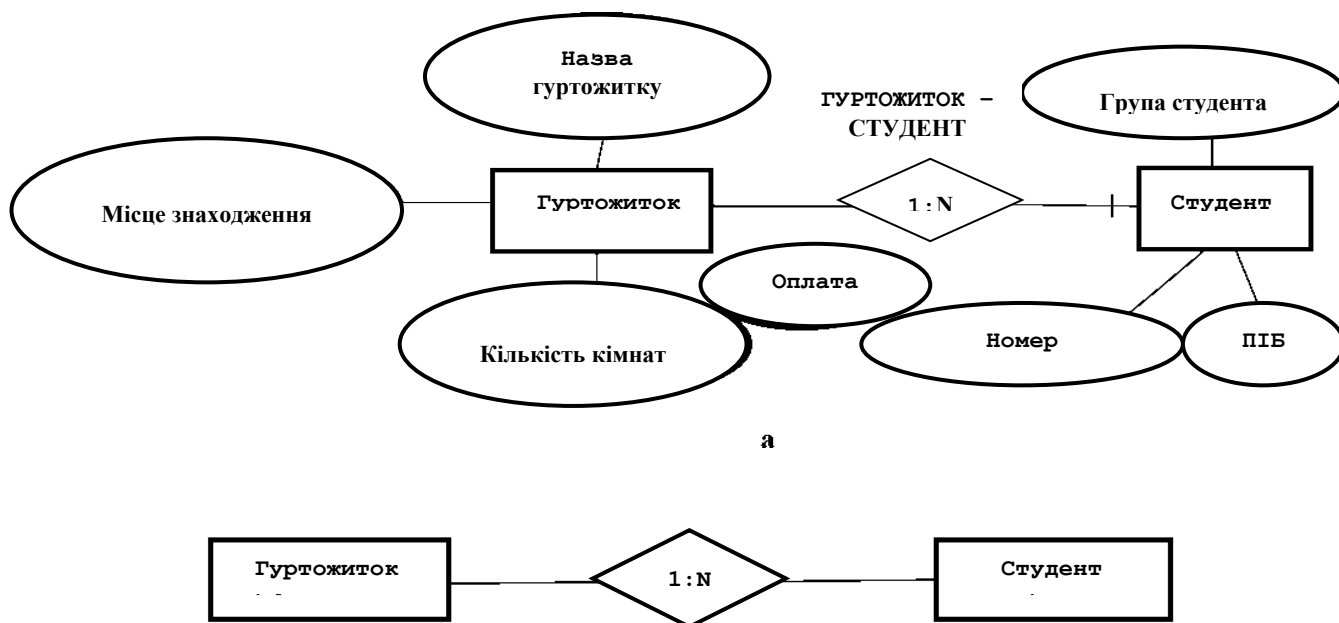


Рис. 3.5. Рекурсивний зв'язок

Зображення атрибутів в діаграмах «сутність-зв'язок»

У деяких версіях ER-діаграм атрибути позначаються еліпсами, з'єднаними з сутністю або зв'язком, яким вони належать. На рис. 3.6а показані сутності ГУРТОЖИТОК і СТУДЕНТ і зв'язок ГУРТОЖИТОК- СТУДЕНТ з належними їм атрибутами. Як видно з рисунку, сутність ГУРТОЖИТОК має атрибути Назва гуртожитку, Розташування і Кількість кімнат, а сутність СТУДЕНТ має атрибути Група студента, Номер студента та ПІБ. Зв'язок ГУРТОЖИТОК-СТУДЕНТ має атрибут Плата, який показує внесену студентом плату за проживання в конкретному гуртожитку.

Якщо сутність має багато атрибутів, такий їх перелік в ER-діаграмі може зробити її занадто громіздкою і важкою для сприйняття. У подібних випадках список атрибутів сутностей дається окремо. Багато CASE-засобів показують такі атрибути в вікнах, які розкриваються.



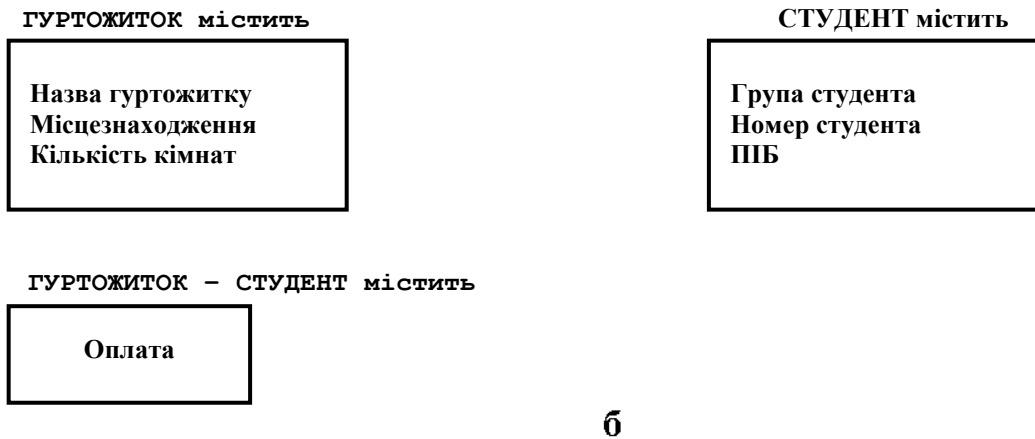


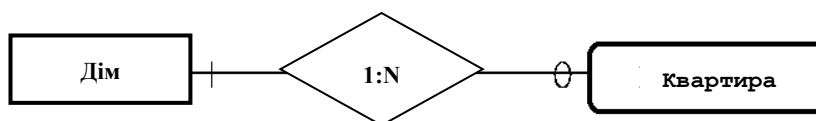
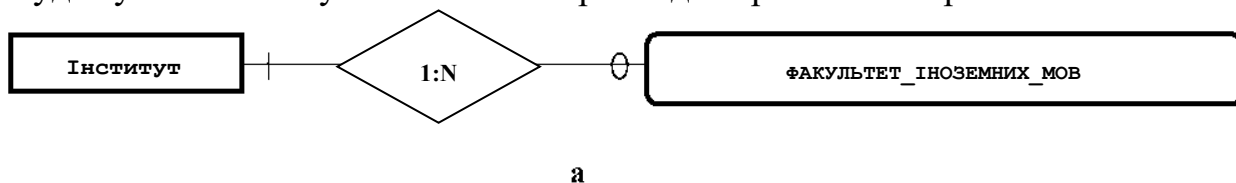
Рис. 3.6. Зображення властивостей на діаграмі «сутність-зв'язок»: а - зображення на діаграмі; б - окреме перерахування

Слабкі сутності

У моделі «сутність-зв'язок» визначено особливий тип сутності, що називається слабкою сутністю (*weak entity*). До **слабких сутностей** відносяться такі сутності, які можуть існувати в базі даних тільки в тому випадку, якщо в ній присутня сутність деякого іншого типу.

Сутність, яка не є слабкою, називається **сильною сутністю** (*strong entity*).

Щоб розібратися в тому, що таке слабкі сутності, розглянемо класи сутностей ІНСТИТУТ та ФАКУЛЬТЕТ_ІНОЗЕМНИХ_МОВ. Припустимо, що екземпляр сутності ІНСТИТУТ може існувати, не будучи пов'язаним ні з однією сутністю класу ФАКУЛЬТЕТ_ІНОЗЕМНИХ_МОВ, але сутність ФАКУЛЬТЕТ_ІНОЗЕМНИХ_МОВ не може існувати поза зв'язком з класом ІНСТИТУТ. Тоді сутність ФАКУЛЬТЕТ_ІНОЗЕМНИХ_МОВ є слабкою. Це означає, що дані про сутність ФАКУЛЬТЕТ_ІНОЗЕМНИХ_МОВ можуть з'явитися в базі даних тільки в тому випадку, якщо ця сутність має зв'язок з якою-небудь сутністю класу ІНСТИТУТ. Приклад зображений на рис. 3.7.



Ідентифікатор:

Назва дому

Ідентифікатор:

{Назва дому

Номер квартири}

б

Рис. 3.7. Слабкі сутності: а - приклад слабкої сутності, б - приклад ідентифіковано-залежної сутності

У моделі «сутність-зв'язок» є особливий тип слабких сутностей, який називається **ідентифікаційний-залежними сутностями** (*ID-dependent entities*) - це такі сутності, ідентифікатори яких містять ідентифікатор іншої сутності.

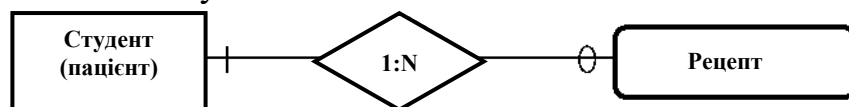
Розглянемо сутності ДІМ і КВАРТИРА. Нехай ідентифікатором сутності ДІМ є атрибут Назва будинку, а ідентифікатором сутності КВАРТИРА є композитний ідентифікатор {Назва будинку, Номер квартири}. Оскільки ідентифікатор сутності КВАРТИРА містить в собі ідентифікатор сутності ДІМ (Назва будинку), то сутність КВАРТИРА є ідентифікаційний-залежною від сутності ДІМ. По-іншому можна представити це так, що, як логічно, так і фізично, квартира не може існувати, якщо не існує відповідної будівлі.

На жаль, у визначенні терміна слабка сутність є прихована неоднозначність, і вона по-різному інтерпретується різними проектувальниками баз даних (а також різними авторами книг). Ця неоднозначність полягає в наступному: *в строгому сенсі, слабка сутність визначається як будь-яка сутність, чия присутність в базі даних залежить від іншої сутності; тоді будь сутність, що бере участь у зв'язку мінімальної кардинальності 1 з іншою сутністю, є слабкою*. Таким чином, розглядаючи базу даних освітнього закладу, можна постановити наступне: якщо у студента повинен бути керівник, то сутність СТУДЕНТ є слабкою, оскільки вона не може бути записана в базу даних без зв'язку з будь сутністю класу КЕРІВНИК. Це тлумачення, однак, здається деяким людям дуже широким. Студент фізично не залежить від керівника (на відміну від випадку з квартирою і будинком), як не залежить від нього і логічно (що б з цього приводу не думали студент і керівник!), Тому сутність СТУДЕНТ повинна вважатися сильною сутністю.

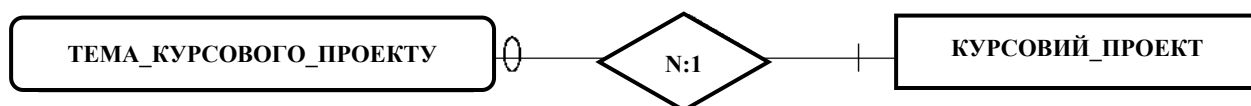
Щоб уникнути подібних ситуацій, деякі інтерпретують визначення слабкої сутності більш обмежено. *Щоб сутність можна було віднести до розряду слабких, вона повинна логічно залежати від іншої сутності*. Згідно цьому визначенню, сутності ФАКУЛЬТЕТ_ІНОЗЕМНИХ_МОВ і КВАРТИРА повинні вважатися слабкими, а сутність СТУДЕНТ - ні. Факультет іноземних мов не може існувати поза інститутом, а квартира не може існувати без будівлі, в якій вона могла б розташовуватися. Студент же може логічно існувати і без керівника, навіть якщо регламент цього не допускає. Щоб проілюструвати цю інтерпретацію, розглянемо декілька прикладів.

Нехай модель даних включає в себе зв'язок між сутностями СТУДЕНТ (пацієнт) і РЕЦЕПТ, зображену на рис. 3.8а. Тут рецепт не може логічно існувати без студента (пацієнта). Отже, крім того, що мінімальне кардинальне число сутності РЕЦЕПТ дорівнює 1, ця сутність також залежить від існування сутності СТУДЕНТ (пацієнт). Звідси випливає, що РЕЦЕПТ - це слабка сутність. Тепер

розглянемо сутність ТЕМА_КУРСОВОГО_ПРОЕКТУ, зображену на рис. 3.8б. Ідентифікатор цієї сутності містить ідентифікатор сутності КУРСОВИЙ_ПРОЕКТ. Тут, крім того, що сутність ТЕМА_КУРСОВОГО_ПРОЕКТУ має мінімальну кардинальність 1 і залежить від існування сутності КУРСОВИЙ_ПРОЕКТ, вона є ще й ідентифікаційно-залежна від останньої, тому що її ключ містить ключ сутності КУРСОВИЙ_ПРОЕКТ. Таким чином, сутність ТЕМА_КУРСОВОГО_ПРОЕКТУ є слабкою. Будь-яка слабка сутність має мінімальне кардинальне число 1 у зв'язку з сутністю, від якої залежить, але довільно взята сутність з мінімальним кардинальним числом, рівним 1, не обов'язково повинна бути слабкою.



а



Ідентифікатор:

{Назва курсового проекту,

Назва задачі}

Ідентифікатор:

Назва курсового проекту

б

Рис. 3.8 Приклад обов'язкових сутностей

Багатозначні атрибути представляються в моделі «сутність-зв'язок» шляхом створення нової слабкої сутності та побудови зв'язку виду «один до багатьох». Розглянемо багатозначний атрибут ТЕМА_КУРСОВОГО_ПРОЕКТУ сутності СТУДЕНТ. Створюється нова сутність з ім'ям ТЕМА_КУРСОВОГО_ПРОЕКТУ з єдиним атрибутом тема курсового проекту. Зв'язок між сутностями СТУДЕНТ і ТЕМА_КУРСОВОГО_ПРОЕКТУ має вигляд «один до багатьох». Створена таким чином сутність повинна бути слабкою, оскільки вона логічно залежить від сутності, що має багатозначний атрибут. На рис. 3.9б зображено представлення багатозначного складеного атрибуту Адреса. Нова слабка сутність АДРЕСА містить всю сукупність атрибутів, а саме атрибути Вулиця, Місто, Поштовий індекс.

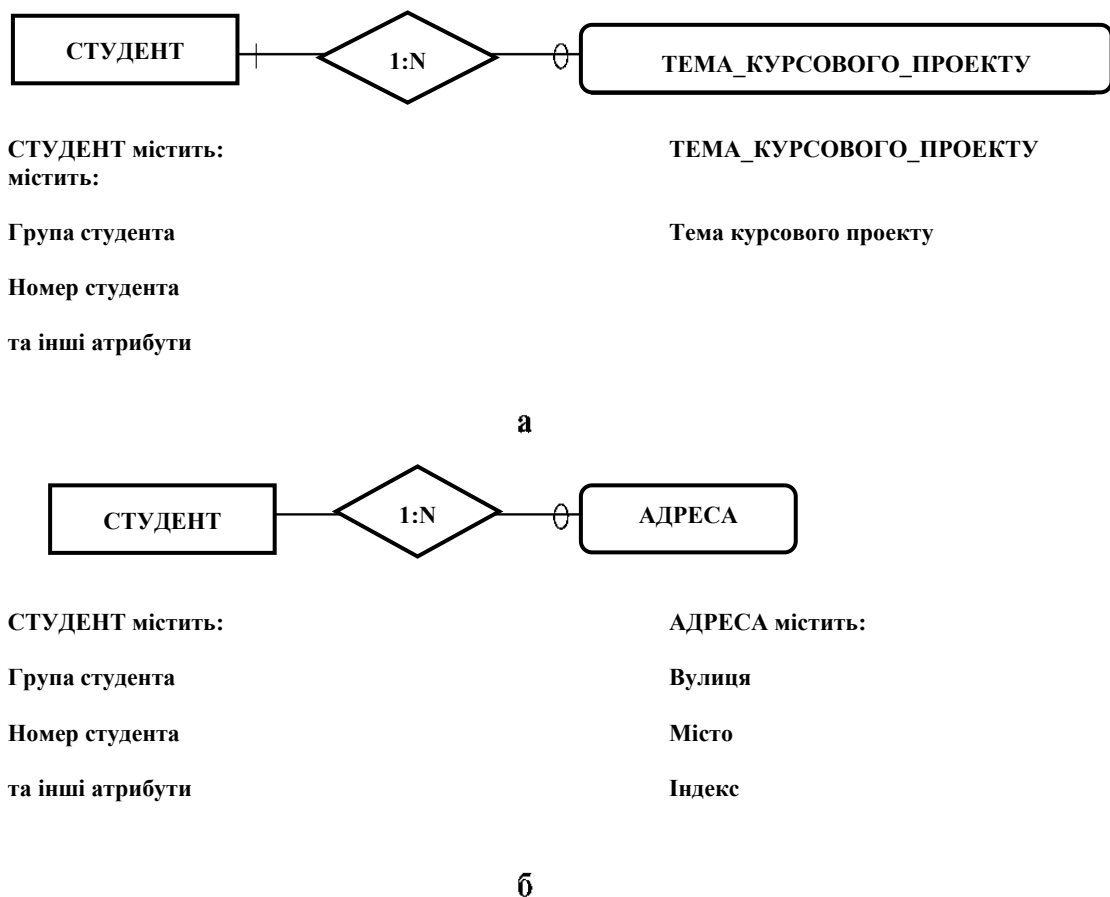


Рис. 3.9. Представлення багатозначних атрибутів за допомогою слабких сутностей

Підтипи сутностей

Деякі сутності мають необов'язкові набори атрибутів; ці сутності часто представляються за допомогою **підтипів** (*subtypes*).

Розглянемо, наприклад, сутність **СТУДЕНТ** з атрибутами Назва факультету, Група студента, Номер студента та ПІБ. Припустимо, що студент може бути главою профбюро, фахівцем в області програмування і що необхідно вказувати деяку додаткову інформацію, яка залежить від типу студента. Нехай ця інформація має такий зміст:

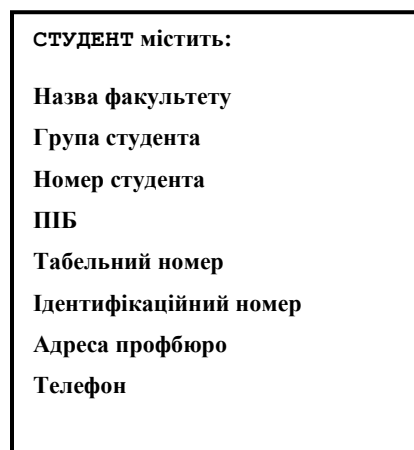
ГОЛОВА_ПРОФБЮРО: Адреса профбюро, Телефон.

СПЕЦІАЛІСТ_ПРОГРАМУВАННЯ: Табельний номер, Ідентифікаційний номер.

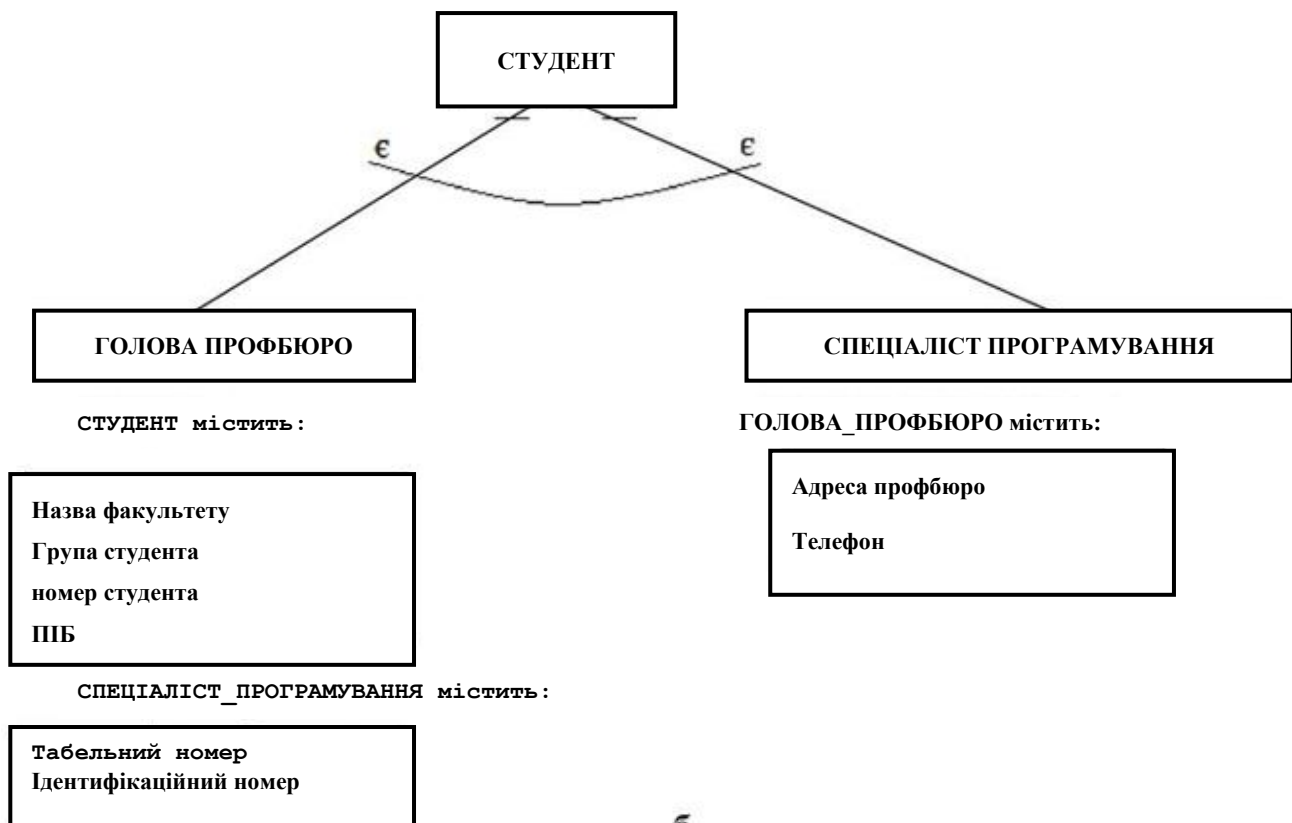
Одна з можливостей - віднести всі ці атрибути до сутності **СТУДЕНТ**, як показано на рис. 3.10а. У цьому випадку, однак, деякі атрибути будуть невикористовуваними. Наприклад, такий атрибут, як Адреса та Телефон профбюро, не має сенсу для студента, як фахівця в області програмування, і таким чином, він не може мати якого-небудь значення.

У моделі, більш близькою до реальної ситуації, замість цього буде визначено два підтипи сутностей, як показано на рис. 3.10 б, в. Тут сутності **ГОЛОВА_ПРОФБЮРО** **СПЕЦІАЛІСТ_ПРОГРАМУВАННЯ** зображені як підтипи, сутності **СТУДЕНТ**. Остання, у свою чергу, є надтип (*supertype*) для сутностей **ГОЛОВА_ПРОФБЮРО** і **СПЕЦІАЛІСТ_ПРОГРАМУВАННЯ**.

Символ Є поруч з лініями зв'язку вказує, що сутності ГОЛОВА_ПРОФБЮРО і СПЕЦІАЛІСТ_ПРОГРАМУВАННЯ є підтипами сутності СТУДЕНТ. Кожен підтип повинен належати надтипу СТУДЕНТ. *Крива лінія з цифрою 1 поруч показує, що сутність СТУДЕНТ повинна належати до одного і тільки одному підтипу.* Це означає, що підтипи є взаємовиключними і що потрібно тільки один з них. Наприклад, якщо ми стверджуємо, що студент, який є фахівцем в області програмування, не може бути главою профбюро, і глава профбюро в свою чергу не може бути фахівцем, тоді сутність СТУДЕНТ належить тільки одному з підтипів (рис. 3.10б). А якщо припускаємо, що студент може бути ГОЛОВА_ПРОФБЮРО і СПЕЦІАЛІСТ_ПРОГРАМУВАННЯ належать сутності СТУДЕНТ (рис.3.10в).



А



б

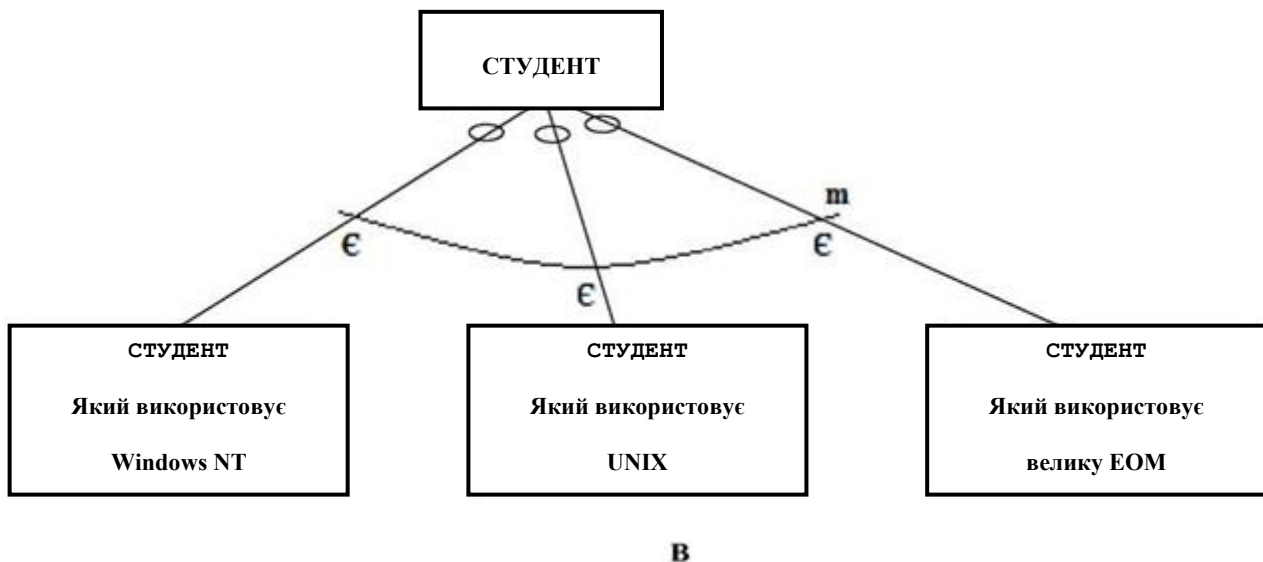


Рис. 3.10. Підтипи сутностей: а - СТУДЕНТ без підтипів; б - СТУДЕНТ з підтипами; в - невзаємовиключні підтипи з необов'язковим надтипом.

Сутності зі зв'язком типу «Є» повинні мати один і той же ідентифікатор, оскільки вони представляють різні аспекти одного і того ж екземпляра об'єкта (або сутності). У даному випадку таким ідентифікатором є ПІБ студента.

Ієрархії генералізації мають спеціальну характеристику, що називається **наслідуванням** (*inheritance*), яка означає, що підтипи класів сутностей наслідують атрибути від надтипу.

Сутність СТУДЕНТ, наприклад, наслідують атрибути ПІБ студента і КУРС від сутності СТУДЕНТ. Причини, з яких підтипи використовуються в моделюванні даних, відрізняються від причин, через які вони використовуються в об'єктно-орієнтованому програмуванні. Фактично, в моделі даних вони застосовуються з *єдиною метою: уникнути ситуацій, при яких деякі атрибути повинні мати нульові значення.*

Наприклад, якщо всі атрибути сутності СПЕЦІАЛІСТ_ПРОГРАМУВАННЯ перенести в одну загальну таблицю сутності СТУДЕНТ, то поля, які характеризують студента саме як фахівця, будуть не заповнені для багатьох студентів. Для економії дискового простору вигідно інформацію рознести, виділивши окремо надтип і підтип, і записувати тільки значущу інформацію, виключаючи нулі.

Приклад ER-діаграми

На рис. 3.11 показаний приклад ER-діаграми, яка містить всі елементи моделі «сутність-зв'язок», про які йшла мова до цих пір. Вона зображує сутності та зв'язку навчального процесу.

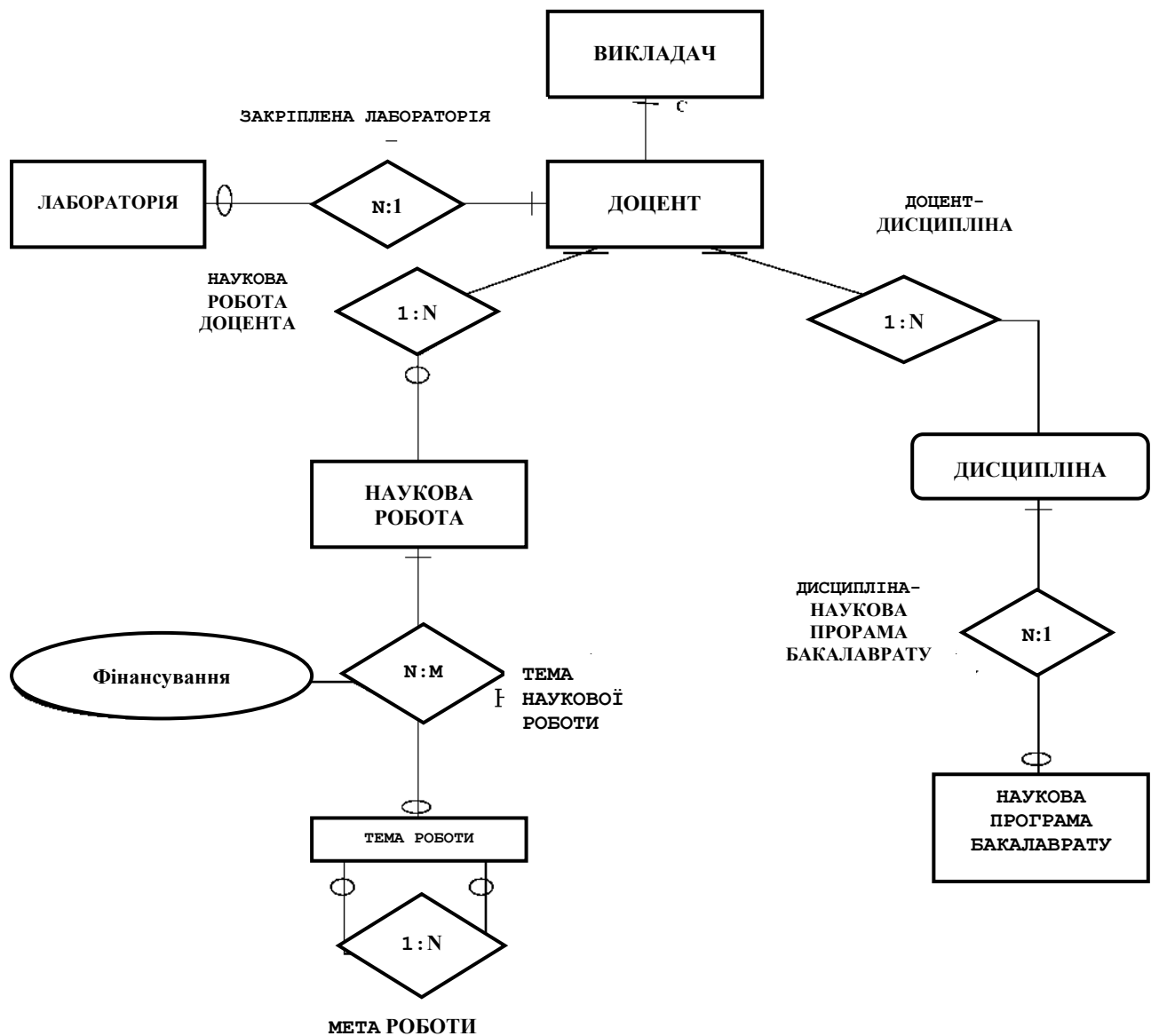


Рис. 3.11. Приклад діаграми «сутність-зв'язок»

На діаграмі є клас сутностей, що представляє викладачів. Оскільки деякі викладачі є доцентами, сутність ВИКЛАДАЧ пов'язана з сутністю ДОЦЕНТ як підтип. Сутність ДОЦЕНТ має зв'язок 1: N з сутністю ЛАБОРАТОРІЯ; кожна лабораторія повинна бути закріплена за якимось доцентом, але не у всіх доцентів є лабораторії.

Доценти ведуть наукову роботу (сутність НАУКОВА РОБОТА), маючи певну тему (сутність ТЕМА РОБОТИ). Доцент може не вести ніяких наукових робіт (інакше кажучи, виконувати нуль робіт) або ввести багато робіт, але кожна окремо взята наукова робота може виконуватися тільки одним конкретним доцентом. На одну велику тему може виконуватися кілька наукових робіт, одночасно будь-яка велика наукова робота може ділитися на декілька тем. Зв'язок ТЕМА_НАУКОВОЇ_РОБОТИ має атрибут Фінансування, який показує суму, необхідну для здійснення даної наукової роботи. Фінансування може вестися або по кожній роботі, або по темі.

Іноді одна тема наукової роботи може мати кілька цілей, що показується за допомогою рекурсивного зв'язку МЕТА РОБОТИ.

Сутність ДИСЦИПЛІНА показує, що даний доцент володіє знаннями і викладає певну дисципліну. Доцент може мати конкретну навчальну програму⁶³

бакалаврату (сутність НАВЧАЛЬНА ПРОГРАМА бакалаврат). Існування сутності ДИСЦИПЛІНА залежить від сутності ДОЦЕНТ через зв'язок ДОЦЕНТ-ДИСЦИПЛІНА.

Діаграми «сутність-зв'язок» у стилі UML

Уніфікована мова моделювання (UML, Unified Model Language) - це набір структур і методик для моделювання та проектування об'єктно-орієнтованих програм (ООП) та прикладного програмного забезпечення.

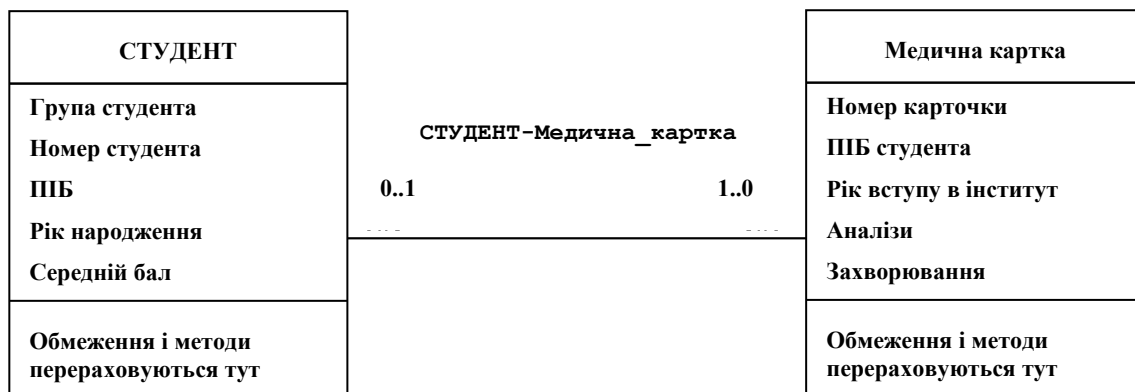
UML - це одночасно і методологія розробки систем ООП, і набір інструментів для розробки таких систем. UML здобула популярність стараннями групи OMG (Object Management Group) - організації, яка займається розробкою ООП-моделей, технології і стандартів з 1980-х років. Ця мова стала також знаходити широке застосування в середовищі професіоналів ООП. На UML базуються інструменти для об'єктно-орієнтованого проектування, розроблені компанією Rational Systems.

Будучи методологією розробки прикладного програмного забезпечення, UML є предметом курсу системної розробки і тому становить для нас лише обмежений інтерес. Вам можуть, однак, зустрітися діаграми «сутність-зв'язок», виконані в стилі UML, тому слід мати представлення про цей стиль.

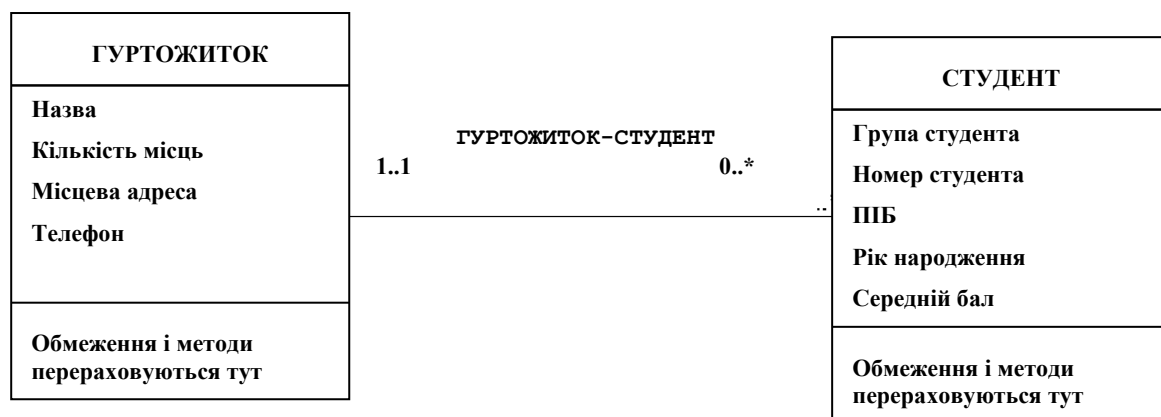
Потрібно просто усвідомити, що коли справа стосується проектування баз даних, робота з цими діаграмами відбувається точно так само, як і з традиційними ER-діаграмами.

Сутності та зв'язки в UML

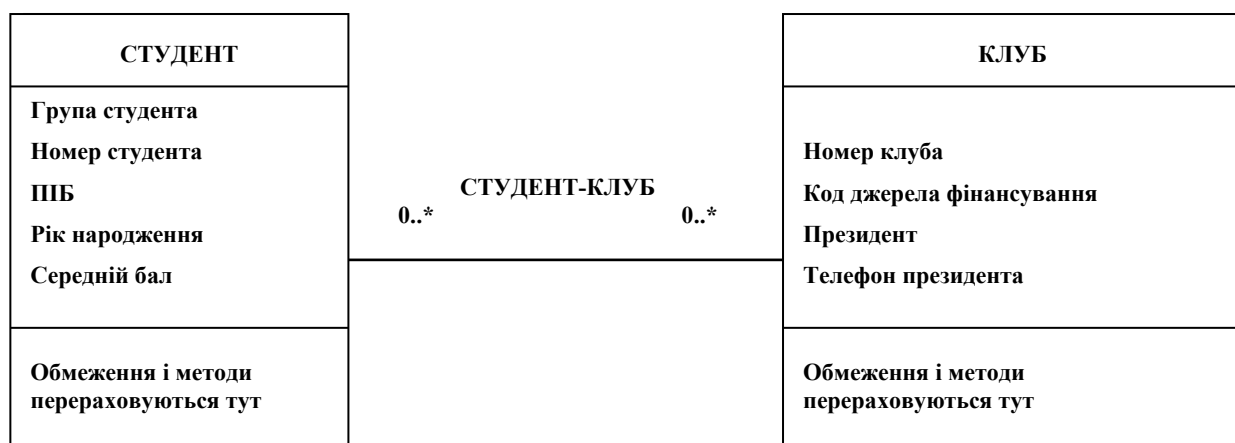
На рис. 3.12 наведено UML -представлення структур, зображених на рис.3.3. Кожна сутність представлена класом сутностей, який зображений у вигляді прямокутника з трьома сегментами. У верхньому сегменті вказано ім'я сутності та інші дані, про які ми будемо говорити далі. У другому сегменті перераховані імена атрибутів сутності, а в третьому описані обмеження і методи (програмні процедури), що відносяться до даної сутності.



а



б



в

Рис. 3.12. Представлення різних типів зв'язку в ЦМБ: а - зв'язок 1:1, б - зв'язок 1[^], в - зв'язок №М.

Зв'язки показані лініями, що з'єднують дві сутності. Кардинальність представлена у форматі x..y, де x - це необхідний мінімум, а y - допустимий максимум. Так, 0 .. 1 означає, що наявність даної суті необов'язково, а максимально допустима кількість - одна. Зірочка представляє *необмежену кількість*. Наприклад, запис 1 .. * означає, що потрібна одна сутність, а допускається необмежена кількість.

Представлення слабких сутностей

На рис. 3.13 зображено UML -представлення слабких сутностей. На лінії зв'язку поруч з батьком слабкою сутністю (тобто поруч із сутністю, від якої залежить слабка сутність) поміщається *зафарбований ромб*. На рис. 3.13 сутність РЕЦЕПТ є слабкою сутністю, а сутність СТУДЕНТ (пацієнт) - її батьком. Всі слабкі сутності мають батька, тому їх кардинальність у зв'язку з батьком завжди 1..1. Виходячи з цього, кардинальність на батьківській стороні позначається просто як 1.

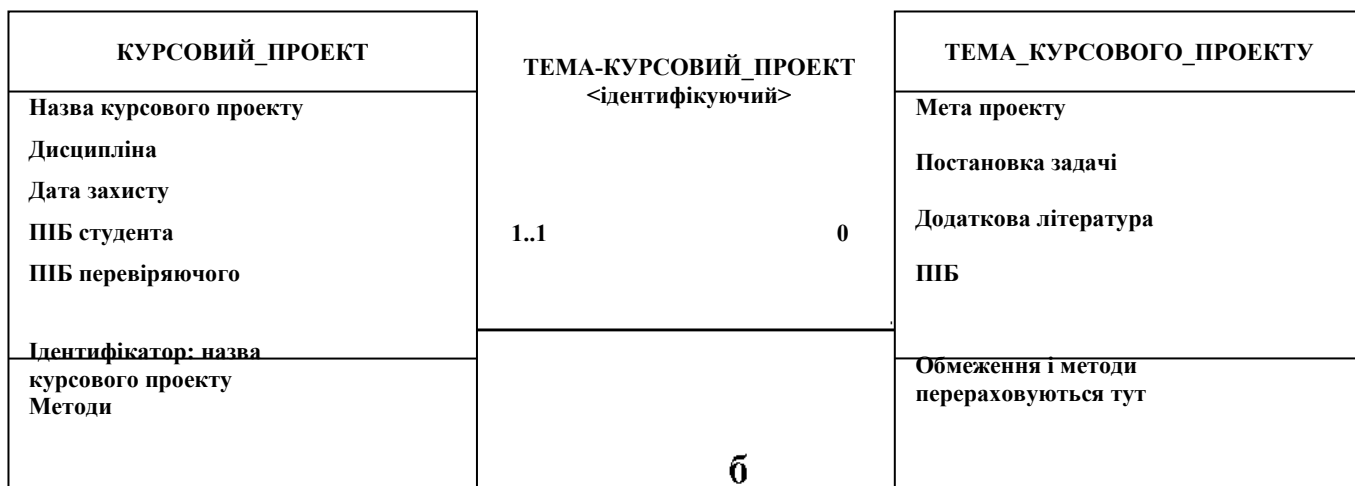
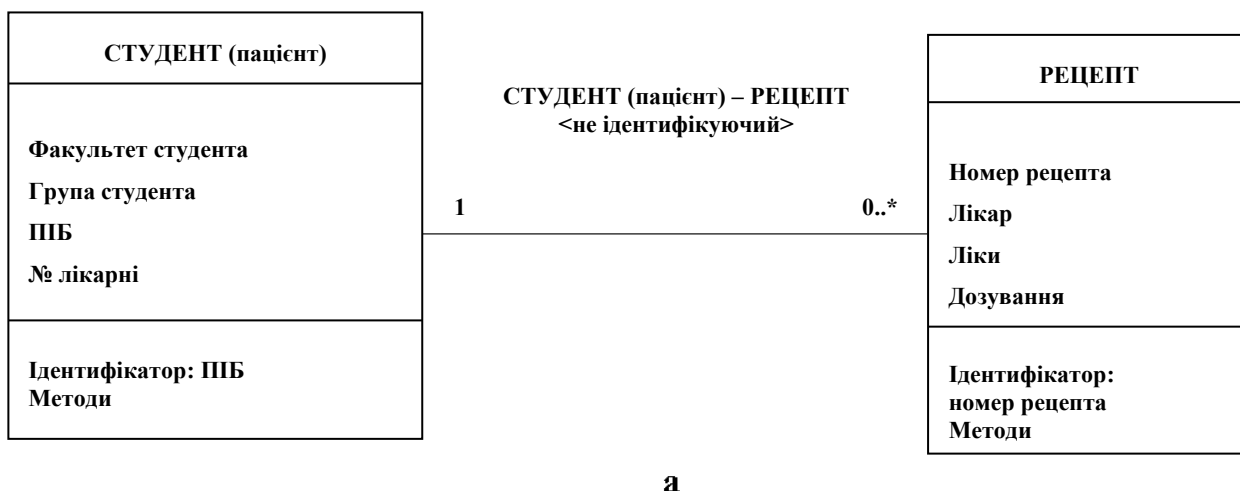


Рис. 3.13. Представлення слабких сутностей в UML: а - слабка сутність, яка не є ідентифікаційно-залежною; б - ідентифікаційно-залежна слабка сутність

На рис. 3.13а показана слабка сутність, яка не є ідентифікаційно- залежною. Це позначається виразом <non-identifying> (не ідентифікується) на зв'язку СТУДЕНТ (пацієнт)-РЕЦЕПТ. На рис. 3.13б зображена ідентифікаційно-залежна слабка сутність. На це вказує ярлик <identifying> (ідентифікуюча).

Представлення підтипів

Спосіб подання підтипів в ЦМБ зображений на рис. 3.14. На цьому рисунку допустимими підтипами сутності Студент є ГОЛОВА ПРОФБЮРО і СПЕЦІАЛІСТ ПРОГРАМУВАННЯ. Згідно з рисунком, кожен студент може мати один або два зазначених підтипу. Для даної ситуації це не має сенсу: студент повинен бути одного і тільки одного типу. Поточна версія UML не надає способів для документування взаємовиключення. Можна, однак, додати в діаграму відповідне позначення.



Рис. 3.14. Представлення підтипів UML

На рис. 3.15 представлена UML-версія діаграми «сутність-зв'язок», показаної раніше на рис. 3.11. Оскільки зв'язок між сутностями НАУКОВА РОБОТА та ТЕМА РОБОТИ має атрибут Фінансування, для цього атрибута виділена спеціальна сутність ДОГОВІР ДЛЯ ВИКОНАННЯ НАУКОВОЇ РОБОТИ. Така стандартна практика при використанні коштів ЦМБ. *Зверніть також увагу на подання рекурсивного зв'язку МЕТА РОБОТИ.*

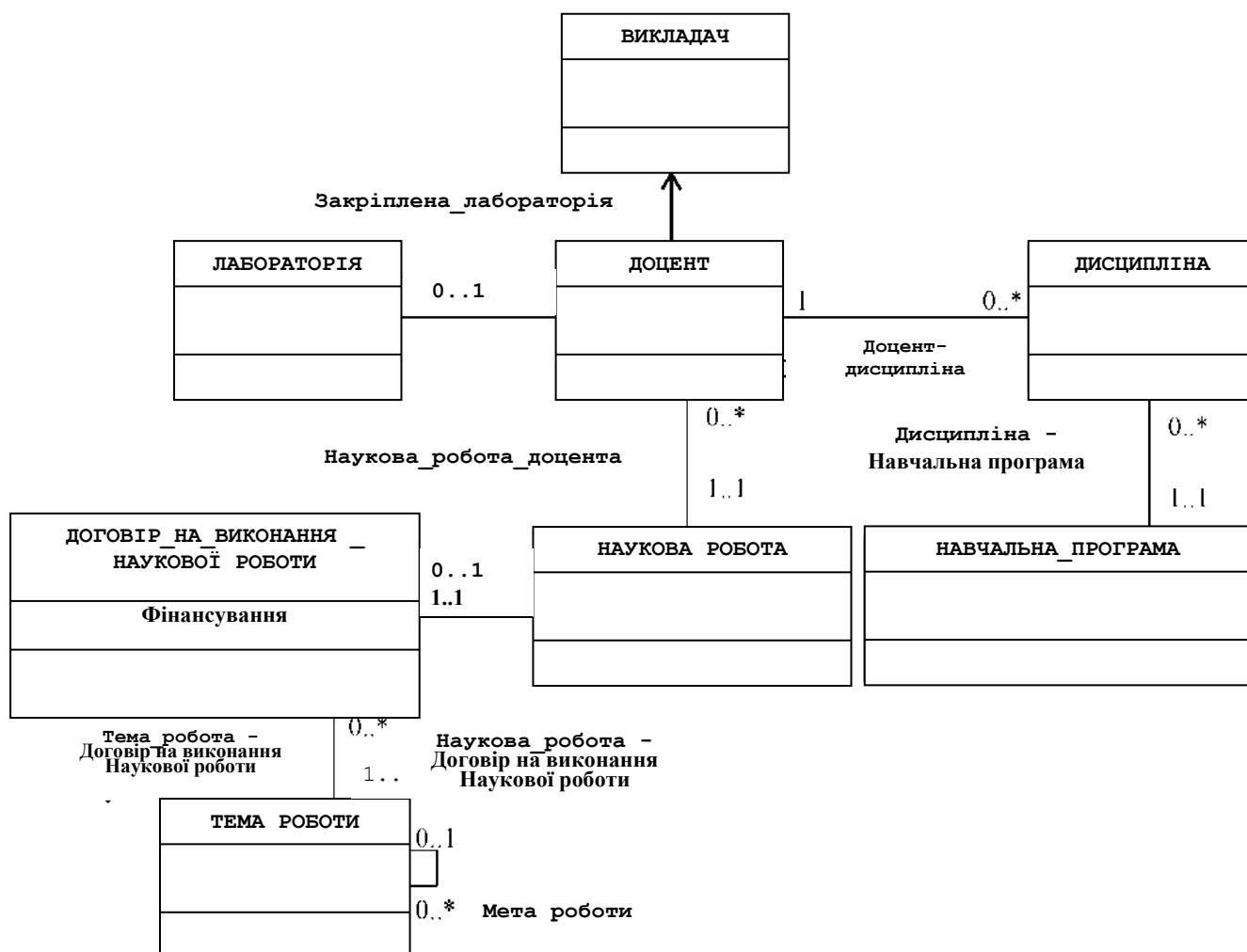


Рис. 3.15. UML-версія діаграми «сутність-зв'язок» з рис. 3.11.

Конструкції ООП, введені мовою UML

Так як UML є об'єктно-орієнтованою технологією, до класів сутностей UML були додані деякі конструкції ООП. По-перше, класи всіх сутностей, які повинні зберігатися в базі даних, позначаються стереотипом «Persistent» (стійкий). Це означає, що існування даних повинне продовжуватися навіть після того, як буде зруйнований об'єкт, який їх обробляв. Простіше кажучи, це означає, що клас сутності повинен зберігатися в базі даних.

Далі, UML допускає призначення атрибутів класам сутностей. Атрибути класу (class attributes) відрізняються від атрибутів сутностей тим, що вони належать всьому класу сутностей даного типу. Так, на рис. 3.16 атрибут Кількість пацієнтів сутності СТУДЕНТ (пацієнт) є атрибутом всієї сукупності сутностей цього типу, наявних у базі даних. Джерело надходження - це атрибут, документуючий джерело надходження всіх пацієнтів, присутніх у базі даних.

Як ви пізніше дізнаєтеся, в рамках реляційної моделі такі атрибути класів просто ніде зберігати. Замість того щоб зберігати атрибути подібні до атрибуту Число пацієнтів в базі даних, вони іноді обчислюються на етапі виконання

програми. В інших випадках для зберігання цих атрибутів виділяється спеціальний клас сутностей. Для класу сутностей **СТУДЕНТ** (пацієнт), зображеного на рис. 3.16, можна створити нову сутність під назвою **ДЖЕРЕЛО НАДХОДЖЕННЯ** пацієнтів, що мають атрибути **Кількість пацієнтів** і **Джерело надходження**. У такому випадку всі сутності класу **СТУДЕНТ** (пацієнт) будуть пов'язані з сутністю **ДЖЕРЕЛО НАДХОДЖЕННЯ ПАЦІЄНТА**.

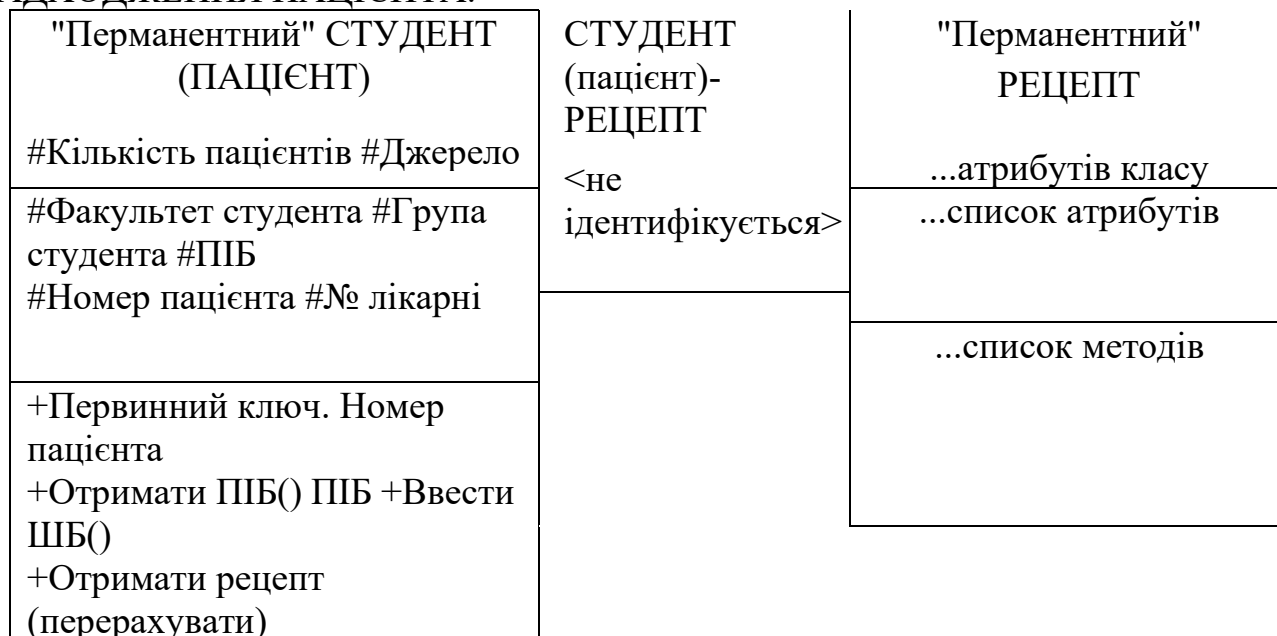


Рис. 3.16. Представлення класів сутностей в UML за допомогою конструкцій ООП.

Третьою новою особливістю є те, що **UML** використовує *об'єктно-орієнтовану нотацію* для позначення видимості атрибутів і методів. Атрибути, назвам яких передують знак «+», є відкритими, атрибути зі знаком «#» є захищеними, а зі знаком «-» - закритими. На рис. 3.16 атрибут **ПІБ** сутності **СТУДЕНТ** (пацієнт) є захищеним.

Ці терміни мають коріння в об'єктно-орієнтованому програмуванні. **Відкритим** (public) називається такий атрибут, який може читатися і змінюватися будь-яким методом будь-якого об'єкта. **Термін захищений** (protected) означає, що атрибут або метод доступний тільки для методів даного класу і його підкласів, а **термін закритий** (private) вказує на те, що відповідний атрибут або метод доступний тільки для методів даного класу.

Нарешті, в **UML** задаються *обмеження і методи*, для чого служить третій сегмент прямокутника, що зображує клас сутностей. На рис. 3.16 на

значення атрибута **Номер пацієнта** накладається обмеження первинного ключа. Це означає, що **Номер пацієнта** є унікальним ідентифікатором. Крім того, рис. 3.16 вказує, що повинні бути створені такі методи: **Отримати ПІБ ()** - для відкритого доступу до атрибуту **ПІБ** зверніть увагу на знак «+» перед **Отримати ПІБ ()**, **Ввести ПІБ ()** - для установки значення цього атрибута, і **Отримати рецепт ()** - для перебору сукупності сутностей класу **РЕЦЕПТ**, пов'язаних з даною сутністю **СТУДЕНТ** (пацієнт).

Семантична об'єктна модель

Семантична об'єктна модель (*semantic object model*) так само як і модель «сутність-зв'язок» використовується для моделювання даних. Як показано на рис. 3.17, команда розробників опитує користувачів, аналізує надані ними звіти, форми та запити і на їх основі будує для користувача модель даних. Ця модель даних в подальшому втілюється в структурі бази даних.

Конкретна форма моделі даних залежить від конструкцій, що використовуються для її побудови. Якщо використовується модель «сутність-зв'язок», конструйована модель буде містити сутності, зв'язки та т.п. У разі використання семантичної об'єктної моделі конструйована модель буде містити семантичні об'єкти та пов'язані з ними конструкції, які описуються в даній частині лекції.

Модель «сутність-зв'язок» і семантична об'єктна модель подібні двом різним лінзам, крізь які дивляться розробники баз даних при вивченні та документуванні користувача даних. Обидві лінзи діють, і обидві вони в кінцевому рахунку втілюються в певній структурі бази даних. Однак, оскільки ці лінзи не однакові і створюють різні зображення, структура баз даних, отриманих з їх допомогою, може дещо відрізнятися. Розробляючи базу даних, ви повинні вирішити, який підхід застосовувати, так само як фотографу доводиться вирішувати, яку використовувати лінзу.

Семантична об'єктна модель була вперше представлена в 1988 р. Вона заснована на концепціях, розроблених і опублікованих Коддом, Хаммером і Мак-Леодом. Семантична об'єктна модель - *це модель даних, що моделює сенс даних користувача.*

Її не слід плутати з об'єктно-орієнтованою обробкою баз даних. Ви дізнаєтеся, чим цілі, властивості і конструкції семантичної об'єктної моделі відрізняються від об'єктно-орієнтованих баз даних.

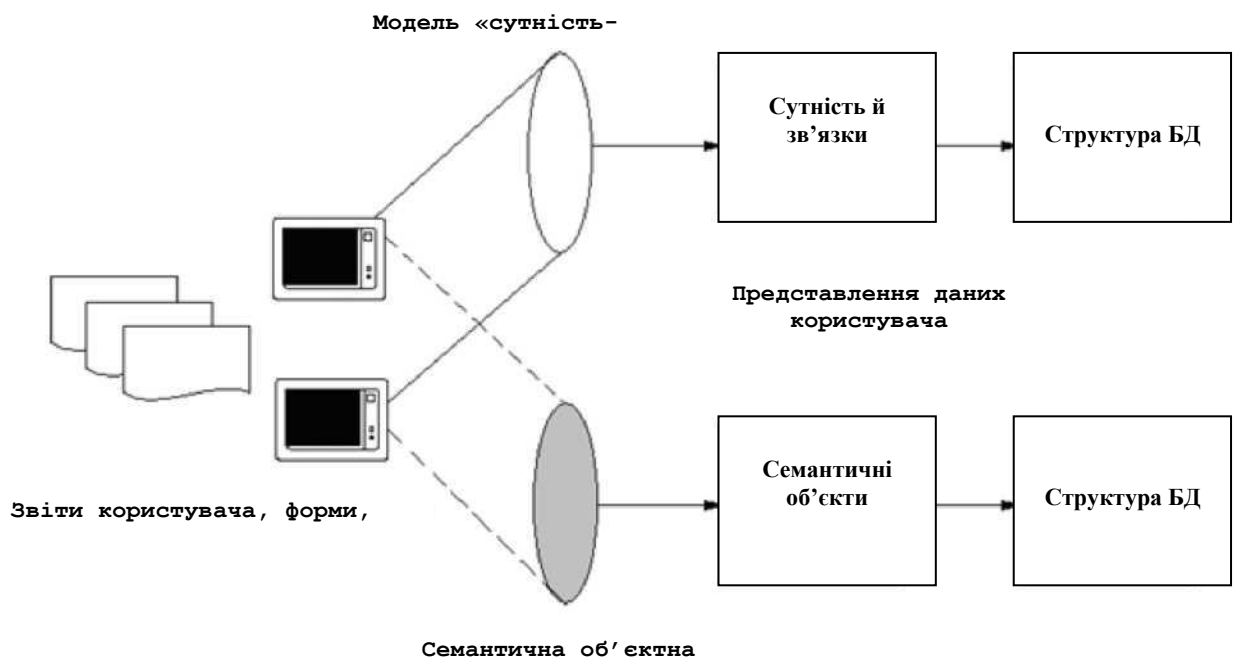


Рис. 3.17. Використання різних моделей при розробці БД

Семантичні об'єкти

Завдання програми бази даних полягає в тому, щоб надавати форми, звіти та запити, за допомогою яких користувачі могли б відстежувати сутності або об'єкти, що представляють інтерес для їх діяльності. *Метою ранніх стадій розробки бази даних є визначення того, які об'єкти фізичного світу повинні бути представлені в базі даних, задання характеристик цих об'єктів і встановлення зв'язку між ними.*

У моделі «сутність-зв'язок» називаються ці об'єкти сутностями.

При семантичному моделюванні будемо їх називати *семантичними об'єктами* (semantic objects) або просто об'єктами.

Слово семантичний означає «смісловий», і семантичний об'єкт – *це* об'єкт, який певною мірою моделює сенс даних користувача.

Семантичні об'єкти моделюють сприйняття користувача більш точно, ніж модель «сутність-зв'язок». Ми використовуємо зі словом об'єкт прикметник семантичний, щоб відрізнити об'єкти, про які йде мова в цій лекції, від об'єктів, якими оперують об'єктно-орієнтовані мови програмування.

Визначення семантичних об'єктів

Сутності і об'єкти в деяких відношеннях схожі, але у них є і відмінності. Ми почнемо зі схожості.

Семантичний об'єкт - *це подання деякої речі, ідентифікованої в робочому середовищі користувача.*

Якщо висловлюватися більш формально, **семантичний об'єкт** – *це* іменована сукупність атрибутів, які в достатній мірі описують окремий феномен.

Подібно сутностям, семантичні об'єкти групуються в класи. У об'єктного класу є ім'я, яке відрізняє його від інших класів і відповідає іменам речей, які представляються цим класом. Так, в базі даних, яка містить інформацію про студентів, мається об'єктний клас під назвою СТУДЕНТ. Зверніть увагу, що імена об'єктних класів, як і імена класів сутностей, пишуться великими літерами. Окремий семантичний об'єкт являє собою екземпляр класу. Наприклад, Бойко Н.В. є екземпляром класу СТУДЕНТ, а Кафедра економіки є екземпляром класу КАФЕДРА.

Подібно сутностям, об'єкт має набір атрибутів. Кожен атрибут описує одну з характеристик представляє мого феномена. Наприклад, об'єкт СТУДЕНТ може мати атрибути ПІБ студента, Спеціальність, Номер групи, Рік народження, Середній бал. Цей набір атрибутів також є *достатнім описом* (sufficient description), тобто він описує всі характеристики, необхідні користувачам для роботи. Достатність опису означає також, що об'єкти є *самодостатніми*. Наприклад, всі необхідні дані про студентів містяться в об'єкті ДЕКАНАТ, куди при необхідності можна звернутися, таким чином, об'єкт ДЕКАНАТ є єдиним достовірним джерелом даних.

Об'єкти представляють окремі феномени (distinct identity), тобто у сприйнятті користувачів вони є чимось незалежним і самостійним, що вимагає врахування.

Феномени - *це сутності, інформація про які необхідна.*

Щоб краще усвідомити значення терміна окремий феномен, згадайте, що⁷¹

існує різниця між об'єктами та екземплярами об'єктів

Нарешті, варто відзначити, що феномени, що подаються об'єктами, можуть існувати фізично, як наприклад, об'єкти класу СТУДЕНТ, а також можуть і не існувати, як об'єкти класу ПРЕДМЕТ.

Таким чином, щоб щось могло вважатися об'єктом, йому не обов'язково існувати фізично - потрібно лише, щоб це *щось* мало *самостійне значення в поданні користувачів*.

Атрибути

Семантичні об'єкти мають атрибути, що описують їх характеристики. Є три типи атрибутів.

Прості атрибути (*simple attributes*) складаються з одного елемента.

Прикладами можуть бути атрибути Номер групи, Рік народження, Середній бал.

Групові атрибути (*group attributes*) являють собою сукупності з атрибутів.

Наприклад, атрибут ПІБ студента, що включає в себе атрибути {Прізвище, Ім'я, По батькові}.

Семантичні об'єктні атрибути (*semantic object attributes*) - це атрибути, які встановлюють зв'язок між двома семантичними об'єктами.

Щоб краще зрозуміти ці визначення, погляньте на рис. 3.18а, який являє собою приклад *семантичної об'єктної діаграми* (*semantic object diagram*), або просто *об'єктної діаграми* (*object diagram*).

Об'єкт ФАКУЛЬТЕТ містить приклад кожного з трьох типів атрибутів. Атрибути Назва факультету, Номер телефону деканату і Номер факсу є простими: кожен з них представляє один елемент даних. Місцева адреса - груповий атрибут, що складається з простих атрибутів Корпус і Кабінет деканату. Нарешті, КАФЕДРА, викладачі та студенти - це семантичні об'єктні атрибути, тобто ці об'єкти пов'язані з об'єктом ФАКУЛЬТЕТ і логічно містяться в ньому.

Сенс цих об'єктних атрибутів, або *об'єктних посилань* (*object links*), як їх іноді називають, полягає в тому, що коли користувач думає про певний факультеті, він має на увазі не тільки назва факультету, локальний адресу, номер телефону деканату та номер факсу, але також кафедру, що знаходиться в цьому факультеті, професорів, які викладають на ньому, і студентів, що займаються на даному факультеті. Оскільки КАФЕДРА, викладачі та студенти також є об'єктами, повна модель даних містить діаграми і для них. Об'єкт КАФЕДРА несе в собі атрибути кафедри, об'єкт ВИКЛАДАЧ - атрибути членів професорсько-викладацького складу, а об'єкт СТУДЕНТ містить атрибути студентів.

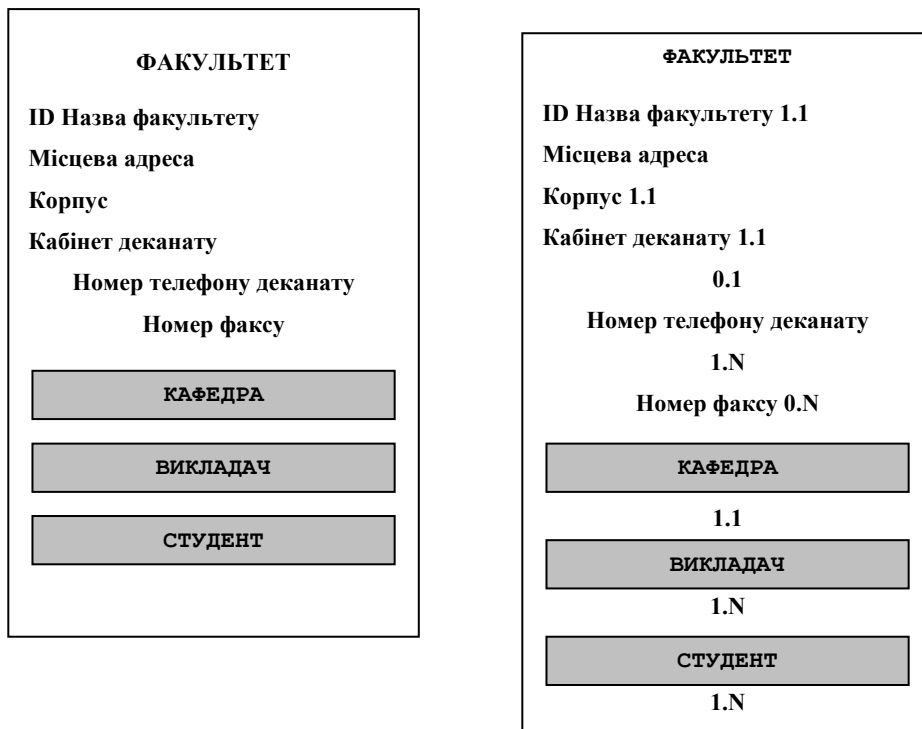


Рис. 3.18. Діаграма об'єкта КАФЕДРА: а - об'єкт КАФЕДРА; б - об'єкт КАФЕДРА з кардинальними числами.

Кардинальне число атрибуту

Кожен атрибут семантичного об'єкта має максимальне і мінімальне кардинальні числа.

Мінімальне кардинальне число показує кількість екземплярів атрибуту, які повинні існувати, щоб об'єкт був допустимим.

Зазвичай це число дорівнює 0 або 1. Якщо воно дорівнює 0, атрибут не зобов'язаний мати значення, а якщо 1, то атрибут зобов'язаний мати значення. Хоча це і незвично, мінімальне кардинальне число іноді може бути більше одиниці. Наприклад, атрибут Гравець в об'єкті під назвою БАСКЕТБОЛЬНА_КОМАНДА може мати мінімальне кардинальне число, рівне 5, оскільки таке найменше число гравців, необхідне для створення баскетбольної команди.

Максимальне кардинальне число показує максимальну кількість екземплярів атрибуту, яке може мати об'єкт.

Зазвичай воно дорівнює 1 або N. Якщо воно дорівнює 1, атрибут може мати не більше одного екземпляру; якщо воно дорівнює N атрибут може мати багато екземплярів, і гранична кількість не задана. Іноді максимальне кардинальне число дорівнює певному числу. Нехай максимальне кардинальне число дорівнює 5 - це означає, що об'єкт може мати не більше п'яти примірників атрибута. Наприклад, атрибут Гравець в об'єкті БАСКЕТБОЛЬНА_КОМАНДА може мати максимальне кардинальне число, рівне 15, і це означатиме, що до складу команди може бути включено не більше 15 гравців.

Кардинальність зображується у вигляді нижнього індексу атрибуту в форматі КМ, де N - мінімальне кардинальне число, а М - максимальна.

Кардинальні числа груп і атрибутів груп, як правило, невеликі. Візьмемо атрибут Місцева адреса. Його кардинальність 0.1, тобто факультет не зобов'язаний мати адресу, але якщо має, то тільки одну. Тепер розглянемо прості атрибути, з⁷³

яких складається атрибут Місцева адреса. Як Корпус, так і Кабінет деканату мають кардинальність 1.1. Ви можете здивуватися, яким чином виходить, що група може бути необов'язковою, якщо атрибути, складові цієї групи, є обов'язковими. Справа в тому, що кардинальні числа діють тільки між атрибутом і його контейнером (групою, яка містить цей атрибут). Мінімальна кардинальне число атрибуту Місцева адреса показує, що цей атрибут не зобов'язаний мати значення, тобто факультет не зобов'язаний мати адресу. А мінімальні кардинальні числа атрибутів Корпус і Кабінет деканату показують, що ці атрибути повинні існувати в атрибуті Місцева адреса. Таким чином, група Місцева адреса не зобов'язана існувати, але якщо вже вона існує, то складові її атрибути Корпус і Кабінет деканату повинні мати значення.

Екземпляри об'єктів

Об'єктні діаграми для факультету, показані на рис. 3.18, являють собою просто формат, загальну структуру, яку можна віднести до будь-якого факультету. На рис. 3.19 представлений екземпляр об'єкта ФАКУЛЬТЕТ. Тут вказані значення кожного атрибута конкретного факультету. Факультет має назву «Інститут телекомунікаційних систем», а деканат розташований в кабінеті № 315 корпусу № 30 НТУУ «КПІ». Зверніть увагу, що атрибут Номер телефону деканату має три значення, тобто в кабінеті деканату інституту телекомунікаційних систем є три телефонні лінії. Деканати інших факультетів можуть мати меншу або більшу кількість телефонів, але у кожного деканату є, принаймні, один телефон.

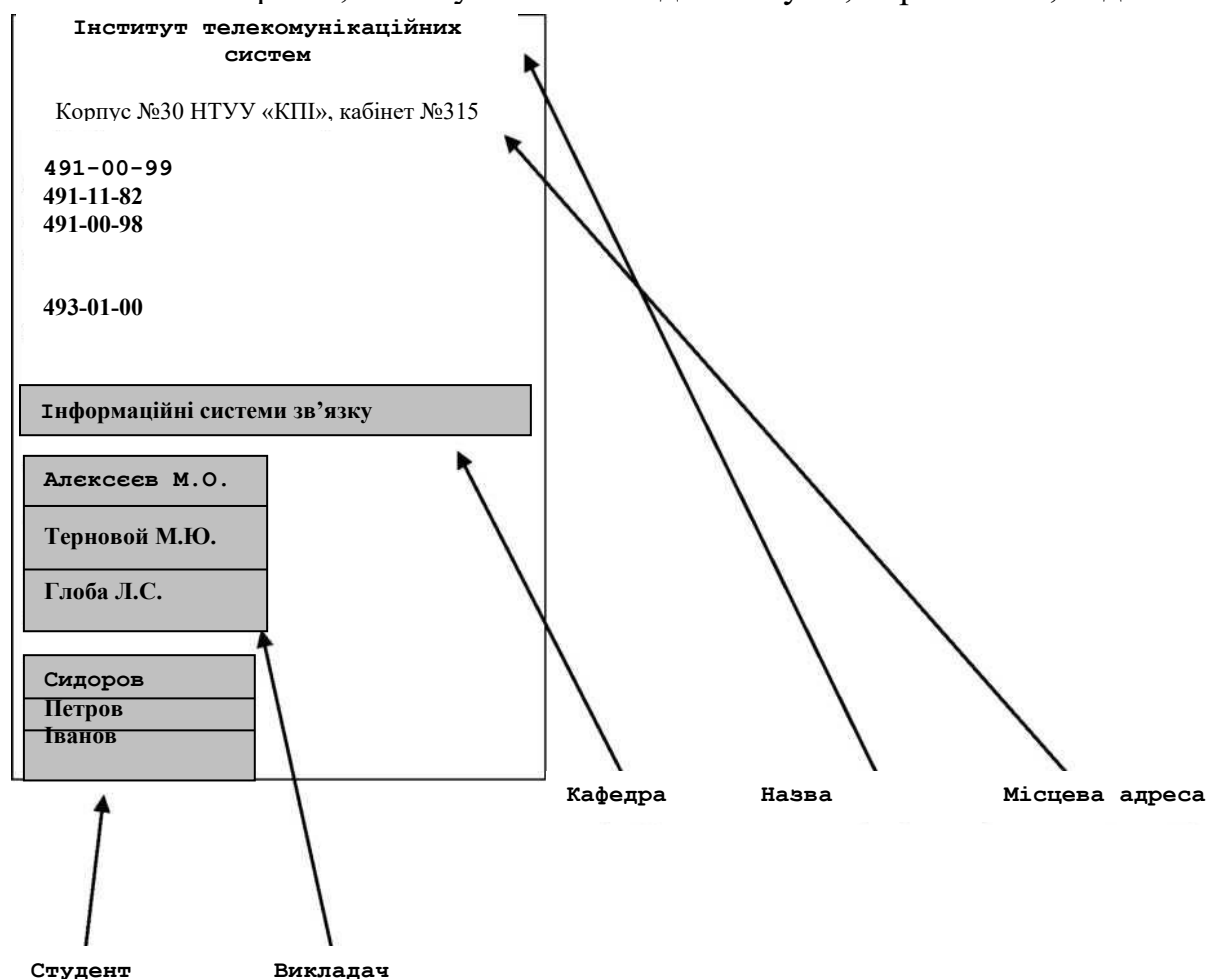


Рис. 3.19. Екземпляр об'єкта ФАКУЛЬТЕТ.

Далі, є один екземпляр атрибута КАФЕДРА, зі значенням Інформаційні системи зв'язку, і множина екземплярів об'єктних атрибутів викладачів і студентів. Кожен з цих об'єктних атрибутів є повноцінним об'єктом і має всі атрибути, визначені для об'єкта даного типу. Щоб не ускладнювати діаграму, ми вказали на ній тільки імена екземплярів об'єктних атрибутів.

Об'єктна діаграма - це картина сприйняття користувачем об'єкта в робочому середовищі.

Парні атрибути

У семантичній об'єктній моделі немає однонапрямлених зв'язків між об'єктами. Якщо один об'єкт містить у собі інший об'єкт, то цей інший буде, у свою чергу, містити в собі перший.

Наприклад, якщо об'єкт ФАКУЛЬТЕТ містить в собі об'єктний атрибут КАФЕДРА, то й об'єкт КАФЕДРА буде містити відповідний об'єктний атрибут ФАКУЛЬТЕТ.

Ці об'єктні атрибути називаються іноді *парними атрибутами* (*paired attributes*), так як вони завжди з'являються по два.

Чому об'єктні атрибути повинні бути парними? Відповідь полягає в тому способі, яким людські істоти уявляють собі зв'язки між об'єктами.

Якщо об'єкт А має зв'язок з об'єктом Б, то об'єкт Б буде мати зв'язок з об'єктом А. Принаймні, Б пов'язаний з А як «річ, яка пов'язана з Б». Якщо цей аргумент здається неочевидним, спробуйте уявити односпрямовану зв'язок між об'єктами. Це зробити неможливо.

Об'єктні ідентифікатори

Об'єктний ідентифікатор (*object identifier*) - це один або кілька об'єктних атрибутів, за допомогою яких користувачі ідентифікують екземпляри об'єктів.

Такі ідентифікатори являють собою потенційні імена семантичного об'єкта.

Для об'єкта ВИКЛАДАЧ можливими ідентифікаторами є Табельний номер та ПІБ викладача. Кожен з цих ідентифікаторів є атрибутом, що розглядаються користувачами в якості допустимого імені для екземплярів об'єктного класу ВИКЛАДАЧ (дані ідентифікатори визначають екземпляр об'єкту класу ВИКЛАДАЧ однозначно). Порівняйте ці ідентифікатори з атрибутами Дисципліна, Кількість лекційних годин і Аудиторія. Такі атрибути не є ідентифікаторами, тому що користувачі не думають про них як про імена екземплярів об'єктів класу ВИКЛАДАЧ (дані ідентифікатори не визначають екземпляр об'єкту класу ВИКЛАДАЧ однозначно).

Груповий ідентифікатор (*group identifier*) - це ідентифікатор, що містить більше одного атрибута.

Прикладами можуть служити ідентифікатори {Прізвище, Ім'я, По батькові} і {Прізвище, Номер телефону}.

Об'єктні ідентифікатори можуть бути *унікальними* або *неунікальними*, залежно від того, як користувачі уявляють собі свої дані.

Наприклад, Номер заліковки є унікальним ідентифікатором об'єкта СТУДЕНТ, а Номер групи не є унікальним ідентифікатором даного об'єкта, оскільки група містить кількох студентів, і ми не зможемо однозначно визначити,⁷⁵

про якого студенті йдеться.

У семантичних об'єктних діаграмах об'єктні ідентифікатори *позначаються за допомогою літер ID* перед атрибутом. Якщо ідентифікатор є унікальний, ці букви підкреслені.

Зазвичай, якщо атрибут повинен використовуватися в якості ідентифікатора, він *зобов'язаний мати значення*. Крім того, як правило, для даного об'єкта є не більше одного ідентифікуючого атрибута. Тому в більшості випадків кардинальність атрибута-ідентифікатора дорівнює 1.1, і це значення ми використовуємо за замовчуванням.

Є, однак, відносно невелике число випадків, коли кардинальність ідентифікатора відрізняється від 1.1. Розглянемо, наприклад, атрибут Ім'я користувача семантичного об'єкта ЛЮДИНА. Людина не зобов'язана мати псевдонім, однак може мати і декілька. Отже, кардинальність цього атрибута дорівнює O.N.

Домени атрибутів

Домен (domain) атрибуту - *це опис множини його значень*.

Характеристики домену залежать від типу атрибута. **Домен простого атрибута** складається з фізичного і семантичного опису. **Фізичний опис** (physical description) показує тип даних (наприклад, число або рядок), довжину даних та інші обмеження (наприклад, вимога, щоб перший символ був буквою або щоб значення не перевищувало 9999.99). **Семантичний опис** (semantic description) вказує функцію, або призначення даного атрибута; він відрізняє цей атрибут від інших атрибутів з тим же фізичним описом.

Наприклад, домен атрибута Назва факультету може бути визначений як «набір рядків довжиною до 7 символів, що представляють найменування факультетів НТУУ «КПІ»». Фраза «набір рядків довжиною до 7 символів» представляє собою фізичний опис домену, а «... що представляють найменування факультетів НТУУ «КПІ»» - семантичний опис. Семантичний опис відрізняє рядки з семи символів, що представляють назви факультетів, від рядків такої ж довжини, які становлять, скажімо, назви навчальних дисциплін, корпусів або ще якісь атрибути.

У деяких випадках фізичний опис домену простого атрибута представляє собою нумерований список - набір окремих значень атрибутів. Доменом атрибута Колір олівця може бути, наприклад, нумерований список {«Синій», «Жовтий», «Червоний»}.

Домен групового атрибуту також має фізичний і семантичний опис. **Фізичний опис** - це список всіх атрибутів в групі в порядку їх слідування. **Семантичний опис** - це опис функції, або призначення, даної групи.

Так, фізичним описом домену Місцева адреса (див. рис. 3.18) є список {Корпус, Кабінет деканату}; семантичним описом є фраза «місце розташування кабінету деканату в корпусі № 30 НТУУ «КПІ»».

Домен об'єктного атрибуту - *це набір екземплярів об'єктів даного типу*.

На рис. 3.18, наприклад, доменом об'єктного атрибуту ВИКЛАДАЧ є сукупність всіх екземплярів об'єктів класу ВИКЛАДАЧ, представлених у базі даних. Доменом об'єкта КАФЕДРА є сукупність всіх об'єктів класу КАФЕДРА, 76

представлених у базі даних.

У певному сенсі, **домен атрибута** - це динамічно нумерований список, який містить всі екземпляри об'єктів даного типу.

Представлення семантичних об'єктів

Користувачі мають доступ до значень об'єктних атрибутів через прикладне програмне забезпечення бази даних, яке надає форми для введення даних, звіти і запити. У більшості випадків такі форми, звіти та запити не вимагають доступу до всіх атрибутів об'єкта.

Наприклад, на рис. 3.20. показані представлення об'єкта ФАКУЛЬТЕТ для двох різних сфер використання прикладних систем. Деякі атрибути об'єкта ФАКУЛЬТЕТ (наприклад, Назва факультету) є видимими в обох представленнях. Інші атрибути видимі тільки в одному з представлень. Наприклад, атрибут СТУДЕНТ є видимим тільки в представленні Список студентів, а атрибут ВИКЛАДАЧ є видимим тільки в представленні Персонал.

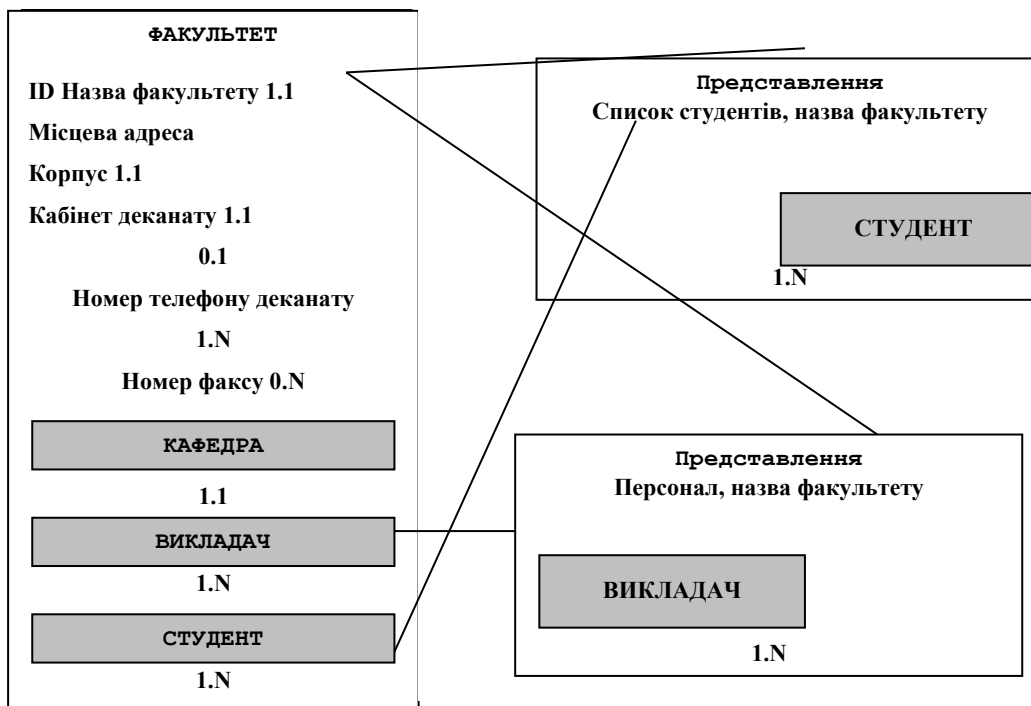


Рис. 3.20. Два представлення семантичного об'єкта ФАКУЛЬТЕТ - Список студентів й Персонал.

ФАКУЛЬТЕТ - Список студентів і Персонал.

Частина об'єкта, видима для конкретної прикладної системи називається **представленням семантичного об'єкта** (semantic object view), або просто **представленням** (view).

Представлення складається з імені об'єкта та списку всіх атрибутів, видимих в цьому представленні.

Представлення використовуються двояким чином. Коли ви розробляєте базу даних, ви можете використовувати їх для розробки моделі даних. Погляньте знову на рис. 3.17. Як можна бачити, при розробці моделі даних розробники бази даних

та програм рухаються у зворотному напрямку. Тобто вони починають з форм, звітів і запитів, які користувачі оголошують необхідними, і рухаються назад, до структури бази даних. Для цього команда вибирає необхідну форму, звіт або запит і визначає, яке представлення має існувати, щоб можна було створити таку форму, звіт або запит. Потім команда переходить до наступної форми, звіту або запиту і робить те ж саме. Далі отримані два представлення об'єднуються. Описаний процес повторюється доти, поки структура бази даних не буде повністю сформована.

Другий аспект використання представлень виникає, коли структура бази даних вже створена. У цьому разі подання створюються для підтримки нових форм, звітів і запитів на основі існуючої структури бази даних.

Створення семантичних об'єктних моделей даних

У цьому розділі ілюструється процес розробки семантичних об'єктів, в ході якого розробники аналізують інтерфейс програми (форми, звіти, запити) і рухаються у зворотному напрямку, отримуючи з цієї інформації структуру об'єктів. Наприклад, щоб змодельовати структуру об'єкта ФАКУЛЬТЕТ, ми спочатку збираємо всі звіти, форми та запити, пов'язані з факультетом. Виходячи з них, ми визначаємо об'єкт ФАКУЛЬТЕТ, який дозволяє конструювати ці звіти, форми і запити.

Однак для абсолютно нової прикладної системи в розпорядженні розробників немає комп'ютерних звітів, форм і запитів, які можна було б вивчати. У цій ситуації розробники починають з визначення того, які об'єкти користувачі хочуть відстежувати. Потім шляхом опитування користувачів команда з'ясовує, які об'єктні атрибути є важливими. На основі цієї інформації можуть бути створені прототипи форм і звітів, за допомогою яких модель даних буде далі уточнюватися.

Приклад: база даних адміністрації НТУУ «КПІ»

Припустимо, що адміністрація НТУУ «КПІ» бажає вести облік даних по факультетах і кафедрах. Припустимо, далі, що прикладна система повинен генерувати чотири типи звітів (мал. 3.21, 3.23, 3.25 і 3.27). Мета - вивчити ці звіти і, використовуючи інженерний аналіз, визначити, які об'єкти і атрибути повинні зберігатися в базі даних.

Об'єкт ФАКУЛЬТЕТ

Звіт, зображений на рис. 3.21, містить інформацію про факультет, а саме про Інститут телекомунікаційних систем. Даний звіт - це тільки один окремий приклад. НТУУ «КПІ» має подібні звіти і про інші факультети, зокрема про факультет лінгвістики та факультет менеджменту. Створюючи модель даних, важливо зібрати достатню кількість прикладів, щоб на їх основі можна було побудувати зразок звіту про факультет. Тут припустимо, що звіт на рис. 3.21 є типовим.

Досліджуючи звіт, ми знайдемо дані, специфічні для даного факультету (назва, ім'я директора, номер телефону і місцева адреса, а також відомості про кожну з кафедр факультету). Це наводить на думку про те, що база даних може містити об'єкти факультетів і кафедр, а також зв'язки між ними.

Ці попередні висновки відображені в об'єктних діаграмах на рис. 3.22. Зверніть увагу, що ми не вказали кардинальність простих атрибутів, у яких вона дорівнює 0.1.

Інститут телекомунікаційних систем			
Ільченко М.Ю., директор			
Телефон: 236-62-13		Місцева адреса: Корпус № 30 НТУУ «КПІ»	
<u>Кафедра</u>	<u>Завідуючий</u>	<u>Телефон</u>	<u>Кількість студентів</u>
Інформаційно-телекомунікаційні мережі	Глоба Л.С.	236-54-25	156
Телекомунікаційні мережі	Бунін С.Г.	237-98-04	248
Засоби телекомунікацій	Ільченко М.Ю.	237-67-53	293

Рис. 3.21. Приклад звіту про факультет.

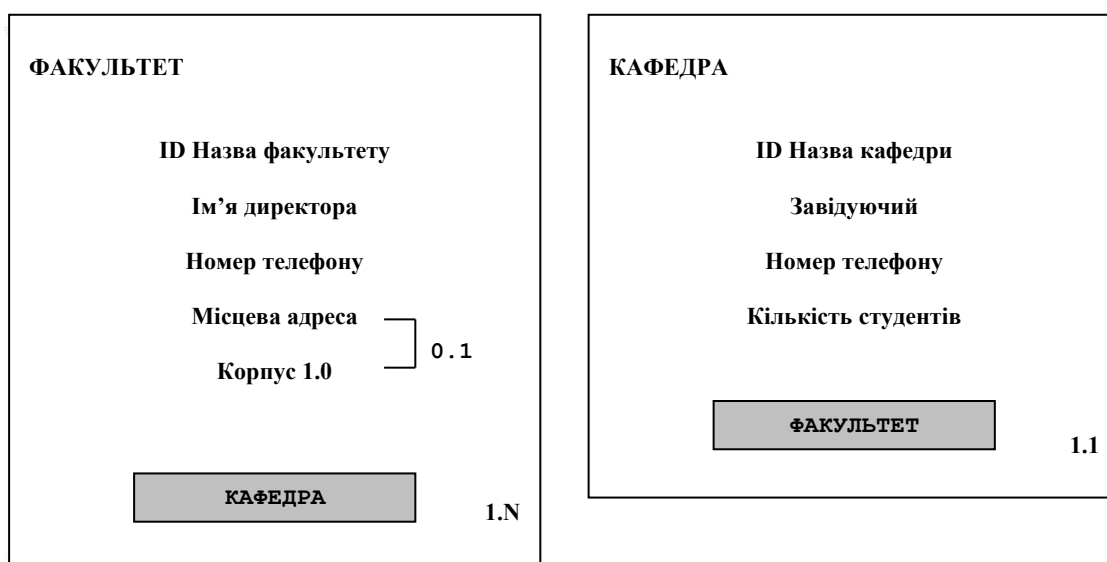


Рис. 3.22. Перша версія об'єктів ФАКУЛЬТЕТ і КАФЕДРА.

Кардинальність атрибуту КАФЕДРА об'єкта ФАКУЛЬТЕТ дорівнює 1.N-це означає, що факультет повинен мати як мінімум одну кафедру і кафедр

може бути багато. Це мінімальне кардинальне число не може бути виведено з звіту на рис. 3.21. Щоб його отримати, користувачам було поставлено питання про те, чи може існувати факультет без кафедр, і їх відповідь була негативною.

Також зверніть увагу, що структура об'єкта КАФЕДРА виведена на основі даних, представлених на рис. 3.21. Оскільки об'єктні атрибути завжди є парними, об'єкт ФАКУЛЬТЕТ показаний всередині об'єкта КАФЕДРА, хоча, строго кажучи, цей факт не можна отримати з аналізу рис. 3.21. Як і в ситуації з атрибутом КАФЕДРА об'єкта ФАКУЛЬТЕТ, користувачів попросили визначити кардинальність атрибуту ФАКУЛЬТЕТ. Вона дорівнює 1.1 що означає, що кафедра може бути пов'язана з одним і тільки одним факультетом.

По ходу викладу звіт на рис. 3.21 інтерпретували таким чином, що групи повторюваних даних належать до об'єкта КАФЕДРА як до незалежного об'єкту. Фактично наявність таких груп часто є сигналом про те, що існує якийсь інший

об'єкт. Однак це не завжди так. Повторювана група може бути також груповим атрибутом, у якого виявилось кілька значень.

Можливо, буде цікаво, як розрізнити ці два типи повторюваних даних - ті, які представляють об'єкт, і ті, які представляють групу. Якого-небудь твердого та чіткого правила на цей рахунок не існує, оскільки відповідь залежить від того, як користувачі уявляють собі свій світ. Отже, найкращим рішенням буде запитати самих користувачів про семантику їх даних. Запитайте, чи є ці повторювані групи даних частиною факультету, або вони відносяться до чогось ще - чогось самостійного. У першому випадку вони становлять *груповий атрибут*, у другому - *семантичний об'єкт*. Подивіться також інші звіти (або форми, або запити). Чи є у користувача звіти по кафедрах? Якщо так, то припущення про те, що кафедра може бути представлена семантичним об'єктом, підтвердиться. Насправді персонал НТУУ «КПІ» працює з двома видами звітів по кафедрах. Цей факт ще більше підтверджує думку про те, що слід ввести об'єкт під назвою КАФЕДРА.

Далі, групи атрибутів, що представляють незалежний об'єкт, зазвичай містять явний ідентифікуючий атрибут або декілька атрибутів.

Об'єкт КАФЕДРА

Звіт, представлений на рис. 3.23, містить інформацію про кафедри, а також список працюючих на них викладачів. Зверніть увагу, що цей звіт містить місцеву адресу кафедри. Оскільки ці дані не присутні у версії об'єкта КАФЕДРА, зображеної на рис. 3.22, їх необхідно додати до об'єкта, як показано на рис. 3.24. Таке уточнення є типовим для процесу моделювання даних. Тобто структура семантичних об'єктів постійно уточнюється в міру того, як ідентифікуються і аналізуються нові звіти, форми і запити

Кафедра інформаційно-телекомунікаційних мереж		
Інститут телекомунікаційних систем		
Завідуючий:	Глоба Л.С	
Телефон:	236-54-25	
Місцева адреса:	Корпус № 30 НТУУ «КПІ», кабінет № 409	
<u>Викладач</u>	<u>Кабінет</u>	<u>Телефон</u>
Алексєєв М.О.	Корпус № 30 НТУУ «КПІ», кабінет № 316	248-08-16
Терновой М.Ю.	Корпус № 30 НТУУ «КПІ», кабінет № 316	248-08-16
Ладик А.І.	Корпус № 30 НТУУ «КПІ», кабінет № 210	254-21-67

Рис. 3.23. Приклад звіту про кафедру

Об'єкт ВИКЛАДАЧ

Звіт на рис. 3.23 не тільки вказує на необхідність введення об'єкта КАФЕДРА, а й наводить на думку, що для представлення даних про професора може знадобитися ще один

об'єкт. Відповідно, в модель доданий об'єкт ВИКЛАДАЧ, як показано на рис. 3.24. Ідентифікатор об'єкта ВИКЛАДАЧ, яким є атрибут ПІБ викладача, не є⁸⁰

унікальним; на це вказує той факт, що букви ГО на рис. 3.24 не підкреслені.

Відповідно з об'єктними діаграмами на рис. 3.24, на кожній кафедрі повинен бути як мінімум один викладач, причому на одній кафедрі викладачів може бути декілька, але кожен викладач повинен працювати на одній і тільки на одній кафедрі. Таким чином, відповідно до цієї моделі, робота за сумісництвом заборонена. Це обмеження є частиною ділового регламенту, який повинен визначатися з опитувань користувачів.

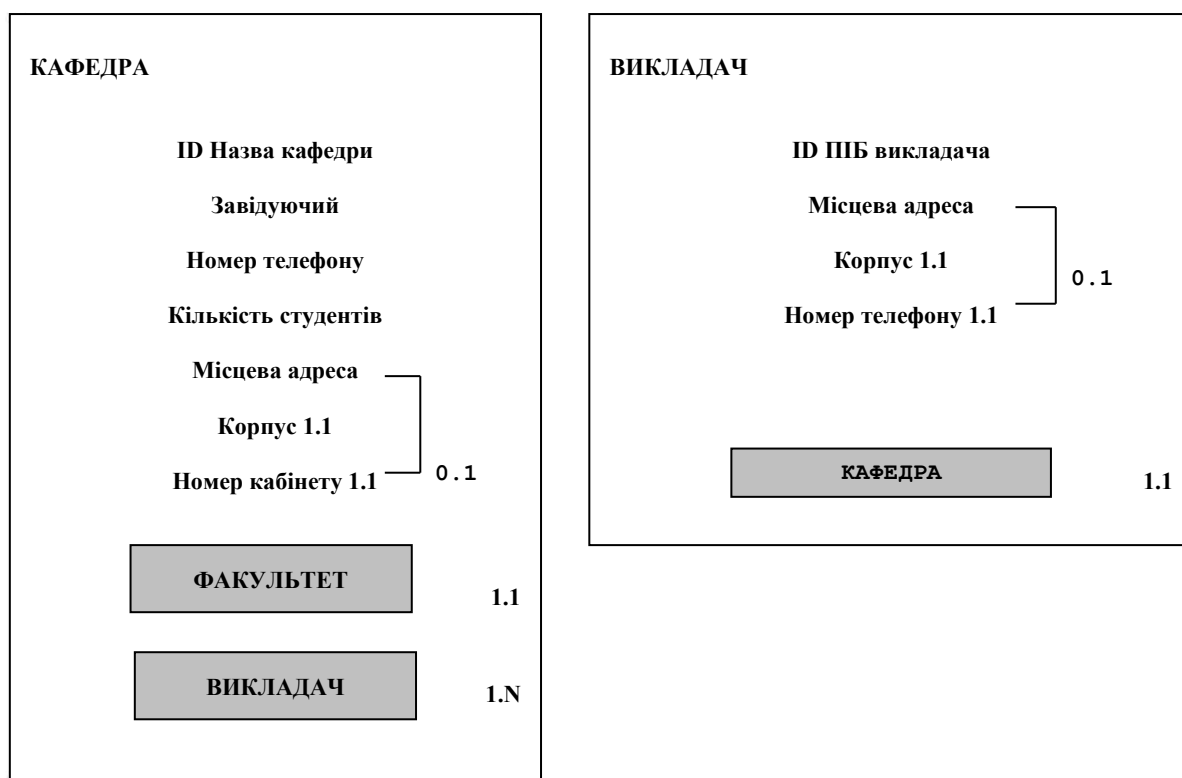


Рис. 3.24. Уточнений об'єкт КАФЕДРА і новий об'єкт ВИКЛАДАЧ.

На рис. 3.25 показаний другий звіт про кафедру. У ньому представлена інформація про кафедру і про студентів, спеціальність яких має відношення до цієї кафедри. Ситуація, коли про один об'єкт є два звіти, типова; ці звіти просто документують різні подання одного і того ж об'єкта реального світу. Більше того, існування другого звіту зміцнює впевненість у тому, що кафедра є об'єктом в розумінні користувачів.

Об'єкт СТУДЕНТ

Звіт на рис. 3.25 містить дані про студентів, профілюючи спеціальність яких відноситься до області діяльності даної кафедри, тобто студенти також є об'єктами. Тому об'єкт КАФЕДРА повинен також містити об'єкти студентів і викладачів, як показано на рис. 3.26.

Кафедра інформаційно-телекомунікаційних мереж Інститут телекомунікаційних систем			
Завідуючий: Телефон:		Глоба Л.С 236-54-25	
<u>ПІБ студента</u> <u>Група</u> <u>студента</u> <u>Номер</u> <u>студента</u> <u>Телефон</u>	Сидоров Н.А. ТІ-41 17 254-48-16	Петров М.Ю. ТІ-42 15 447-08-16	Іванов А.І. ТІ-41 6 354-21-67

Рис. 3.25. Другий приклад звіту про кафедру

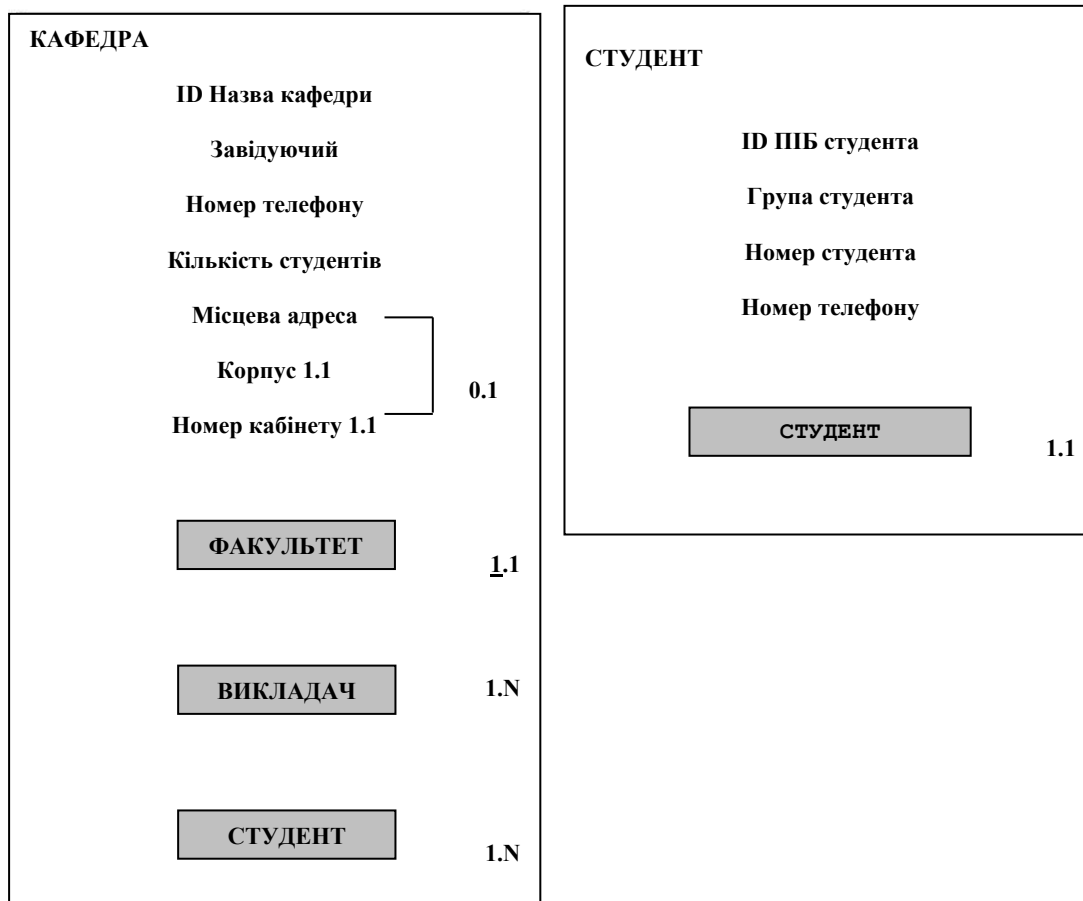


Рис. 3.26. Уточнений об'єкт КАФЕДРА і новий об'єкт СТУДЕНТ.

Об'єкт СТУДЕНТ на рис. 3.26 має атрибути ПІБ студента, Група студента, Номер студента і Номер телефону - атрибути, перелічені в звіті на рис. 3.25. Зверніть увагу, що ПІБ студента *не є унікальним ідентифікатором*.

На рис. 3.27 представлений інший приклад звіту про студентів - лист-запрошення, який університет розсилає вступившим до нього студентам. Навіть, незважаючи на те, що цей лист, він все ж є звітом.

Ті елементи даних у листі, які повинні зберігатися в базі даних, надруковані жирним шрифтом. Крім даних, що відносяться до студента, лист також містить дані про профільну кафедру студента і про його керівника. Оскільки керівник є викладачем, цей лист підтверджує потребу в окремому об'єкті під назвою ВИКЛАДАЧ.

Петрову Олегу Олександровичу
м. Київ
вул. Ставропольська, буд.37, кв. 42

Шановний Олегу Олександровичу,

Радий повідомити, що ви прийняті на кафедру **інформаційно-телекомунікаційні мережі** Інституту Телекомунікаційних Систем НТУУ «КПІ», починаючи з **осіннього семестру 2012 року**. Кафедра інформаційних систем зв'язку знаходиться за адресою: № 30 НТУУ «КПІ», кабінет **316**.

Вашим керівником призначена завідувача кафедри Глоба Л.С., її номер телефону 236-54-25. кабінет 409 розташований в корпусі № 30 НТУУ «КПІ». Прохання призначити зустріч з керівником до початку семестру.

Вітаю, і ласкаво просимо в НТУУ «КПІ»!

Ільченко М.Ю.

Директор Інституту Телекомунікаційних Систем

Рис. 3.27. Лист-запрошення.

Уточнені діаграми об'єктів викладачів і студентів представлені на рис.3.28, Дивлячись, на об'єкт **СТУДЕНТ**, можна бачити, що як **КАФЕДРА**, так і **ВИКЛАДАЧ** мають єдине значення (їх максимальне кардинальне число дорівнює 1). Отже, студент цього університету має максимум одну профільну кафедру і одного керівника, причому має обов'язково.

Об'єкт **СТУДЕНТ** на рис. 3.28 відповідає даним, зображеним на рис.3.27. Може, однак, виявитися, що студент має більше однієї спеціальності, і тоді атрибути **КАФЕДРА** і **ВИКЛАДАЧ** будуть багатозначними. Цей факт не може бути виявлений з єдиного стандартного листа, який було представлено, тому потрібно проаналізувати інші листи і провести додаткові опитування, щоб визначити, чи дозволено мати кілька профільюючих спеціальностей. Тут ми припускаємо, що така спеціальність може бути тільки одна.

ПІБ студента дається спочатку у форматі **Прізвище Ім'я По батькові** у верхній сходитці листи, а потім у форматі **Ім'я По батькові** у привітанні. ПІБ керівника також дається у форматі **{Прізвище, Ім'я, По батькові}**

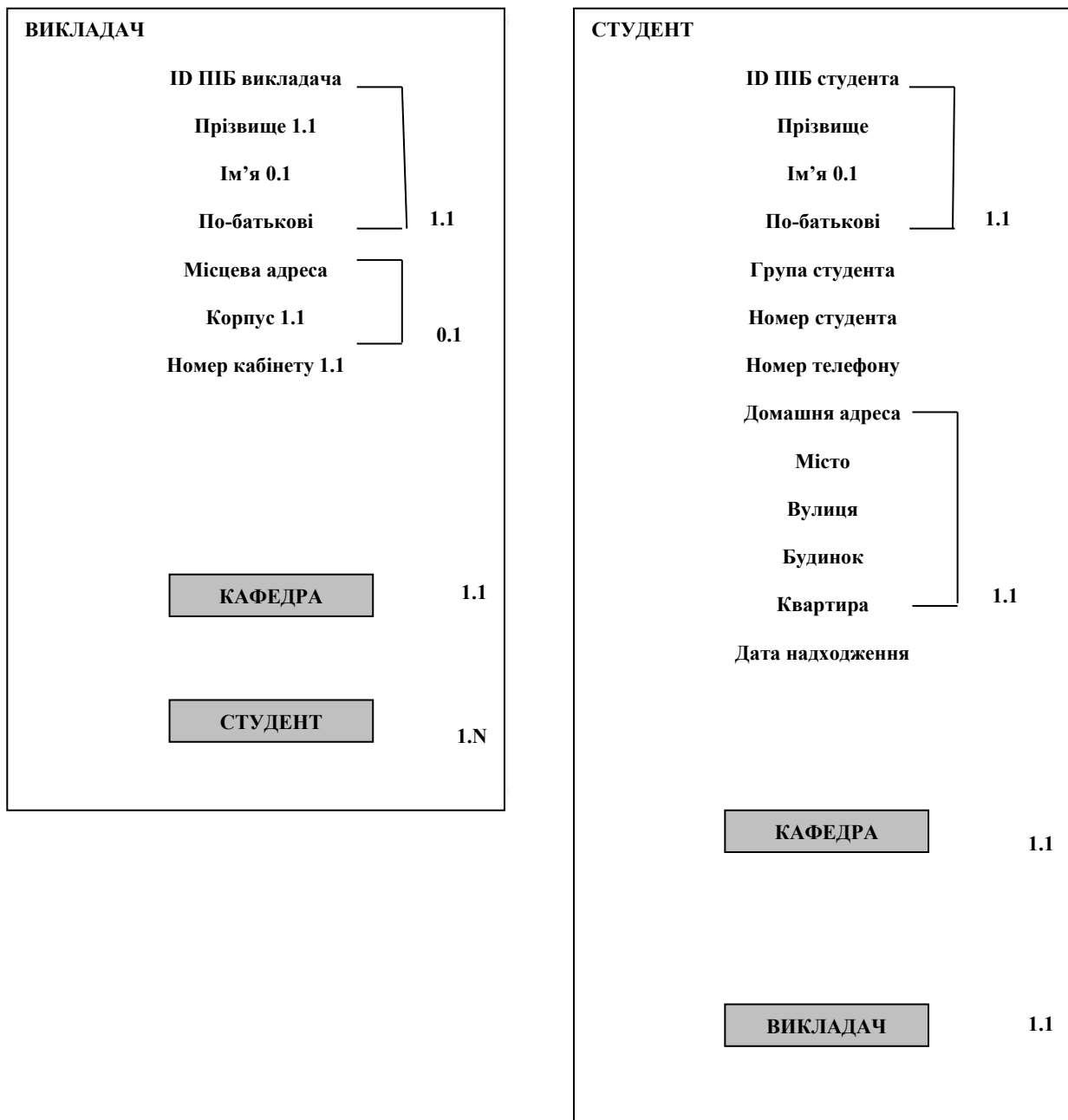


Рис. 3.28. Уточнені об'єкти ВИКЛАДАЧ і СТУДЕНТ.

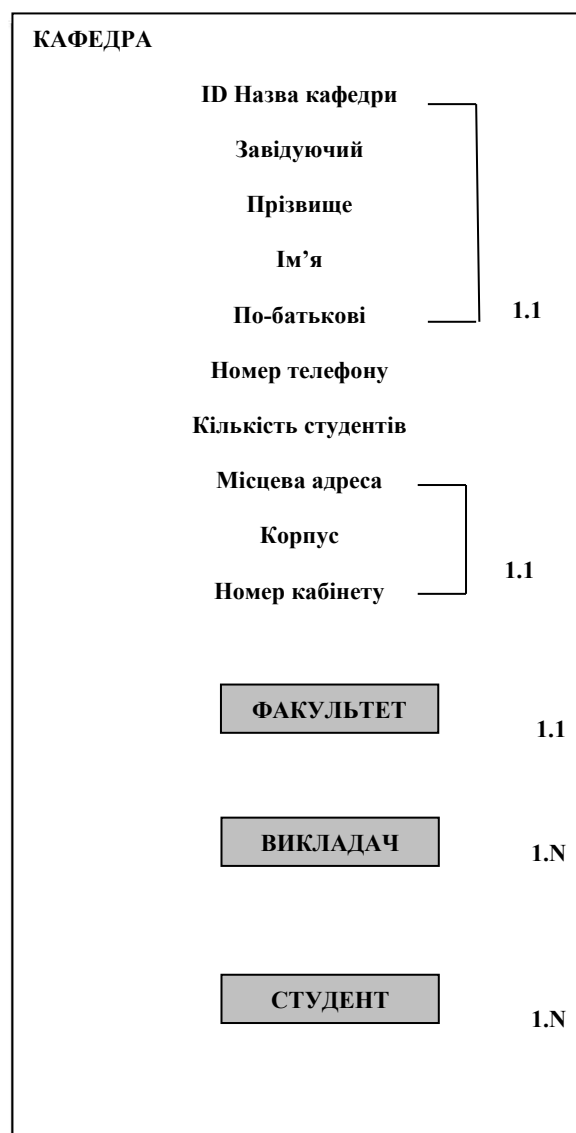
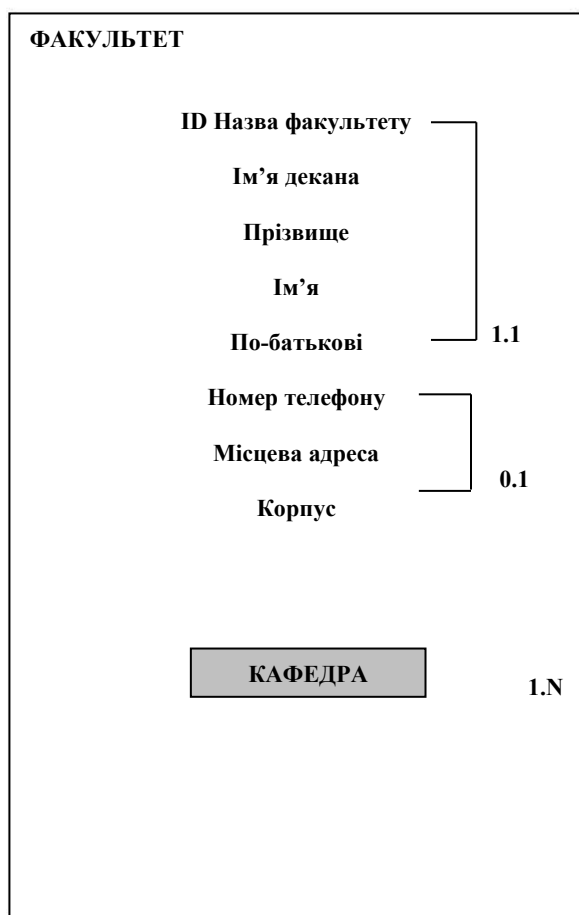
Знову-таки, ці зміни ілюструють ітеративну природу моделювання даних. Наявні рішення часто доводиться переглядати і коригувати знову і знову. Це не означає, що процес проектування йде з помилками; фактично, такі ітерації є звичайними і очікуваними.

Специфікація об'єктів

На рис. 3.29 показаний остаточний варіант об'єктних діаграм для бази даних НТУУ «КПТ». Внесено кілька додаткових змін: атрибути Тм'я декана змодельовані у форматі {Прізвище, Тм'я, По батькові}, щоб всі імена мали однаковий формат. Далі, щоб збільшити точність моделі, атрибут ВИКЛАДАЧ об'єкта СТУДЕНТ був перейменований - тепер він називається КЕРІВНИК (необхідність перейменування полягає в тому, що керівник, наприклад, дипломного проекту може бути викладачем кафедри, а може бути людиною з іншої організації, де студент проходив практику) . Екземпляр об'єкта ВИКЛАДАЧ, пов'язаний через цей

атрибут з екземпляром об'єкта **СТУДЕНТ**, це не просто хтось із викладачів даного студента, а цілком конкретний викладач, який є керівником даного студента, і термін керівник є більш точним, ніж термін викладач.

Домен цього атрибуту не змінюється. Атрибут **КЕРІВНИК** має домен викладач, такий же домен і у атрибуту **ВИКЛАДАЧ**. Він як і раніше вказує на екземпляри семантичного об'єкта **ВИКЛАДАЧ**. Зміна імені - це лише уточнення ролі, яку домен **ВИКЛАДАЧ** грає в семантичних об'єктах **СТУДЕНТ**. Аналогічна зміна була зроблена і в об'єкті **ВИКЛАДАЧ**: атрибут **СТУДЕНТ** був перейменований і називається тепер **КЕРОВАННИЙ_СТУДЕНТ**, цей атрибут і раніше пов'язаний з об'єктами з домену **СТУДЕНТ**.



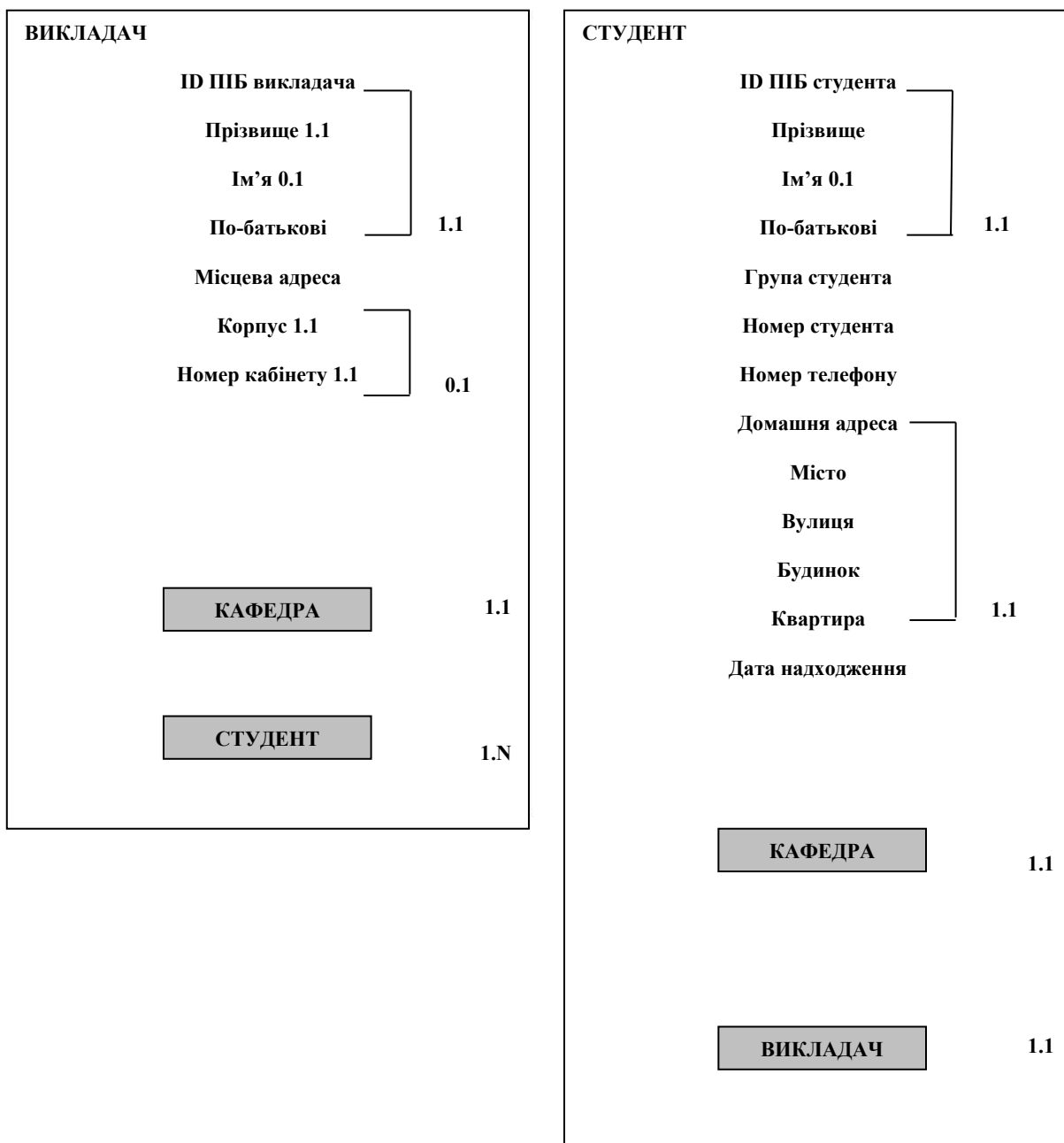


Рис. 3.29. Повний набір семантичних об'єктів

У таблицях 3.1 та 3.2 представлена специфікація моделі даних. Семантичні об'єкти і атрибути визначені в специфікації семантичних об'єктів, а домени визначені в специфікації доменів. Перша таблиця є альтернативним поданням інформації в семантичних об'єктних діаграмах, і її інтерпретація очевидна.

Друга таблиця, таблиця доменів, містить таку інформацію про домени, яка не може бути отримана з семантичних об'єктних діаграм. Як було зазначено раніше, домен має як семантичний, так і фізичний опис. Семантичний опис

кожного об'єкта дається в стовпці Опис, а фізичний опис дається в стовпці Специфікація. Зміст стовпця Опис говорить сам за себе.

Специфікація доменів включає в себе фізичний опис і в деяких випадках множину значень і формат. Домен Номер студента визначений, наприклад, як ціле число, що набуває значення між 00 і 40. Фізичний опис групового домену складається зі списку доменів, що входять до групи. Фізичний опис домену семантичного об'єкта - це просто посилання на опис семантичного об'єкта.

Домен атрибуту Кількість студентів являє собою приклад четвертого типу домену - формули (formula domain). Формули представляють атрибути, обчислені на основі інших значень. Домен Число студентів - це число об'єктів типу СТУДЕНТ, пов'язаних з даним об'єктом типу КАФЕДРА. У даному випадку не представлені способи, за допомогою яких це обчислення описується у визначенні домену. На даному етапі важливо тільки вказати потребу у формулі і її специфікацію.

Таблиця 3.1. Специфікації семантичних об'єктів для бази даних НТУУ «КПІ».

Ім'я об'єкта	Ім'я властивості	Мін.	Макс.	Статус	Ім'я домену
ФАКУЛЬТ	Назва факультету	1	1	<i>ID</i>	Назва
	Ім'я декана	1	1		Ім'я людини
	Прізвище	1	1		Прізвище
	Ім'я	0	1		Ім'я
	По-батькові	0	1		По-батькові
	Телефон	0	1		Телефон
	Місцева адреса	0	1		Місцева адреса
	Корпус	1	1		Корпус
	КАФЕДРА	1	N		КАФЕДРА
	Назва кафедри	1	1		Назва кафедри
КАФЕДРА	Завідуючий	1	1	<i>ID</i>	Ім'я людини
	Прізвище	1	1		Прізвище
	Ім'я	0	1		Ім'я
	По-батькові	0	1		По-батькові
	Телефон	0	1		Телефон
	Кількість	0	1		Кількість
	Місцева адреса	0	1		Місцева адреса
	Корпус	1	1		Корпус
	Номер кабінету	1	1		Номер кабінету
	ФАКУЛЬТЕТ	1	1		ФАКУЛЬТЕТ
ВИКЛАДА	ВИКЛАДАЧ	1	N	<i>ID</i>	ВИКЛАДАЧ
	СТУДЕНТ	1	N		СТУДЕНТ
	ПІБ викладача	1	1		ПІБ людини
	Прізвище	1	1		Прізвище
	Ім'я	0	1		Ім'я
	По-батькові	0	1		По-батькові
	Місцева адреса	0	1		Місцева адреса
	Корпус	1	1		Корпус
	Номер кабінету	1	1		Номер кабінету
	Телефон	0	1		Телефон
СТУДЕНТ	КАФЕДРА	1	1	<i>ID</i>	КАФЕДРА
	КЕРОВАННИЙ СТУ	1	N		СТУДЕНТ
	ПІБ студента	1	1		ПІБ людини
	Прізвище	1	1		Прізвище
	Ім'я	0	1		Ім'я
	По-батькові	0	1		По-батькові
	Група студента	1	1		Група студента
	Номер студента	1	1		Номер студента
	Телефон	0	1		Телефон
	Домашня адреса	1	1		Адреса
	Місто	1	1		Місто
	Вулиця	1	1		Вулиця
	Будинок	1	1		Будинок
	Квартира	1	1		Квартира
	Дата надходження	0	1		Квартальна дата
	КАФЕДРА	1	1		КАФЕДРА
	КЕРІВНИК	1	1		ВИКЛАДАЧ

Таблиця 3.2. Специфікація доменів для бази даних НТУУ «КПІ»

Ім'я домену	Тип	Семантичний опис	Фізичний опис
Адреса	Груповий	Адреса в Україні	Місто Вулиця Дім
Корпус	Простий	Назва корпусу в університеті	Текст 20
Місцева адреса	Груповий	Адреса в університеті	Корпус
Місто	Простий	Назва міста	Текст 20
ФАКУЛЬТЕТ	Семантичний	Один із факультетів НТУУ «КПІ»	Див. специфікацію
Назва факультету	Простий	Офіційна назва факультету НТУУ	Текст 25
КАФЕДРА	Семантичний	Навчальна кафедра університету	Див. специфікацію
Назва кафедри	Простий	Офіційна назва навчальної кафедри	Текст 25
Ім'я	Простий	Частина групи ім'я людини,	Текст 20
Прізвище	Простий	Частина групи ім'я людини,	Текст 20
По-батькові	Простий	Частина групи ім'я людини, яка представляє по-батькові	Текст 20

Кількість студентів	Формула	Число студентів на даній кафедрі	Integer; значення в інтервалі {0..60}
Номер кабінету	Простий	Номер кабінету в	Текст 5
Ім'я людини	Груповий	Прізвище, ім'я, по-батькові	Прізвище Ім'я По батькові
Телефон	Простий	Номер телефону	Текст 7
Викладач	Семантичний об'єкт	Ім'я постійного члена Професорсько-	Див. специфікацію семантичних
Квартальна дата	Простий	Навчальний семестр і рік	Текст 3; значення {q01}, де q=1 чи 2, а 01 -
Вулиця	Простий	Назва вулиці	Текст 30
Дім	Простий	Номер дому	Текст 5
Квартира	Простий	Номер квартири	Текст 5
СТУДЕНТ	Семантичний	Особа, прийнята в НТУУ «КПІ»	Див. специфікацію семантичних
Номер студента	Простий	Номер по порядку студента в групі	Integer; інтервал - {00..}
Група студента	Простий	Номер групи в якій студент навчається	Текст 5; значення {AB - 01}, де A і B - букви {А-Я}, а

Типи об'єктів

У цьому розділі описуються і демонструються сім типів об'єктів. Для кожного типу об'єктів ми розглянемо звіт або форму і покажемо, як їх можна представити за допомогою даного об'єкта.

У даному розділі використовуються три нових терміна.

Однозначний атрибут (*single value attribute*) - це атрибут з максимальним кардинальним числом 1.

Багатозначний атрибут (*multi-value attribute*) - це атрибут, що має максимальне кардинальне число більше 1.

Необ'єктний атрибут (*non-object attribute*) - це простий або груповий атрибут.

Прості об'єкти

Простий об'єкт (*simple object*) - це семантичний об'єкт, що має тільки однозначні, прості або групові атрибути.

Приклад такого об'єкта зображений на рис. 3.30. У лівій частині цього рисунка представлені два екземпляри звіту під назвою ЯРЛИК ОБЛАДНАННЯ. Такі ярлики, які наклеюють на лабораторне обладнання, допомагають вести його облік. Їх можна розглядати як звіти.

На рис. 3.30б зображений простий об'єкт ОБЛАДНАННЯ, за допомогою якого моделюється ярлик для обладнання. Атрибути об'єкта включають в себе те, що вказано на ярлику: Інвентарний номер, Опис, Дата придбання і Вартість. Зверніть увагу, що жоден з цих атрибутів не є багатозначним, як жоден з них не є і об'єктним. Отже, об'єкт ОБЛАДНАННЯ є простим об'єктом.

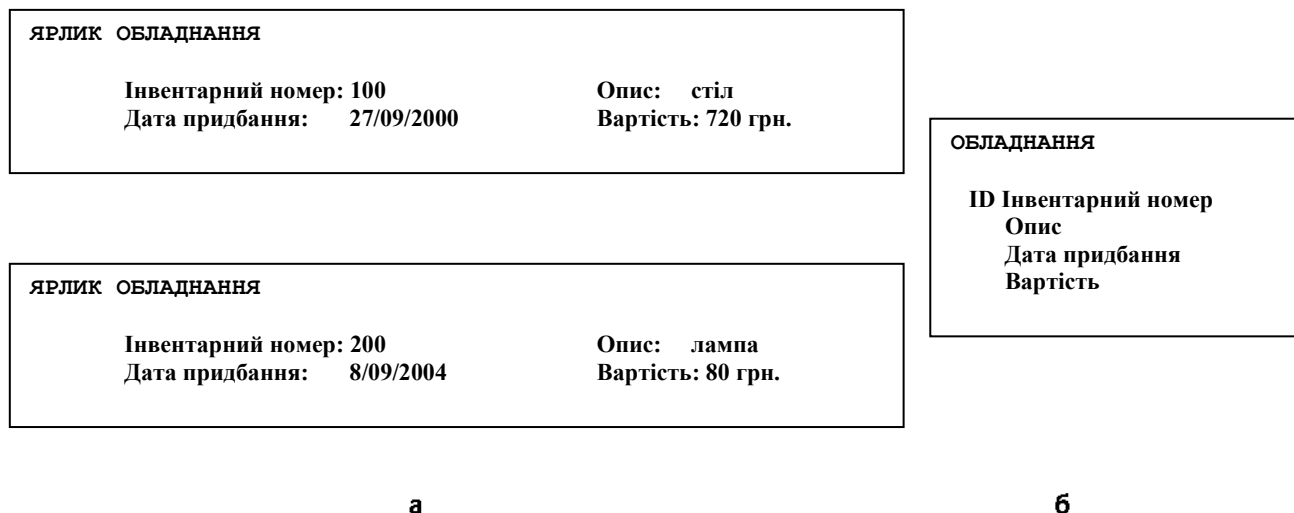


Рис. 3.30 Приклад простого об'єкту : а - звіти, оснований на простому об'єкті; б - простий об'єкт ОБЛАДНАННЯ

Композитний об'єкт

Композитний об'єкт (*composite object*) - це семантичний об'єкт, що містить один або кілька багатозначних, простих або групових атрибутів, але не має об'єктних атрибутів.

Рахунок за послуги санаторію, зображений на рис. 3.31а, демонструє потребу в композитному об'єкті. Цей рахунок містить дані, що відносяться до рахунку в цілому: Номер рахунку, Дата придбання, ПІБ клієнта і Сума до оплати. Крім того, в ньому є повторювана група атрибутів, яка описує надані клієнту послуги. Кожна група включає в себе атрибути Дата придбання послуги, Опис послуги і Вартість.

На рис. 3.31б показана діаграма об'єкта РАХУНОК_ІЗ_САНАТОРІЮ. Атрибут Рядок витрат - це груповий атрибут, що має максимальне кардинальне число N, що означає, що група {Дата надання послуги, Опис послуги і Вартість} може з'являтися багаторазово в одному і тому ж екземплярі семантичного об'єкта РАХУНОК_ТЗ_САНАТОРТЮ.

Рядок витрат не моделюється як самостійний семантичний об'єкт, а вводиться як атрибут об'єкта РАХУНОК_ТЗ_САНАТОРТЮ. Така організація пояснюється тим, що санаторій не розглядає кожний рядок витрат в рахунку клієнта як щось окреме, і рядки видатків не мають власних ідентифікаторів. Службовці не вводять дані в рядок витрат інакше як у контексті рахунку. Спочатку службовець вводить дані для рахунку № 1546, а потім, в контексті даного рахунку, вводить суми. Буває і так, що службовець бере вже існуючий рахунок і вносить до нього додаткові витрати.

САНАТОРІЙ «ПРИБЕРЕЖНИЙ»			РАХУНОК_ІЗ_САНАТОРІЯ	
Ялта, Україна			ID Номер рахунку	
Номер рахунку:	1546	Дата прибуття:	10/07/2012	Дата прибуття
ПІБ клієнта:	Бондаренко Н.І.		ID ПІБ клієнта	
			Рядок витрат	
			Дата надання послуги 1.1.	
			Опис послуги 1.1.	
			Вартість 1.1.	
10/07/2012	Номер	\$55.00	} 0.N	
10/07/2012	Їжа	\$27.00		
10/07/2012	Телефон	\$2.50		
			Сума до сплати 1.1.	
10/08/2012	Номер	\$55.00		
10/08/2012	Їжа	\$37.60		
10/08/2012	Телефон	\$10.00		
Сума до сплати \$187.1			б	
а				

Рис. 3.31. Приклад композитного об'єкту: а - звіт, оснований на композитному об'єкті; б - композитний об'єкт РАХУНОК_ІЗ_САНАТОРІЯ.

Мінімальне кардинальне число атрибута Рядок витрат дорівнює 0, що означає, що об'єкт РАХУНОК_ІЗ_САНАТОРІЮ може існувати без єдиного рядка витрат. Це дозволяє відкривати рахунок в той момент, коли клієнт тільки вселяється, і до того, як з'являться які-небудь витрати. Якщо б мінімальне кардинальне число дорівнювало 1, то не можна було б відкрити рахунок, перш ніж буде нарахована хоча б одна сума. Рішення про це має прийматися у світлі існуючого ділового регламенту. Політика санаторію може полягати в тому, щоб не відкривати рахунок, поки не будуть нараховані які-небудь витрати. У такому випадку мінімальне кардинальне число атрибута Рядок витрат повинне дорівнювати 1.

Композитний об'єкт може мати більше одного багатозначного атрибута.

На рис. 3.32а показаний рахунок за послуги санаторію, в якому ПІБ клієнта і вартість послуг представлені багатозначними атрибутами. Кожна з цих груп незалежна від іншої. Наприклад, другі дані у групі ПІБ клієнта не пов'язані логічно з другим рядком витрат.

На рис. 3.32б представлена об'єктна діаграма для рахунку, показаного на рис. 3.32а. Атрибут ПІБ клієнта показаний як багатозначний. Він не поміщений в дужку Рядок витрат, оскільки повторення імен не мають нічого спільного з повтореннями послуг. Вони незалежні, як ми тільки що відзначили.

І прості, і групові атрибути можуть бути багатозначними. На рис. 3.32а, наприклад, ПІБ клієнта є простим багатозначним атрибутом. Одного цього достатньо, щоб об'єкт можна було назвати композитним.

САНАТОРІЙ «ПРИБЕРЕЖНИЙ» Ялта, Україна			РАХУНОК_ІЗ_САНАТОРІЯ	
Номер рахунку:	1546	Дата прибуття: 10/07/2012	ID Номер рахунку	}
ПІБ клієнта:	Бондаренко Н.І. Сидоренко А.Ю. Анісова Н.А.		Дата прибуття ID ПІБ клієнта	
			Рядок витрат	} 0.N
10/07/2012	Номер	\$55.00	Дата надання послуги 1.1.	
10/07/2012	Їжа	\$27.00	Опис послуги 1.1.	
10/07/2012	Телефон	\$2.50	Вартість 1.1.	
10/08/2012	Номер	\$55.00		
10/08/2012	Їжа	\$37.60		
10/08/2012	Телефон	\$10.00		
Сума до сплати \$187.1			Сума до сплати 1.1.	
а			б	

Рис. 3.32. Композитний об'єкт з двома групами: а - об'єкт РАХУНОК_ІЗ_САНАТОРІЯ з багатозначним ПІБ клієнта; б - об'єкт РАХУНОК_ІЗ_САНАТОРІЯ з двома багатозначними групами.

Багатозначні атрибути можуть також бути вкладеними. Уявіть собі, наприклад, що потрібно вести облік окремих видів витрат у рамках одного рядка. У формі на рис. 3.33а витрати поділяються на види. Наприклад, їжа ділиться на сніданок, обід і закуски. Об'єктна діаграма з такими вкладеними атрибутами, представлена на рис. 3.33б, показує, що оплата будь-яких послуг може бути розділена на складові.

Для закріплення повторимо, що **композитний об'єкт** - це об'єкт, що має один або кілька простих або групових атрибутів. Об'єктних атрибутів він не містить.

Рис. 3.33. Композитний об'єкт з вкладеними групами: а - об'єкт РАХУНОК_ІЗ_САНАТОРІЯ з різновидами послуг; б - об'єкт РАХУНОК_ІЗ_САНАТОРІЯ з вкладеними багатозначними групами.

Складені об'єкти

Складений об'єкт (*compound object*) має мінімум один об'єктний атрибут.

На рис. 3.34а показані дві різні форми для введення даних. Одна з них використовується студентською поліклінікою. У другу форму вводяться дані про самих студентів. Згідно з цими формами, конкретна картка закріплена не більше ніж за одним студентом, і за конкретним студентом може бути закріплена не більше ніж одна картка.

На рис. 3.34б представлені діаграми для об'єктів СТУДЕНТ і КАРТКА СТУДЕНТА. Об'єкт СТУДЕНТ містить в якості одного з атрибутів об'єкт КАРТКА СТУДЕНТА, а КАРТКА СТУДЕНТА, у свою чергу, містить в якості одного з атрибутів об'єкт СТУДЕНТ. Оскільки і СТУДЕНТ, і КАРТКА СТУДЕНТА мають об'єктні атрибути, обидва вони є складеними об'єктами. Більше того, так як жоден з атрибутів не є багатозначним, зв'язок між об'єктами

СТУДЕНТ і КАРТКА СТУДЕНТА А має вигляд «один до одного» - 1:1.

На рис. 3.34а форми ДАНІ СТУДЕНТА і ДАНІ КАРТКИ СТУДЕНТА включають одна іншу. Тобто у формі ДАНІ КАРТКИ СТУДЕНТА є поле «Закріплена за студентом», а у формі ДАНІ СТУДЕНТА є поле «Закріплена картка». Але це не завжди так: іноді зв'язок може здаватися односпрямованим.

СТУДЕНТ

ID ПІБ студента

ID Номер студ. квитка

Назва факультету

Номер групи

Дата вступу в інститут

Адреса

Телефон

КАРТЧКА СТУДЕНТА

ДАНІ КАРТОЧКИ СТУДЕНТА

Номер карточки

ПІБ лікаря

Номер кабінету

Дата проходження мед. огляду

Захворювання

Закріплена за студентом

а

ДАНІ СТУДЕНТА

Номер студ. Квитка

ПІБ студента

Назва факультету

Номер групи

Дата вступу в інститут

Адреса

Телефон

Закріплена картка

б

КАРТЧКА СТУДЕНТА

ID Номер карточки

ПІБ лікаря

Номер кабінету

Дата проходження проф. огляду

Захворювання

СТУДЕНТ

1.1

Рис. 3.34. Складові об'єкти зі зв'язком виду 1:1 між властивостями : а - приклади форм для вводу даних про карточку і студента; б - складові об'єкти СТУДЕНТ і КАРТЧКА СТУДЕНТА.

Розглянемо звіт і форму на рис. 3.35, в яких описуються два об'єкти: ГУРТОЖИТОК і СТУДЕНТ. Із ЗВІТУ ЗАЙНЯТОСТІ ГУРТОЖИТКУ видно, що в представленні користувачів гуртожиток має атрибути, пов'язані як із самим гуртожитком номер гуртожитку, ПІБ коменданта, телефон, так і з студентам, що проживають в ньому - ПІБ студента, група, номер студента.

З іншого боку, ФОРМА З ДАНИМИ ПРО СТУДЕНТІВ на рис. 3.35а містить інформацію, яка стосується виключно самого студента, - ніяких даних про гуртожиток у цій формі немає.

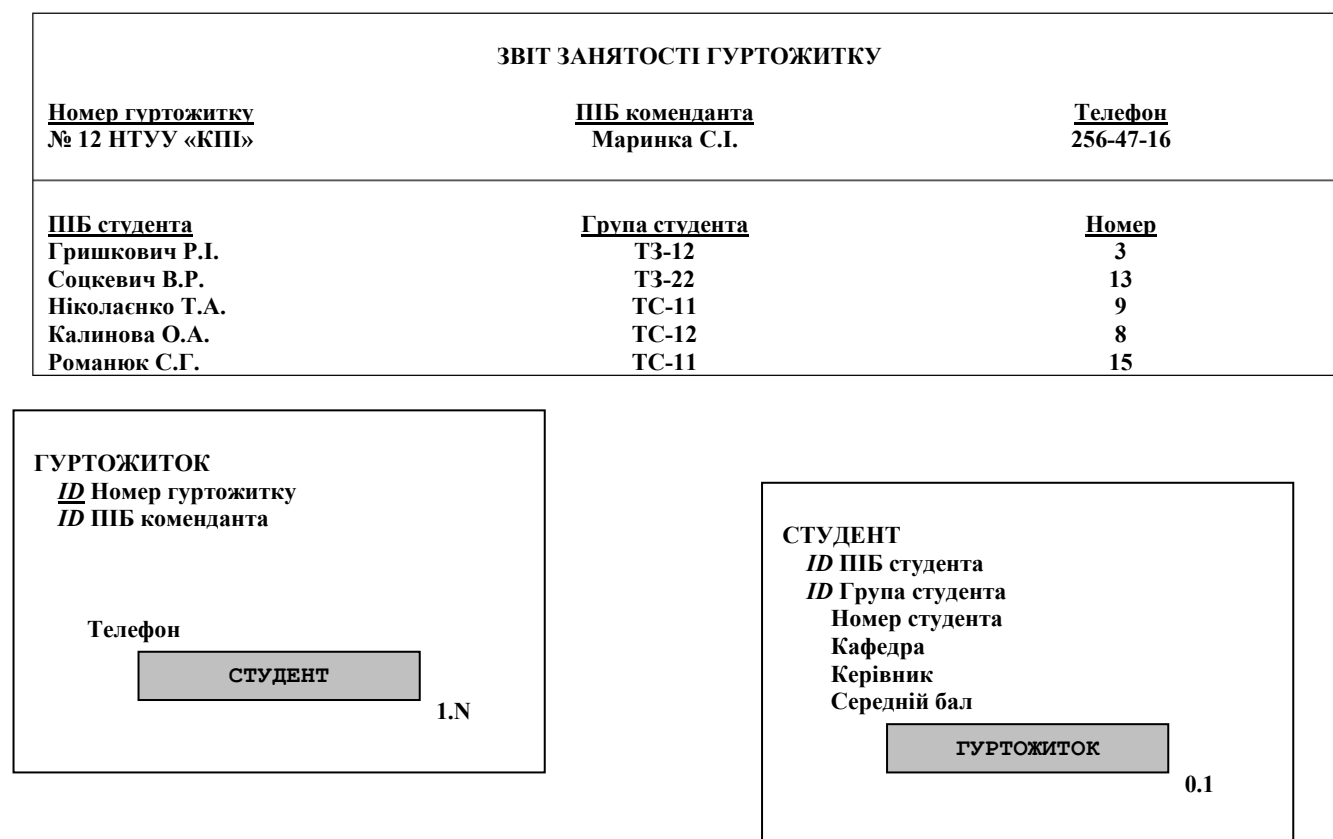


Рис. 3.35. Складові об'єкти зі зв'язком виду 1:K між властивостями - складові об'єкти ГУРТОЖИТОК і СТУДЕНТ.

Як вже зазначалося раніше, *об'єктні атрибути* завжди *з'являються парами*. Навіть якщо форми, звіти та запити демонструють тільки одну сторону зв'язку, у зв'язку завжди є дві сторони. Зв'язок можна порівняти з мостом, з'єднуючим два острови: міст стикається з обома островами, і рухатися по ньому можна в обох напрямках, навіть якщо за звичаєм чи законом цей міст є односпрямованим.

Якщо не вдається знайти ні однієї форми або звіту, який документує одну зі сторін зв'язку, команда розробників повинна запитати користувачів про кардинальні числа цього зв'язку. У цьому випадку команді потрібно з'ясувати, скільки гуртожитків може бути у студента і чи повинен студент бути прикріплений до гуртожитку. Тут ми допустимо, що відповіді на це питання були наступними: студент може бути прикріплений максимум до одного гуртожитку, але може і не бути прикріпленим до жодного. Так, на рис. 3.35б об'єкт ГУРТОЖИТОК містить багатозначний об'єктний атрибут СТУДЕНТ, а об'єкт СТУДЕНТ містить однозначний об'єктний атрибут ГУРТОЖИТОК. Таким чином, зв'язок між об'єктами ГУРТОЖИТОК і СТУДЕНТ має вигляд «один до багатьох», або 1: N.

На рис. 3.36 зображені всі чотири типи складових об'єктів. Взагалі, об'єкт А може містити максимум один або багато об'єктів Б. Аналогічно, об'єкт Б може містити один або багато об'єктів Б.

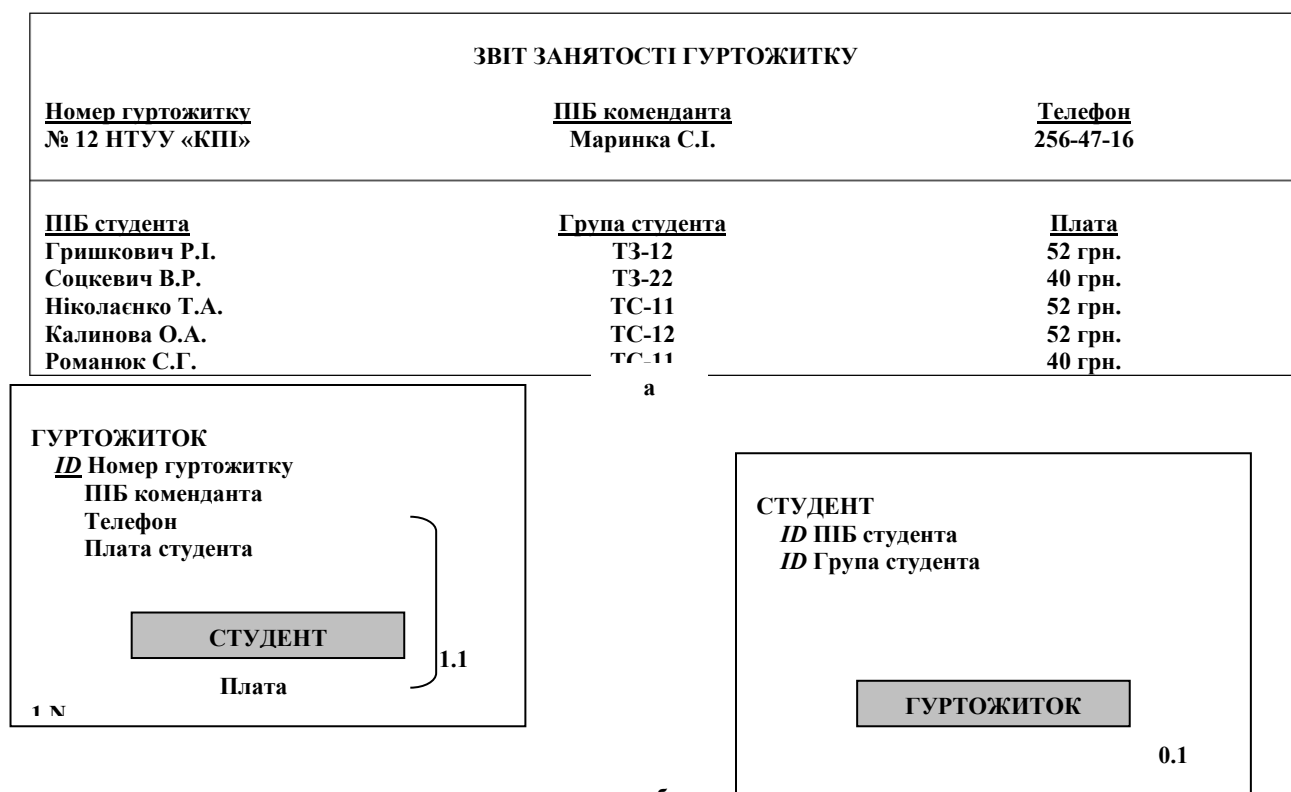
Об'єкт 1 може містити			
Об'єкт 2 може		Один	Багато
	Один	1 : 1	1 : N
	Багато	M : 1	M : N
			містити

Рис. 3.36. Чотири типи складових об'єктів

Гібридні об'єкти

Гібридні об'єкти (*hybrid objects*) - це комбінації композитних і складових об'єктів. Зокрема, **гібридний об'єкт** - це семантичний об'єкт, що має мінімум один багатозначний груповий атрибут, до складу якого входить об'єктний атрибут.

На рис. 3.37а зображена друга версія звіту про зайнятість гуртожитку, представленого на рис. 3.35 а. Різниця полягає в тому, що третій стовпець даних студента містить атрибут Плата замість атрибута Номер студента. Ця різниця є суттєвою, оскільки плата за проживання в гуртожитку є атрибутом не студенти, а гуртожитку, і відноситься до комбінації об'єктів СТУДЕНТ і ГУРТОЖИТОК.



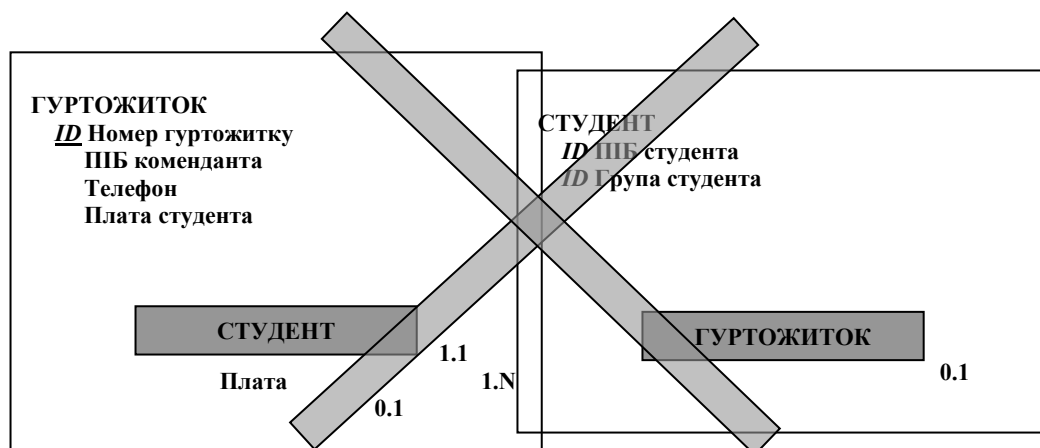


Рис. 3.37. Гібридний об'єкт ГУРТОЖИТОК: а - звіт про гуртожиток з властивістю Плата; б - правильний вид об'єктів ГУРТОЖИТОК і СТУДЕНТ; в - неправильний вид об'єктів ГУРТОЖИТОК і СТУДЕНТ.

На рис 3.37б показана об'єктна діаграма, що моделює цю форму. Об'єкт ГУРТОЖИТОК містить багатозначну групу, що включає в себе об'єктний атрибут СТУДЕНТ і необ'єктний атрибут Плата. Це означає, що атрибути Плата і СТУДЕНТ є спареними в контексті об'єкта.

Розглянемо тепер альтернативний варіант об'єкта ГУРТОЖИТОК, представлений на рис 3.37в. Це неправильна модель звіту, зображеного на рис 3.37а, так як вона показує, що атрибути Плата і СТУДЕНТ є багатозначними незалежно один від одного, що невірно, тому що ці атрибути є багатозначними як пара.

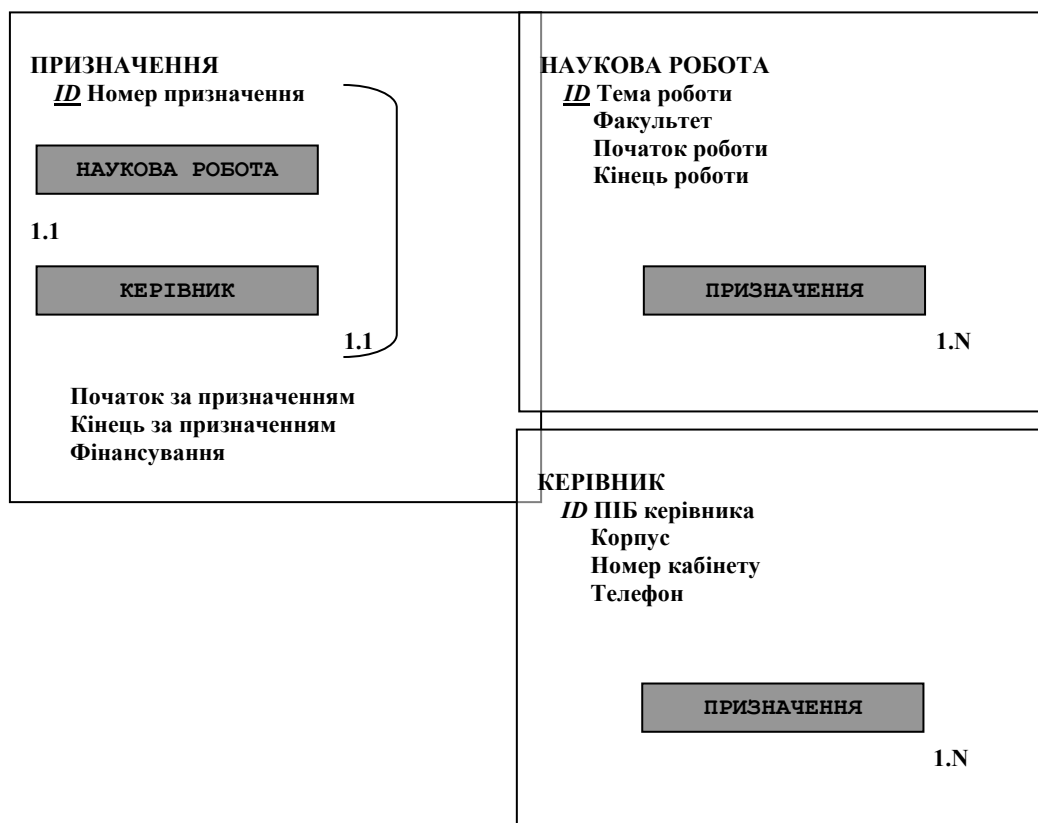
Асоціативні об'єкти

Асоціативний об'єкт (*association object*) - це об'єкт, який пов'язує два або більше об'єкта і містить дані, які стосуються цього зв'язку.

Розглянемо рис. 3.38а, на якому зображений звіт про призначення з наукової роботи. Хоча саме призначення не має явного ідентифікатора, насправді ідентифікатором є комбінація {Тема роботи, ПІБ керівника}. Ці атрибути, однак, належать об'єктам НАУКОВА РОБОТА та КЕРІВНИК відповідно, а не об'єкту ПРИЗНАЧЕННЯ. Ідентифікатором призначення є, таким чином, комбінація ідентифікаторів речей, які призначаються один одному.

ЗВІТ ПРО ПРИЗНАЧЕННЯ ПО НАУКОВІЙ РОБОТІ			
Тема роботи	Ефективність використання інформаційних технологій для мобільних компаній	Керівник наукової роботи Корпус	Карпова О.А. № 7 НТУУ «КПІ»
Факультет	Менеджмент	Номер кабінету	516
Початок роботи	9/09/2011	Телефон	276-45-12
Кінець роботи	9/10/2012		
Початок роботи за призначенням		9/10/2011	
Кінець роботи за призначенням		10/06/2012	
Фінансування		5200 грн.	

а



б

Рис. 3.38. Асоціативний об'єкт ПРИЗНАЧЕННЯ : а - приклад звіту про призначення; б - об'єкт ПРИЗНАЧЕННЯ з семантичним об'єктним ідентифікатором.

На рис. 3.38б показані об'єктні діаграми для цього випадку. І НАУКОВА РОБОТА, і КЕРІВНИК є об'єктними атрибутами об'єкта ПРИЗНАЧЕННЯ, а група {НАУКОВА РОБОТА, КЕРІВНИК} є ідентифікатором об'єкта ПРИЗНАЧЕННЯ. Це означає, що комбінація екземпляра об'єкта НАУКОВА РОБОТА та екземпляра об'єкта КЕРІВНИК ідентифікує конкретне призначення.

Зверніть увагу, що ідентифікатор Номер призначення на рис. 3.38б може бути не унікальним; тим самим вказується, що керівник може бути призначений на одну⁹⁹

і ту ж роботу кілька разів. Якщо це не так, ідентифікатор повинен бути оголошений унікальним. Далі, якщо співробітник (наприклад, той же керівник) може бути призначений на один і той же проект більше одного разу і якщо з якоїсь причини важливо мати унікальний ідентифікатор для призначення, до групи слід додати атрибут дати або який-небудь ще атрибут, який вказує час (тиждень, квартал і т. д.).

Об'єкти виду батько / підтип

Щоб зрозуміти, що являють собою **об'єкти виду батько / підтип** (parent / subtype objects), розглянемо об'єкт СТУДЕНТ на рис. 3.39а. Деякі атрибути об'єкта СТУДЕНТ відносяться до всіх студентів, а деякі - тільки до тих студентів, які є спортсменами. Об'єкт на рис. 3.39а не дуже точно відображає дійсність, оскільки атрибути, властиві спортсменам, не підходять для студентів, які не є спортсменами.

Більш вдала модель показана на рис. 3.39б. Тут об'єкт СТУДЕНТ містить об'єкт підтипу СПОРТСМЕН. Всі атрибути, пов'язані з спортсменом, перенесені в об'єкт СПОРТСМЕН. Студенти, які не є спортсменами, мають один екземпляр об'єкта СТУДЕНТ і жодного екземпляру об'єкта СПОРТСМЕН. Студенти-спортсмени мають по одному екземпляру об'єктів СТУДЕНТ і СПОРТСМЕН. У цьому прикладі об'єкт СТУДЕНТ називається **батьківським об'єктом** (parent object), або **об'єктом надтипу** (supertype object), а об'єкт СПОРТСМЕН називається **об'єктом підтипу** (subtype object).

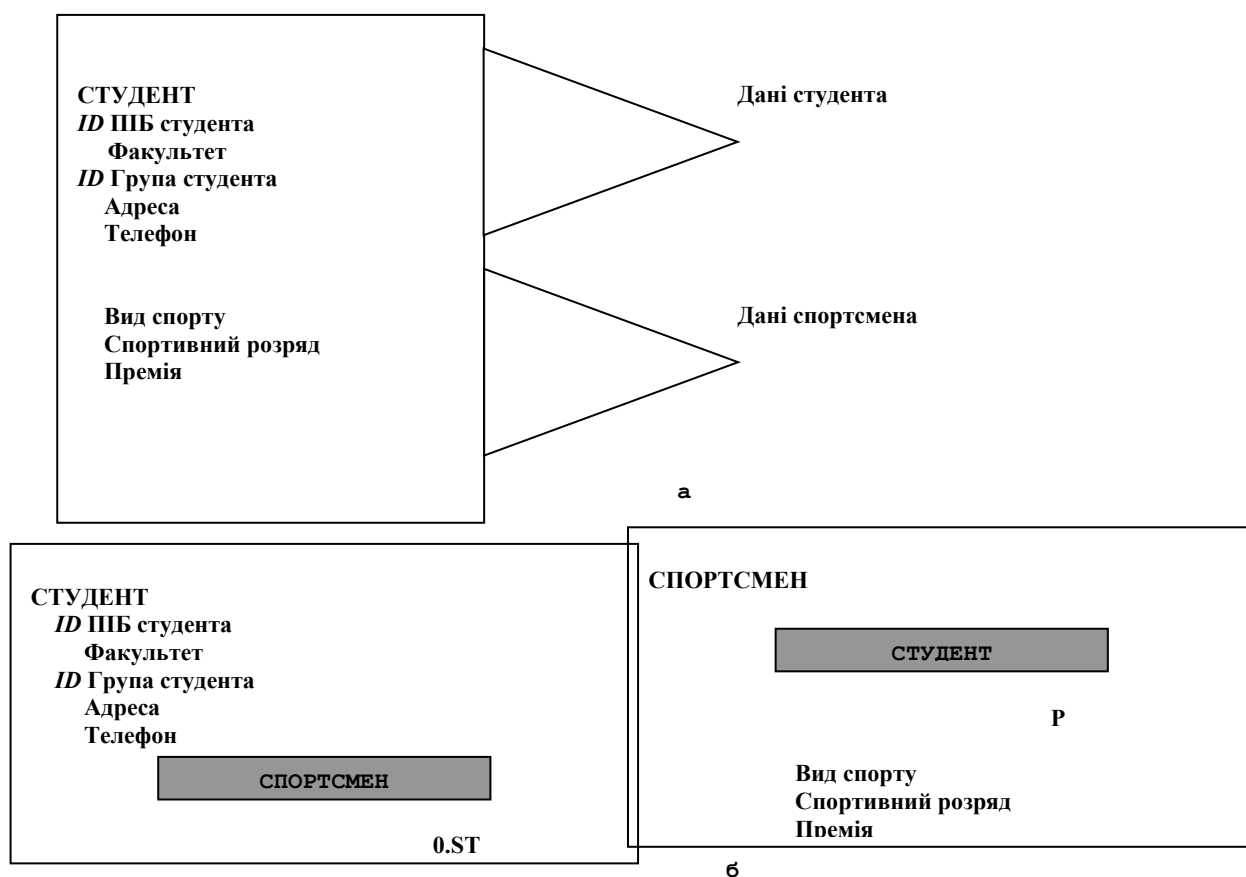


Рис.3.39 Необхідність введення підтипу СПОРТСМЕН : а - об'єкт СТУДЕНТ без підтипу; б - об'єкт СТУДЕНТ з під типом СПОРТСМЕН

Першим атрибутом підтипу є батьківський об'єкт, позначений індексом Р. Вказівка батьківського атрибуту є обов'язковою завжди. Ідентифікатори у підтипу₁₀₀

ті ж, що і у батька. На рис. 3.39 атрибути ПІБ студента і Група студента є ідентифікаторами як об'єкта СТУДЕНТ, так і об'єкта СПОРТСМЕН.

Атрибути підтипу показуються за допомогою індексів 0.ST або 1.ST. Перша цифра (0 або 1) - це мінімальне кардинальне число підтипу. Якщо воно рівне 0, підтип є необов'язковим, а якщо 1, то підтип є обов'язковим. Букви ST (subtype - підтип) вказують на те, що атрибут є підтипом, або атрибутом типу «Є».

Об'єкти виду батько / підтип володіють важливою характеристикою, званої *наслідуванням*. Підтип набуває, або успадковує, всі атрибути свого батька, і тому об'єкт СПОРТСМЕН успадковує всі атрибути об'єкта СТУДЕНТ. До того ж батьків набуває всі атрибути своїх підтипів, і студент, який є спортсменом, набуває всі атрибути спортсмена.

Іноді підтипи виключають один одного. Тобто, наприклад, транспортний засіб може бути легковим автомобілем або вантажівкою, але не тим і іншим одночасно. Студент може бути главою профбюро, спеціалістом програмування, але тільки одним з цих двох типів. Коли підтипи виключають один одного, вони поміщаються в групу підтипів, і групі присвоюється індекс у форматі XYZ X - це мінімальне кардинальне число, рівне 0 або 1, залежно від того, чи є група підтипів обов'язковою, Y і Z вказують кількість атрибутів у групі, яким дозволяється мати значення Y - мінімальна кількість, Z - максимальна.

На рис. 3.40 два типи студентів зображені як група підтипів. Індекс групи 0.1.1 означає, що підтип не потрібен, але якщо він існує, в групі має існувати мінімум один і максимум один підтип (інакше кажучи, рівно один). Зауважте, що кожен з підтипів має індекс О^Т, тобто всі вони є необов'язковими, як і має бути. Якби всі вони були обов'язковими, максимальна кількість атрибутів було б 2, а не 1. Цей запис досить надійний, щоб передбачити ситуації, коли обов'язковими є три з п'яти або сім з десяти підтипів.

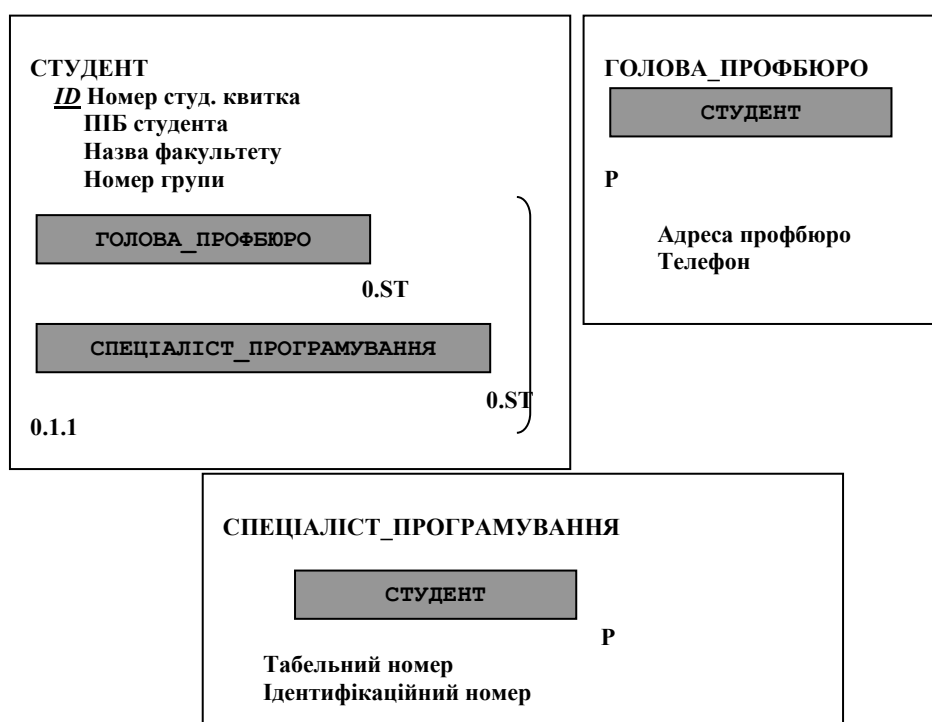


Рис. 3.40. Взаємовиключні підтипи

Об'єкти виду архетип / версія

Останній тип об'єктів - це **об'єкти виду архетип / версія** (archetype / version objects).

Об'єкт архетип (archetype object) - це семантичний об'єкт, який породжує інші семантичні об'єкти, які представляють версії (version objects), випуски або видання архетипу

Наприклад, на рис. 3.41 об'єкт-архетип ПІДРУЧНИК породжує об'єкти - версії ВИДАННЯ. Відповідно до цієї моделі, атрибути Код підручника, Назва, Автор і Видавництво належать об'єкту ПІДРУЧНИК, а атрибути Порядковий номер видання, Дата виходу і Кількість сторінок - об'єкту ВИДАННЯ.

Ідентифікаційна група об'єкта ВИДАННЯ складається з двох частин: ПІДРУЧНИК і Порядковий номер видання. Це типовий зразок ідентифікатора об'єкта-версії. Одна частина ідентифікатора містить об'єкт - архетип, а друга частина - це простий атрибут, що ідентифікує версію архетипу.

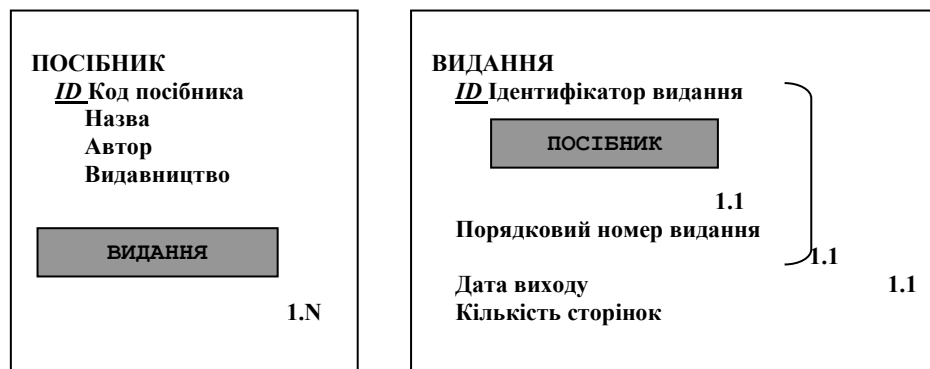


Рис. 3.41. Приклад об'єкта виду архетип / версія

Перехід від семантичної об'єктної моделі до моделі «сутність-зв'язок»

Модель «сутність-зв'язок» і семантична об'єктна модель мають як подібності, так і відмінності. Вони схожі тим, що обидві є інструментами для з'ясування і документування структури даних користувача. Обидві вони мають своєю метою моделювання структури речей у світі користувачів і зв'язків між ними.

Принципова відмінність між двома моделями полягає в орієнтації. Модель «сутність-зв'язок» в якості базової концепції розглядає концепцію сутності. Сутності та їх зв'язки виступають, якщо хочете, як атоми моделі даних. Ці атоми можуть бути організовані в структури, які модель «сутність-зв'язок» називає представленнями користувача (user views). Представлення користувача - це комбінації сутностей, будова яких нагадує будову семантичних об'єктів.

Базове поняття семантичної об'єктної моделі - семантичний об'єкт. Набір семантичних об'єктів в моделі даних - це карта структури речей, які користувачі вважають суттєвими. Ці об'єкти є атомами світу користувачів і являють собою найменші помітні одиниці, якими користувачі бажають оперувати. Вони можуть розбиватися на більш дрібні частини всередині СУБД (або в прикладній програмі), але ці більш дрібні частини не представляють інтересу для користувачів.

З точки зору семантичної об'єктної моделі, сутності, в тому вигляді як вони₁₀₂

визначені в моделі «сутність-зв'язок», не існують. Вони є лише фрагментами, шматками реальних сутностей. Фактично, єдині сутності, які мають сенс для користувачів, - це семантичні об'єкти. По-іншому можна виразити це, сказавши, що семантичні об'єкти є семантично самодостатніми, або семантично завершеними.

Розглянемо приклад. На рис. 3.42а представлені чотири семантичних об'єкту: ПРЕДМЕТ_ВИВЧЕННЯ, СТУДЕНТ, ВИКЛАДАЧ і ТЕМА. Об'єкт ПРЕДМЕТ_ВИВЧЕННЯ містить дані про викладача, який читає даний предмет; студента, який вивчає предмет і тему дисципліни. У той же час на рис. 3.42б зображена модель «сутність-зв'язок» для цих же об'єктів, які виступають тепер в ролі сутностей.

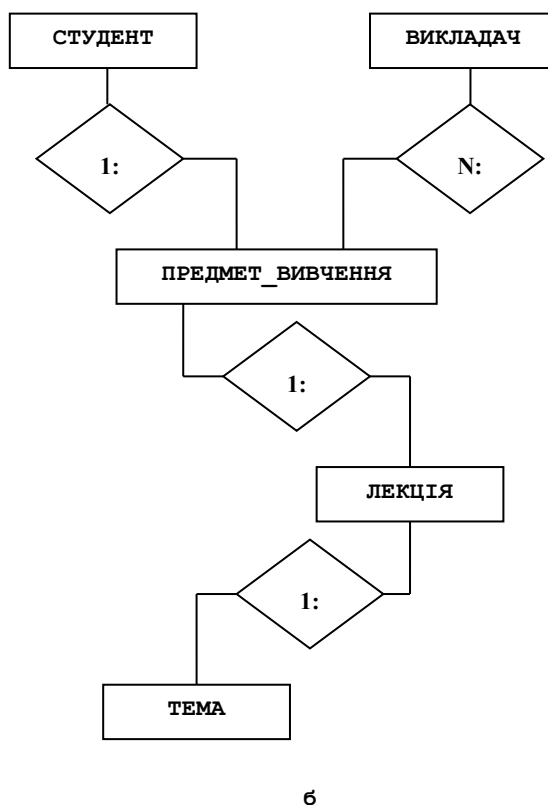
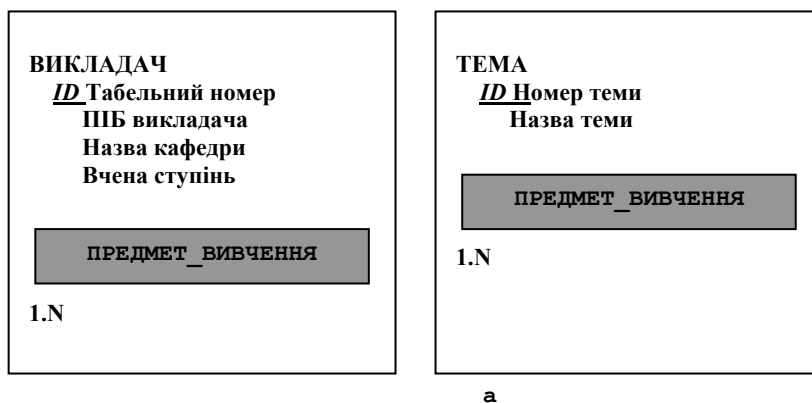


Рис. 3.42. Перехід від семантичної об'єктної моделі до моделі «сутність-зв'язок»:

а - об'єкт ПРЕДМЕТ_ВИВЧЕННЯ та пов'язані з ним семантичні об'єкти; б - модель «сутність-зв'язок» для об'єкта ПРЕДМЕТ_ВИВЧЕННЯ.

З точки зору семантичної об'єктної моделі, в сутностях, як вони представлені

в моделі «сутність-зв'язок», немає необхідності. Семантичні об'єкти можна відразу перетворювати в структуру бази даних, не розглядаючи ER-сутності. Вони *являють собою, так би мовити, недобудовані будинки, споруджені в процесі відходу від парадигми комп'ютерних структур даних до парадигми користувачів.*

Ще одна відмінність полягає в тому, що семантичні об'єкти містять у собі більше метаданих, ніж сутності. У семантичній об'єктній моделі на рис. 3.42a відображено те, що Номер студ. квитка є унікальним ідентифікатором у представленні користувачів. Він може використовуватися або не використовуватися в якості ідентифікатора у відповідній таблиці, але даний факт не істотний для моделі даних. Більше того, семантична об'єктна діаграма вказує на той факт, що існує семантична група атрибутів під назвою Адреса. Ця група містить інші атрибути, складові разом адресу. Існування цієї групи є важливим фактом при розробці форм і звітів. Нарешті, семантична об'єктна діаграма показує, що конкретна ТЕМА може бути розглянута в декількох предметах, але в кожному предметі ця тема може з'явитися тільки один раз. Цей факт неможливо вказати на діаграмі «сутність - зв'язок».

Запитання:

- 3.1. Дайте визначення сутності та наведіть приклад.
- 3.2. Поясніть різницю між класом сутностей і екземпляром сутності.
- 3.3. Дайте визначення атрибута і наведіть приклади, використовуючи сутності на ваш вибір.
- 3.4. Поясніть, що таке композитний атрибут, наведіть приклад.
- 3.5. Який з атрибутів, наведених вами у відповіді на питання 3.3, ідентифікує сутність?
- 3.6. Дайте визначення зв'язку та наведіть приклад.
- 3.7. Поясніть, у чому різниця між класом зв'язків та екземпляром зв'язку.
- 3.8. Дайте визначення степеня зв'язку. Наведіть приклад зв'язку зі степенем більше 2, відмінний від того, який дано в тексті.
- 3.9. Перерахуйте три типи бінарних зв'язків та наведіть приклади. Намалюйте ER-діаграму для кожного типу.
- 3.10. Дайте визначення термінів максимальне кардинальне число і мінімальне кардинальне число, максимальна кардинальність і мінімальна кардинальність. Знайдіть на рис. 3.12 приклади зв'язків з максимальної кардинальністю 1:1, 1:N і N:M.
- 3.11. Назвіть і намалюйте символи, використовувані в діаграмах «сутність зв'язок» для зображення: (а) сутності; (б) зв'язку; (в) слабкою сутності та зв'язку; (г) рекурсивної зв'язку; (д) сутності підтипу.
- 3.12. Наведіть приклад ER-діаграми для сутностей ФАКУЛЬТЕТ і ДОЦЕНТ, що мають зв'язок 1: N. Зробіть припущення, що на факультеті може і не бути доцентів, але кожен доцент повинен належати до якого-небудь факультету.
- 3.13. Наведіть приклад рекурсивного зв'язку та покажіть її на ER-діаграмі.
- 3.14. Наведіть приклади атрибутів для сутностей ФАКУЛЬТЕТ і ДОЦЕНТ з відповіді на питання 3.12 і покажіть їх на ER-діаграмі. Використовуйте для цього символи UML-стилю.
- 3.15. Дайте визначення терміна слабка сутність і наведіть приклад, відмінний від того, який дано в тексті.

3.16. Поясніть, в чому полягає неоднозначність у визначенні терміна слабка сутність. Наведіть приклади, відмінні від тих, які дані в тексті.

3.17. Дайте визначення терміну ідентифікаційно-залежна сутність і наведіть свій приклад.

3.18. Продемонструйте використання слабкої сутності для представлення багатозначного складеного атрибуту Телефон, що складається з однозначних атрибутів Код регіону та Номер телефону. Нехай при цьому атрибут Телефон належить сутності СТУДЕНТ. Вкажіть мінімальне і максимальне кардинальне число на обох сторонах зв'язку. Використовуйте символи UML-стилю.

3.19. Опишіть, що таке підтипи сутностей, і наведіть приклад.

3.20. Поясніть значення терміна наслідування і покажіть, як це стосується вашої відповіді на питання 3.19.

3.21. Поясніть різницю між зв'язком типу «МАС» і зв'язком типу «Є» і наведіть приклад для кожного зв'язку.

3.22. Поясніть, чому важливо перевіряти модель даних після її створення.

3.23. Поясніть, чому модель «сутність-зв'язок» і семантична об'єктна модель схожі на лінзи.

3.24. Дайте визначення семантичного об'єкта.

3.25. Поясніть різницю між ім'ям класу об'єкта і ім'ям екземпляра об'єкта. Наведіть приклади для обох випадків.

3.26. Що потрібно для того, щоб набір атрибутів був достатнім описом?

3.27. Поясніть, що розуміється під словосполученням окремий феномен у визначенні семантичного об'єкта.

3.28. Назвіть три типи атрибутів.

3.29. Наведіть приклади атрибутів наступних типів:

- простий однозначний атрибут;
- груповий однозначний атрибут;
- простий багатозначний атрибут;
- груповий багатозначний атрибут;
- простий об'єктний атрибут;
- багатозначний об'єктний атрибут.

3.30. Що таке мінімальне кардинальне число? Як воно використовується? Які типи атрибутів мають мінімальне кардинальне число?

3.31. Що таке максимальне кардинальне число? Як воно використовується? Які типи атрибутів мають максимальне кардинальне число?

3.32. Що таке парні атрибути? Для чого вони потрібні?

3.33. Що таке ідентифікатор об'єкта? Наведіть приклад простого і групового атрибута, що ідентифікує об'єкт.

3.34. Дайте визначення домену атрибута. Які бувають типи доменів атрибутів? Чому необхідний семантичний опис?

3.35. Що таке представлення семантичного об'єкта? Наведіть приклад об'єкта і двох його представлень, відмінний від того, який дано в тексті.

3.36. Наведіть приклад простого об'єкта, відмінний від того, який дано в цій лекції.

3.37. Наведіть три приклади композитних об'єктів, відмінні від тих, які наведені в цій лекції. У одного об'єкта повинен бути рівно один простий105

багатозначний атрибут, інший об'єкт повинен мати дві незалежні багатозначні групи, а третій об'єкт повинен містити вкладені багатозначні групи.

3.38. Наведіть приклади чотирьох наборів складових об'єктів, відмінні від тих, які дані в тексті. Один набір повинен мати зв'язок 1:1, інший - 1: N, третій - M: 1, і четвертий - M: N.

3.39. Наведіть приклад гібридного об'єкта, відмінний від того, який дано в цій лекції.

3.40. Наведіть приклад одного асоційованого та двох складових об'єктів.

3.41. Наведіть приклад об'єкта надтипу з трьома об'єктами підтипів.

3.42. Вкажіть подібності моделі «сутність-зв'язок» і семантичної об'єктної моделі.

3.43. Вкажіть основні відмінності між моделлю «сутність-зв'язок» і семантичною об'єктною моделлю.

Вправи:

3.1. Зобразіть ER-діаграму для студента, який займається спортивною гімнастикою. Студент відвідує як групові, так і індивідуальні заняття. Інструктор може бути постійним і непостійним. Врахуйте, що студент приймає участі у змаганнях.

3.2. Модифікуйте ER-діаграму попередньої вправи, включивши в неї сутність Заняття. Нехай сутності ЩДІВІЩУАЛЬНОЕ_ЗАНЯТТЯ і ГРУПОВЕ_ЗАНЯТТЯ будуть підтипами сутності ЗАНЯТТЯ. Змініть зв'язки, де це необхідно. Використовуйте традиційні символи.

3.3. Модифікуйте ER-діаграму вправи 3.1, виключивши з неї сутність ІНСТРУКТОР. Змініть зв'язки, де це необхідно. Використовуйте символи UML-стилю.

3.4. Якій з моделей вашої відповіді на вправи 3.2 та 3.3 ви віддасте перевагу? Поясніть причину вашого уподобання.

3.5. Модифікуйте семантичну об'єктну діаграму на рис. 3.29, ввівши в неї об'єкти ГРУПА, ПРЕДМЕТИ_НА_ВИБІР і ЗАПИС. Передбачається, що ЗАПИС - це асоціативний об'єкт, що зв'язує студента з предметом.

Модифікуйте семантичну об'єктну діаграму на рис.

3.29, ввівши в неї об'єкт КОМІТЕТ. Припустимо, що один викладач може бути призначений у кілька комітетів і що комітет включає в себе багато викладачів.

Створіть об'єкт ЗАСІДАННЯ - об'єкт виду архетип / версія, що представляє засідання комітету.

ТЕМА 4

Нормалізація

Нормалізація - це процес перетворення відношення, що має деякі недоліки, в відношення, яке цих недоліків не має.

Нормалізацію можна використовувати як критерій для визначення коректності, цілісності і правильності відношень.

Питання про те, що таке добре структуроване відношення, було предметом численних теоретичних досліджень. Термін Нормалізація зобов'язаний своєю появою одному з піонерів технології баз даних, Е. Ф. Кодду (E. F. Codd), який визначив різні нормальні форми (normal forms) відношень. В цьому розділі обговорюється нормалізація, включаючи результати теорем, важливих і корисних для розробників баз даних. Докази цих теорем та формальне, більш ретельне дослідження даного питання можна знайти в роботі Дейта і Ульмана (C.J. Date і J. D. Ullman).

На жаль, не всі відношення при первинному проектуванні інженером по знаннях є цілісними. Таблиця, що відповідає мінімальному визначенню відношення, може мати неефективну або невідповідну структуру. Для деяких відношень зміна даних може привести до небажаних наслідків, званих **аномаліями модифікації** (*modification anomalies*). Аномалії можуть бути усунені шляхом розбиття вихідного відношення на два або більше нових відношень. У більшості випадків тільки перевизначені, або нормалізовані, відношення гарантують цілісність даних.

4.1 Класи відношень

Відношення можна класифікувати за типами аномалій модифікації, яким вони піддаються. У 1970-х роках теоретики реляційних баз даних поступово скорочували кількість цих типів. Хтось знаходив аномалію, класифікував її і думав, як запобігти її виникненню. Кожен раз, коли це відбувалося, критерії побудови відношень удосконалювалися.

*Класи відношень і способи запобігання аномалій називаються **нормальними формами** (normal forms).*

Залежно від своєї структури, відношення може бути в першій, в другій або в будь-якій іншій нормальній формі.

У своїй роботі, що послідувала за епохальною статтею 1970 р., Кодд і інші визначили **першу, другу і третю нормальні форми** (1НФ, 2НФ і 3НФ). Пізніше була введена **нормальна форма Бойса-Кодда** (НФБК), а потім були визначені четверта і п'ята нормальні форми. Відношення в другій нормальній формі є також ставленням в першій нормальній формі, а відношення в 5НФ (п'ята нормальна форма) знаходиться одночасно в 4НФ, НФБК, 3НФ, 2НФ і 1НФ.

Ці нормальні форми допомагали, але у них було і серйозне обмеження. Не було теорії, яка гарантує, що будь-яка з цих форм усуне всі аномалії: кожна форма могла усунути тільки певні їх види. Ця ситуація вирішилася в 1981 р, коли Р. Фагін (R. Fagin) ввів нову нормальну форму, яку він назвав **доменно-ключовою нормальною формою**, або ДКНФ (domain / key normal form, DK / NF). У своїй¹⁰⁷

важливій статті Фагін показав, що відношення в ДКНФ вільне від всіх аномалій модифікації, незалежно від їх типу! Він також показав, що будь-яке відношення, вільне від аномалій модифікації, має перебувати в ДКНФ.

До введення ДКНФ теоретикам реляційних баз даних доводилося продовжувати пошук все нових і нових аномалій і нормальних форм. Доказ Фагіна спростив ситуацію. Якщо можна привести відношення до ДКНФ, то можна бути впевненим, що в ньому не буде аномалій модифікації. Вся проблема в тому, як привести відношення до ДКНФ.

Класифікація нормальних форм:

- перша нормальна форма (1 НФ);
- друга нормальна форма (2 НФ);
- третя нормальна форма (3 НФ);
- нормальна форма Бойса-Кодда (НФБК);
- четверта нормальна форма (4 НФ);
- п'ята нормальна форма (5 НФ);
- доменно-ключова нормальна форма (ДКНФ).

4.2 Нормальні форми від першої до п'ятої

Перша Нормальна Форма (1 НФ)

Перша Нормальна Форма (1 НФ) - це звичайне відношення, на перетині рядків і стовпців якого розташоване атомарне значення відповідного атрибута, яке має унікальне значення ключового атрибута, що не допускає дублювання кортежів.

Таке відношення автоматично вже знаходиться в 1 НФ. Нагадаємо коротко властивості відношень (це і будуть властивості 1 НФ):

- Відношення не має однакових кортежів.
- Кортежі не впорядковані.
- Атрибути не впорядковані і розрізняються по найменуванню.
- Всі значення атрибутів атомарні.

Розглянемо відношення `СТУДЕНТ_КАФЕДРА_СЕКЦІЯ` (рис. 4.4), яке описує студентський спортивний клуб. Відношення містить атрибути Код кафедри студента, Номер заліковки, ПІБ студента, Телефон кафедри, Секція і Плата. В якості потенційного ключа будемо використовувати Номер заліковки студента і Секцію, яку він відвідує.

Рис. 4.4. Відношення СТУДЕНТ_КАФЕДРА_СЕКЦІЯ.

Код кафедри	Номер заліковки	ПІБ студента	Телефон кафедри	Секція	Плата
101	T33102	Богданенко О. Н.	431-45-78	Аеробіка	75
101	T33208	Лисенко Т. В.	431-45-78	Шейпінг	70
102	TM3107	Іванов Н. І.	452-94-74	Боротьба	60
102	TM3107	Іванов Н. І.	452-94-74	Футбол	65
103	TI3112	Сидоров К. А.	428-03-29	Боротьба	60

Алгоритм нормалізації (приведення до 1НФ)

Крок 1. Задаємо одне або кілька відношень, що відображають поняття предметної області.

Крок 2. Випишуємо виявлені функціональні залежності по моделі предметної області (виключаємо дублювання інформації, а саме значень ключових атрибутів, кортежів і т.п., формуємо унікальний опис об'єктів предметної області). Якщо на перетині рядка і стовпця кожен елемент є унікальним, то всі відношення автоматично знаходяться в 1НФ.

Аномалії модифікації

Навіть одного погляду на відношення СТУДЕНТ_КАФЕДРА_СЕКЦІЯ досить, щоб побачити, що дані зберігаються в ній з *великою надмірністю*. Є повторення прізвища студента, повторення коду і телефону кафедри, а також назви секції. Крім того, в даному відношенні зберігаються разом незалежні один від одного дані - дані про студентів, кафедрах і секціях. Поки ніяких дій з відношенням не проводиться, це не страшно. Але як тільки стан предметної області змінюється, то, при спробах відповідним чином змінити стан бази даних, виникає велика кількість проблем.

Історично ці проблеми отримали назву **аномалії оновлення**. Аномалії визначаються як протиріччя між моделлю предметної області і фізичною моделлю даних, підтримуваних засобами конкретної СУБД. Іншими словами, **аномалії** - *це неадекватність моделі даних предметної області, або деякі додаткові труднощі в реалізації обмежень предметної області засобами СУБД*.

Таким чином, будемо дотримуватися інтуїтивного поняття аномалії як неадекватності моделі даних предметної області, (що говорить насправді про те,109

що логічна модель даних просто складна і не до кінця нормалізована!) Або необхідності додаткових зусиль для реалізації всіх обмежень визначених у предметної області (додатковий програмний код у вигляді тригерів або збережених процедур).

Оскільки аномалії виявляють себе при виконанні операцій, що змінюють стан бази даних, то розрізняють наступні види аномалій:

- Аномалії вставки (INSERT).
- Аномалії оновлення (UPDATE).
- Аномалії видалення (DELETE).

Використовуючи відношення СЕКЦІЯ, наведемо приклад різних аномалій.

Аномалії вставки (INSERT)

У відношення СТУДЕНТ_КАФЕДРА_СЕКЦІЯ не можна вставити дані про студента, який поки не відвідує ніяку з перерахованих секцій. Дійсно, якщо, наприклад, студент Пушніков вирішив вступити в спортивний клуб, але ще не визначив, яку секцію буде відвідувати, то в дане відношення необхідно вставити наступний кортеж {103, ТІЗІ10, Пушніков А.І., 428-03-29, null , null}. Це зробити неможливо, тому що атрибут Секція входить до складу потенційного ключа, і, отже, не може містити null-значень.

Причина аномалії: зберігання в одному відношенні різномірної інформації.

Висновок: логічна модель даних неадекватна моделі предметної області. База даних, заснована на такій моделі, працюватиме неправильно.

Аномалії оновлення (UPDATE)

В відношенні присутні повторення в деяких кортежах прізвища студента, коду і телефону кафедри і назви секції. Тому якщо студент змінює прізвище, або секцію, або кафедра змінює телефон, то такі зміни необхідно *одночасно* виконати у всіх місцях, де це прізвище, секція зустрічаються, інакше відношення стане некоректним (наприклад, прізвище одного і того ж студента в різних кортежі буде різною) . Таким чином, оновлення бази даних однією дією реалізувати неможливо. Для підтримки відношення в цілісному стані необхідно написати тригер, який при оновленні запису коректно виправляв би дані і в інших місцях.

Причина аномалії: надмірність даних, також породжена тим, що в одному відношенні зберігається різномірною інформація.

Висновок: збільшується складність розробки бази даних. База даних, заснована на такій моделі, працюватиме правильно тільки при наявності додаткового програмного коду у вигляді тригерів.

Аномалії видалення (DELETE)

При видаленні деяких даних може відбутися втрата іншої інформації. Наприклад, якщо закрити секцію аеробіки і видалити всі рядки, в яких вона зустрічається, то будуть втрачені всі дані про студентку Богданенко.

Причина аномалії: зберігання в одному відношенні різномірної інформації (і про студентів, і про секціях).

Висновок: логічна модель даних неадекватна моделі предметної області. База даних, заснована на такій моделі, працюватиме неправильно.

Функціональні залежності

Ставлення `СТУДЕНТ_КАФЕДРА_СЕКЦІЯ` знаходиться в 1НФ, при цьому, як було показано вище, логічна модель даних неадекватна моделі предметної області. Таким чином, першої нормальної форми *недостатньо* для правильного моделювання даних.

Визначення функціональної залежності

Для усунення зазначених аномалій (а насправді для *правильного проектування моделі даних!*) застосовується метод нормалізації відношень. Нормалізація заснована на понятті функціональної залежності атрибутів відношень

Нехай R – відношення. Множина атрибутів Y **функціонально залежно** від множини атрибутів X (X *функціонально визначає* Y) тоді і тільки тоді, коли для *будь-якого стану* відношення R для будь-яких кортежів $r_1, r_2 \in R$ з того, що $r_1X = r_2X$ випливає, що $r_1Y = r_2Y$ (тобто у всіх кортежах, що мають однакові значення атрибутів X , значення атрибутів Y також співпадають *в будь-якому стані* відношення R). Символічно функціональна залежність записується $X \rightarrow Y$. Множина атрибутів X називається **детермінантом функціональної залежності**, а множина атрибутів Y називається **залежною частиною**. Зауваження. Якщо атрибути X складають потенційний ключ відношень R , то *будь-який* атрибут відношення R функціонально залежить від X .

В відношенні `СТУДЕНТ_КАФЕДРА_СЕКЦІЯ` можна навести такі приклади функціональної залежності.

Залежність атрибутів від ключа відношення:

{Номер заліковки, Секція} $\rightarrow$ Код кафедри
{Номер заліковки, Секція} $\rightarrow$ ПБІ студента
{Номер заліковки, Секція} $\rightarrow$ Телефон кафедри
{Номер заліковки, Секція} $\rightarrow$ Плата

Залежність атрибутів, що характеризують студента від Номера заліковки студента:

Номер заліковки $\rightarrow$ Код кафедри
Номер заліковки $\rightarrow$ ПБІ студента
Номер заліковки $\rightarrow$ Телефон кафедри

Залежність вартості від назви секції:

Секція $\rightarrow$ Плата

Зауваження. Наведені функціональні залежності *не виведені* з зовнішнього вигляду відношень, наведеного на рис. 5.5. Ці залежності відображають взаємозв'язки, виявлені між об'єктами предметної області і є додатковими обмеженнями, обумовленими предметною областю. Таким чином, функціональна¹¹

залежність - *семантичне поняття*. Вона виникає, коли за значеннями одних даних в предметній області можна визначити значення інших даних. Наприклад, знаючи секцію можна визначити її вартість або знаючи номер заліковки можна визначити ПІБ студента і дані про кафедру студента. Функціональна залежність *задає додаткові обмеження* на дані, які можуть зберігатися в відношеннях. Для коректності бази даних (адекватності предметної області) необхідно при виконанні операцій модифікації бази даних перевіряти всі обмеження, визначені функціональними залежностями.

Функціональні залежності відношень і математичне поняття функціональної залежності

Функціональна залежність атрибутів відношень нагадує поняття функціональної залежності в математиці. *Але це не одне і те ж*. Для порівняння нагадаємо математичне поняття функціональної залежності:

Функціональна залежність (функція) - це *трийка об'єктів* (X, Y, f) , *где* X – множина (**область визначення**),

Y – множина (**множина значень**),

f – правило згідно з яким кожному елементу $x \in X$ ставиться у відповідність один і тільки один елемент $y \in Y$ (**правило функціональної залежності**).

Функціональна залежність зазвичай позначається як $f: X \rightarrow Y$ або $y = f(x)$.

Зауваження. Правило f може бути задано будь-яким способом - у вигляді формули (найчастіше), за допомогою таблиці значень, за допомогою графіка, текстовим описом і т.д.

Функціональна залежність атрибутів відношення теж *нагадує* це визначення. Дійсно:

- В якості області визначення виступає домен, на якому визначено атрибут X (або декартів добуток множин доменів, якщо X є множиною атрибутів)
- В якості значень виступає домен, на якому визначено атрибут Y (або декартів добуток доменів)
- Правило f реалізується наступним алгоритмом:
 - а) за даним значенням атрибута X знайти *будь-який* кортеж відношення, що містить це значення;
 - б) значення атрибута Y в цьому кортежі і буде значенням функціональної залежності, відповідним даному X .

Визначення функціональної залежності в відношенні гарантує, що знайдене значення Y *не залежить від вибору кортежу*, тому правило f визначено коректно.

Відмінність від математичного поняття відношення полягає в тому, що, якщо розглядати математичне поняття функції, то для *фіксованого значення* $x \in X$ відповідне значення функції $y = f(x)$ завжди одне й те саме. Наприклад, якщо задана функція $y = x^2$, то для значення $x = 2$ відповідне значення y *завжди* буде дорівнювати 4. На противагу цьому в відношеннях значення залежного атрибута може приймати *різні* значення в різних станах бази даних.

Наприклад, атрибут ПІБ студента функціонально залежить від атрибута Номер заліковки. Припустимо, що зараз студент з номером залікової книжки ТІЗ112 має прізвище Сидоров, тобто при значенні детермінанта рівного ТІЗ11, значення залежного аргументу рівне "Сидоров". Але студент може змінити прізвище, наприклад на "Петров". Тепер *при тому ж* значенні детермінанта, рівного ТІЗ11, значення залежного аргументу рівне "Петров".

Таким чином, поняття функціональної залежності атрибутів можна вважати повністю еквівалентним математичному поняттю функціональної залежності, тому що значення цієї залежності різні при різних станах відношення, і, найголовніше, ці значення можуть змінюватися *непередбачувано*.

Функціональна залежність атрибутів стверджує лише те, що для кожного конкретного стану бази даних за значенням одного атрибута (детермінанта) можна однозначно визначити значення іншого атрибута (залежної частини). Але конкретні значення залежної частини можуть бути різні в різних станах бази даних.

Друга Нормальна Форма (2НФ)

Відношення R знаходиться в другій нормальній формі (2НФ) тоді і тільки тоді, коли відношення знаходиться в 1НФ і немає неключових атрибутів, залежних від частини складного ключа. (Неключовий атрибут - це атрибут, який не входить до складу жодного потенційного ключа).

Зауваження. Якщо потенційний ключ відношення є простим, то відношення автоматично перебуває у 2НФ.

Відношення СТУДЕНТ_КАФЕДРА_СЕКЦІЯ не перебуває у 2НФ, тому що є атрибути, які залежать від частини складного ключа:

Залежність атрибутів, що характеризують студента від Номеру заліковки студента, вартість від назви секції, є залежністю від частини складного ключа:

Номер заліковки → Код кафедри

Номер заліковки → ПІБ студента

Номер заліковки → Телефон кафедри

Секція → Плата

Для того щоб усунути залежність атрибутів від частини складного ключа, потрібно зробити декомпозицію відношень на декілька відношень. При цьому ті атрибути, які залежать від частини складного ключа, виносяться в окреме відношення.

Відношення **СТУДЕНТ_КАФЕДРА_СЕКЦІЯ** декомпозиємо на два відношення - **СТУДЕНТ_КАФЕДРА** і **СЕКЦІЯ**.

Відношення **СТУДЕНТ_КАФЕДРА** (Код кафедри, Номер заліковки, ПІБ студента, телефон кафедри).

Функціональні залежності:

Залежність атрибутів, що характеризують студента від Номеру заліковки студента:

Номер заліковки → Код кафедри

Номер заліковки → ПІБ студента

Номер заліковки → Телефон кафедри

Відношення СЕКЦІЯ (Секція, Плата)

Функціональні залежності:

Залежність вартості від назви секції:

Секція → Плата

Код кафедри	Номер заліковки	ПІБ студента	Телефон кафедри	Секція
101	T33102	Богданенко О. Н.	431-45-78	Аеробіка
101	T33208	Лисенко Т. В.	431-45-78	Шейпінг
102	TM3107	Іванов Н. І.	452-94-74	Боротьба
102	TM3107	Іванов Н. І.	452-94-74	Футбол
103	TI3112	Сидоров К. А.	428-03-29	Боротьба

а

Секція	Плата
Аеробіка	75
Шейпінг	70

Боротьба	60
Футбол	65

б

Рис. 4.5. Декомпозиція відношення СТУДЕНТ_КАФЕДРА_СЕКЦІЯ:
а - відношення СТУДЕНТ_КАФЕДРА, б - відношення СЕКЦІЯ

Аналіз декомпозованих відношень.

Відношення, отримані в результаті декомпозиції, знаходяться в 2НФ. Дійсно, відношення СТУДЕНТ_КАФЕДРА і СЕКЦІЇ мають прості ключі, отже автоматично знаходяться в 2НФ.

Частина аномалій оновлення усунена. Так, дані про студентів і секціях тепер зберігаються в різних відношеннях, тому при вступі нового студента в спортивний клуб, але ще не визначився яку секцію відвідувати, просто додаються кортежі в відношення СТУДЕНТ_КАФЕДРА. Тепер прізвища студентів і назви секцій зберігаються без надмірності.

Проте, частина аномалій вирішити не вдалося.

Решта аномалії вставки (INSERT)

У відношення СТУДЕНТ_КАФЕДРА не можна вставити кортеж {103, ТІЗ110, Пушніков А.І., 428-04-37}, тому що при цьому вийде, що два співробітники однієї кафедри під номером 103 (Сидоров К.А. і Пушніков А.І.) мають різні номери телефонів кафедри, а це суперечить моделі предметної області. У цій ситуації можна запропонувати два рішення, в залежності від того, що реально відбулося в предметній області. Інший номер телефону може бути введений з двох причин - за помилкою людини, що вводить дані про нового студента, або тому що номер телефону кафедри дійсно змінився. Тоді можна написати тригер, який при вставці запису про студента перевіряє, чи збігається телефон кафедри з уже наявними телефоном у іншого студента цього ж відділу. Якщо номери відрізняються, то система повинна задати питання, чи залишити старий номер кафедри або замінити його новим. Якщо потрібно залишити старий номер (новий номер введений помилково), то кортеж з даними про нового студента буде вставлений, але номер телефону у нього буде той, який вже є на кафедрі (в даному випадку, 428-03-29). Якщо ж номер кафедри дійсно змінився, то кортеж буде вставлений з новим номером, і одночасно будуть змінені номери телефонів у всіх студентів цієї кафедри. І в тому і в іншому випадку не обійтися без розробки громіздкого тригера.

Причина аномалії: надмірність даних, породжена тим, що в одному відношенні зберігається різнорідна інформація (про студентів і про секції).

Висновок: збільшується складність розробки бази даних. База даних, заснована на такій моделі, працюватиме правильно тільки при наявності додаткового програмного коду у вигляді тригерів.

Решта аномалії оновлення (UPDATE)

Одні і ті ж номери телефонів кафедри повторюються в деяких кортежах відношення. Тому якщо на кафедрі змінюється номер телефону, то такі зміни необхідно одночасно виконати у всіх місцях, де цей номер телефону зустрічається, інакше відношення стане некоректним. Таким чином, оновлення бази даних однією дією реалізувати неможливо. Необхідно написати тригер, який при оновленні запису коректно виправляє номери телефонів в інших місцях.

Причина аномалії: надмірність даних, також породжена тим, що в одному відношенні зберігається різнорідна інформація.

Висновок: збільшується складність розробки бази даних. База даних, заснована на такій моделі, працюватиме правильно тільки при наявності додаткового програмного коду у вигляді тригерів.

Решта аномалії видалення (DELETE)

При видаленні деяких даних як і раніше може відбутися втрата іншої інформації. Наприклад, якщо видалити співробітника Сидорова К.А., то буде втрачена інформація про те, що кафедра з кодом 103 має наступний номер телефону: 428-03-29.

Причина аномалії: зберігання в одному відношенні різнорідної інформації (і про студентів, і про кафедрах).

Висновок: логічна модель даних неадекватна моделі предметної області. База даних, заснована на такій моделі, працюватиме неправильно.

Зауважимо, що при переході до другої нормальної форми відношення стали майже адекватними предметної області. Залишилися також труднощі в розробці бази даних, пов'язані з необхідністю написання тригерів, що підтримують цілісність бази даних. Ці труднощі тепер пов'язані тільки з одним відношенням СТУДЕНТ_КАФЕДРА.

Алгоритм переходу від 1 НФ до 2 НФ

Відношення знаходяться в 1НФ.

Крок 1. Виявляємо у відношеннях залежність атрибутів від частини складного ключа.

Крок 2. Проводимо декомпозицію цих відношень на кілька відношень в такий спосіб: ті атрибути, які залежать від частини складного ключа виносяться в окреме відношення разом з цією частиною ключа. У вихідному відношенні залишаються всі ключові атрибути:

Вихідне відношення: $R(K_1, K_2, A_1, \dots, A_n, B_1, \dots, B_m)$.

Ключ: $\{K_1, K_2\}$ – складний.

Функціональні залежності:

$\{K_1, K_2\} \rightarrow \{A_1, \dots, A_n, B_1, \dots, B_m\}$ – залежність всіх атрибутів від ключа відношення.

$\{K_1\} \rightarrow \{A_1, \dots, A_n\}$ – залежність деяких атрибутів від частини складного ключа.

Декомповані відношення:

$R_1(K_1, K_2, B_1, \dots, B_m)$ – залишок від вихідного відношення. Ключ: $\{K_1, K_2\}$.

$R_1(K_1, A_1, \dots, A_n)$ – атрибути, винесені з вихідного відношення разом з частиною складного ключа. Ключ K_1 .

Третя Нормальна Форма (3НФ)

Атрибути називаються взаємно незалежними, якщо жоден з них не є функціонально залежним від іншого.

Відношення R знаходиться в третій нормальній формі (3НФ) тоді і тільки тоді, коли відношення знаходиться в 2 НФ і всі неключові атрибути взаємно незалежні.

2 НФ містить транзитивні залежності. Якщо для атрибутів A , B і C деякого відношення існують залежності виду $A \rightarrow B$ і $B \rightarrow C$, це означає, що атрибут C **транзитивно залежить** від атрибута A , через атрибут B (за умови, що атрибут A функціонально не залежить ні від атрибута B , ні від атрибута C). Таким чином, 3 НФ вільна про транзитивних залежностей.

Відношення `СТУДЕНТ_КАФЕДРА` не перебуває у 3НФ, тому що є функціональна залежність неключових атрибутів (залежність телефону кафедри від коду кафедри):

Код кафедри $\rightarrow$ Телефон кафедри

Номер заліковки $\rightarrow$ Телефон кафедри (явно видно транзитивну залежність)

Для того щоб усунути залежність неключових атрибутів, потрібно зробити декомпозицію відношень на кілька відношень. При цьому ті *неключові атрибути, які є залежними*, виносяться в окреме відношення.

Відношення `СТУДЕНТ_КАФЕДРА` декомпозується на два відношення - `СТУДЕНТ` і `КАФЕДРА`.

Відношення `СТУДЕНТ` (Код кафедри, Номер заліковки, ПІБ студента)

Функціональні залежності:

Залежність атрибутів, що характеризують студента від номера заліковки студента:

Номер заліковки → Код кафедри

Номер заліковки → ПБІ студента

Відношення КАФЕДРА (Код кафедри, Телефон кафедри)

Функціональні залежності:

Залежність номера телефону кафедри від коду кафедри:

Код кафедри → Телефон кафедри

Номер заліковки	ПБІ студента	Код кафедри
ТЗ3102	Богданенко О. Н.	101
ТЗ3208	Лисенко Н. І.	101
ТМ3107	Іванов Н. І.	102
ТМ3107	Іванов Н. І.	102
ТІ3112	Сидоров К. А.	103

а

Код кафедри	Телефон кафедри
101	431-45-78
101	431-45-78
102	452-94-74
102	452-94-74
103	428-03-29

б

Рис. 4.6. Декомпозиція відношення СТУДЕНТ_КАФЕДРА:
а – відношення СТУДЕНТ, б – відношення КАФЕДРА

Звернемо увагу на те, що атрибут Код кафедри, *не був ключовим* щодо СТУДЕНТ_КАФЕДРА, *стає потенційним ключем* щодо КАФЕДРА. Саме за рахунок цього усувається надмірність, пов'язана з багаторазовим зберіганням одних і тих же номерів телефонів.

Висновок: таким чином, всі виявлені аномалії оновлення усунені. Реляційна модель, що складається з трьох відношень СТУДЕНТ, КАФЕДРА, СЕКЦІЯ, що знаходяться в третій нормальній формі, є адекватною описаної моделі предметної області, і вимагає наявності тільки тих тригерів, які підтримують цілісність згідно з посиланням. Такі тригери є стандартними і не вимагають великих зусиль в розробці.

Алгоритм нормалізації (приведення до 3НФ)

Алгоритм нормалізації (тобто алгоритм приведення відношень до 3НФ) описується наступним чином.

Крок 1 (Приведення до 1НФ). На першому кроці задається одне або кілька відношення, що відображають поняття предметної області. За моделлю предметної області (не за зовнішнім виглядом отриманих відносин!) виписуються виявлені функціональні залежності. Всі відношення автоматично знаходяться в 1НФ.

Крок 2 (Приведення до 2НФ). Якщо в деяких відношеннях виявлена залежність атрибутів від частини складного ключа, то проводимо декомпозицію цих відношень на кілька відношень в такий спосіб: ті атрибути, які залежать від частини складного ключа виносяться в окреме відношення разом з цією частиною ключа. У вихідному відношенні залишаються всі ключові атрибути.

Крок 3 (Приведення до 3НФ). Якщо в деяких відношеннях виявлена залежність деяких неключових атрибутів від інших неключових атрибутів, то проводимо декомпозицію цих відношень таким чином: ті неключові атрибути, які залежать від інших неключових атрибутів, виносяться в окреме відношення. У новому відношенні ключем стає детермінант функціональної залежності:

Вихідне відношення: $R(K, A_1, \dots, A_n, B_1, \dots, B_m)$.

Ключ: K .

Функціональні залежності:

$K \rightarrow A_1, \dots, A_n, B_1, \dots, B_m$ – залежність всіх атрибутів від ключа відношення.

$\{A_1, \dots, A_n\} \rightarrow \{B_1, \dots, B_m\}$ – залежність деяких неключових атрибутів від інших неключових атрибутів.

Декомповані відношення:

$R_1(K, A_1, \dots, A_n)$ – залишок від вихідного відношення. Ключ K .

$R_2(A_1, \dots, A_n, B_1, \dots, B_m)$ – атрибути, винесені з початкового відношення разом з детермінантом функціональної залежності. Ключ $\{A_1, \dots, A_n\}$.

Зауваження. На практиці, при створенні логічної моделі даних, як правило, не дотримуються прямо приведенню алгоритму нормалізації. Досвідчені розробники зазвичай відразу будують відношення в 3НФ. Крім того, основним засобом розробки логічних моделей даних є різні варіанти ER-діаграм. Особливість цих діаграм в тому, що вони відразу дозволяють створювати відношення в 3НФ. Проте, наведений алгоритм важливий з двох причин. *По-перше*, цей алгоритм показує, які проблеми виникають при розробці слабо нормалізованих відношень. *По-друге*, як правило, модель предметної області ніколи не буває правильно розроблена з першого кроку. Експерти предметної області можуть забути про будь-що згадати, розробник може неправильно зрозуміти експерта, під час розробки можуть змінитися правила, прийняті в предметної області, і т.д. Все це може привести до появи нових залежностей, які були відсутні в первісній моделі предметної області. Тут як раз і необхідно

використовувати алгоритм нормалізації хоча б для того, щоб переконатися, що відношення залишилися в 3НФ і логічна модель не погіршилася.

Коректність процедури нормалізації - декомпозиція без втрат. теорема Хеза

Як було показано вище, алгоритм нормалізації полягає у виявленні функціональних залежностей предметної області і відповідної декомпозиції відношень. Припустимо, що ми вже маємо працюючу систему, в якій накопичені дані. Нехай дані коректні в поточний момент, тобто факти предметної області правильно відображаються поточним станом бази даних. Якщо в предметній області виявлена нова функціональна залежність (або вона була пропущена на етапі моделювання предметної області, або просто змінилася предметна область), то виникає необхідність заново нормалізувати дані. При цьому деякі відношення доведеться декомпонувати відповідно до алгоритму нормалізації. Виникають природні запитання - що станеться з уже накопиченими даними? Чи не будуть дані втрачені в ході декомпозиції? Чи можна повернутися назад до вихідних відношень, якщо буде прийнято рішення відмовитися від декомпозиції, відновляться при цьому дані?

Для відповідей на ці питання потрібно відповісти на питання - що ж являє собою декомпозиція відношень з точки зору операцій реляційної алгебри? При декомпозиції ми з одного відношення отримуємо два або більше відношень, кожне з яких містить частину атрибутів вихідного відношення. В отриманих нових відношеннях необхідно видалити дублікати рядків, якщо такі виникли. Це в точності означає, що декомпозиція відношень є не що інше, як взяття однієї або декількох проєкцій вихідного відношення так, щоб ці проєкції в сукупності містили (можливо, з повтореннями) *всі* атрибути вихідного відношення. Тобто, при декомпозиції *не повинні губитися атрибути* відношення. Але при декомпозиції також не повинні загубитися і самі дані. Дані можна вважати не втраченими в тому випадку, якщо можлива зворотна операція - по декомпонованим відношенням можна відновити вихідне відношення *в точності в колишньому вигляді*. Операцією, зворотної операції проєкції, є операція з'єднання відношень. Є велика кількість видів операції з'єднання (буде розглянуто нижче).

При відновленні вихідного відношення шляхом з'єднання проєкцій не повинні з'явитися нові атрибути, то необхідно використовувати природне з'єднання.

Проекція $R[X]$ овідношення R на множину атрибутів X називається власною, якщо множина атрибутів X є власною підмножиною множини атрибутів відношення R (тобто множина атрибутів X не співпадає з множиною всіх атрибутів відношення R).

Власні проєкції R_1 і R_2 відношення R називаються декомпозицією без втрат, якщо відношення R точно відновлюється з них за допомогою природнього з'єднання для будь-якого стану відношення R :

$$R_1 JOIN R_2 = R.$$

Розглянемо приклад, який показує, що декомпозиція без втрат відбувається не завжди.

Нехай дано відношення R:

Номер	Прізвище	Зарплата
1	Іванов	1000
2	Петров	1000

Рис. 4.7. Відношення R

Розглянемо перший варіант декомпозиції відношення R на два відношення:

Номер	Зарплата
1	1000
2	1000

Рис. 4.8. Відношення R1

Прізвище	Зарплата
Іванов	1000
Петров	1000

Рис. 4.9. Відношення R2

Природне з'єднання цих проєкцій, що мають загальний атрибут Зарплата, очевидно, буде наступним (кожен рядок однієї проєкції з'єднається з кожним рядком іншої проєкції):

Номер	Прізвище	Зарплата
1	Іванов	1000
1	Петров	1000
2	Іванов	1000
2	Петров	1000

Рис. 4.10. Відношення $R_1 JOIN R_2 \neq R$

Отже, дана декомпозиція не є декомпозицією без втрат, тому що вихідне відношення *не відновлюється в точному вигляді* за проєкціями (сірим кольором виділені зайві кортежі).

Розглянемо інший варіант декомпозиції:

Номер	Прізвище
1	Іванов
2	Петров

Рис. 4.11. Відношення R1

Номер	Зарплата
1	1000
2	1000

Рис. 4.12. Відношення R2

За даними проєкціями, які мають загальний атрибут Номер, вихідне відношення відновлюється в точному вигляді. Проте, не можна сказати, що дана декомпозиція є декомпозицією без втрат, тому що ми розглянули тільки *один конкретний стан* відношення, і не можемо сказати, чи буде і в інших станах відношення відновлюватися точно. Наприклад, припустимо, що відношення перейшло в стан:

Номер	Прізвище	Зарплата
1	Іванов	1000
2	Петров	1000
2	Сидоров	2000

Рис. 4.13. Відношення R

Здається, що цього не може бути, тому що значення в атрибуті Номер повторюються. Але ми ж нічого не говорили про ключі цього відношення! Зараз проєкції матимуть вигляд:

Номер	Прізвище
1	Іванов
2	Петров
2	Сидоров

Рис. 4.14. Відношення R1

Номер	Зарплата
1	1000
2	1000
2	2000

Рис. 4.15. Відношення R2

Природне з'єднання цих проєкцій буде містити зайві кортежі:

Номер	Прізвище	Зарплата
1	Іванов	1000
2	Петров	1000
2	Петров	2000
2	Сидоров	1000
2	Сидоров	2000

Рис. 4.16. Відношення $R_1 \text{ JOIN } R_2 \neq R$

Висновок: Таким чином, *без додаткових обмежень* на відношення не можна говорити про декомпозицію без втрат.

Такими додатковими обмеженнями і є функціональні залежності. Має місце наступна теорема Хеза:

Теорема (Хеза).

Нехай $R(A, B, C)$ являється відношенням, і A, B, C – атрибути або множина атрибутів цього відношення. Якщо є функціональна залежність $A \rightarrow B$, то проєкції $R_1 = R[A, B]$ і $R_2 = R[A, C]$ створюють декомпозицію без втрат.

Основний сенс теореми Хеза полягає в доказі того, що при цьому *не з'являються нові кортежі*, відсутні в вихідному відношенні.

При виконанні декомпозиції і наступному відновленні відношення за допомогою природного з'єднання, кортежі вихідного відношення *не будуть втрачені*.

Так як алгоритм нормалізації (приведення відношень до 3НФ) заснований на наявних у відношеннях функціональних залежностей, то теорема Хеза показує, що алгоритм нормалізації є коректним, тобто в ході нормалізації *не відбувається втрати інформації*.

Нормальна Форма Бойса-Кодда (НФБК)

При приведенні відношення за допомогою алгоритму нормалізації до відношення в 3НФ неявно передбачалося, що всі відношення містять один потенційний ключ. Це не завжди вірно. Розглянемо наступний приклад відношення, що містить два ключі.

Нехай потрібно зберігати дані про отримання книг деякими кафедрами. Припустимо, що назви кафедр є унікальними. Крім того, кожна кафедра має свій унікальний код. Ці дані зведені в відношення представлене на рис. 4.17.

Код кафедри	Назва кафедри	Код книги	Кількість книг

101	Інформаційно-телекомунікаційні мережі	111	32
101	Інформаційно-телекомунікаційні мережі	112	25
101	Інформаційно-телекомунікаційні мережі	113	30
102	Телекомунікаційні мережі	121	42
102	Телекомунікаційні мережі	122	45
103	Засоби телекомунікацій	131	40

Рис.4.17. Відношення КАФЕДРА_КНИГИ

Дане відношення містить два потенційних ключа - {Код кафедри, Код книги} і {Назва кафедри, Код книги}. Видно, що дані зберігаються в відношенні з надмірністю - при зміні назва кафедри, цю назву потрібно змінити у всіх кортежах, де воно зустрічається. Чи можна цю аномалію усунути за допомогою алгоритму нормалізації, описаного в попередньому розділі? Для цього потрібно виявити наявні функціональні залежності:

Код кафедри → Назва кафедри – назва кафедри залежить від коду кафедри.

Назва кафедри → Код кафедри – код кафедри залежить від назви кафедри.

{Код кафедри, Код книги} → Количество книг – отримувана кількість залежить від першого ключа відношення.

{Код кафедри, Код книги} → Назва кафедри – назва кафедри залежить від першого люча відношення.

{ Назва кафедри, Код книги} → Количество книг – отримувана кількість залежить від другого ключа відношення.

{ Назва кафедри, Код книги} → Код кафедри – код кафедри залежить від другого ключа відношення.

Дане відношення не містить *неключових* атрибутів, залежних від частини складного ключа (див. визначення 2НФ). Дійсно, від частини складного ключа залежать атрибути Назва кафедри та Код кафедри, але вони самі є ключовими. Таким чином, відношення знаходиться в 2НФ.

Крім того, відношення не містить залежних один від одного *неключових* атрибутів, тому що неключових атрибут всього один - Кількість книг (див. визначення 3НФ). Таким чином, показано, що відношення КАФЕДРА_КНИГ знаходиться в 3НФ.

Таким чином, описаний раніше алгоритм нормалізації непридатний до цього відношення. Очевидно, однак, що аномалія даного відношення усувається шляхом декомпозиції його на два наступних відношення:

Код кафедри	Назва кафедри
101	Інформаційно-телекомунікаційні мережі
102	Телекомунікаційні мережі
103	Засоби телекомунікацій

а

Код кафедри	Код книги	Кількість книг
101	111	32
101	112	25
101	113	30
102	121	42
102	122	45
103	131	40

б

Рис. 4.18. Декомпозиція відношення КАФЕДРА_КНИГИ на відношення:
а – КАФЕДРА; б - ОТРИМУВАНІ_КНИГИ_1

Відношення R знаходиться в нормальній формі Бойса-Кодда (НФБК) тоді і тільки тоді, коли детермінанти всіх функціональних залежностей є потенційними ключами.

Зауваження. Якщо відношення знаходиться в НФБК, то воно автоматично знаходиться і в ЗНФ. Дійсно, це відразу випливає з визначення ЗНФ.

Відношення КАФЕДРА_КНИГИ не перебуває у НФБК, тому що є залежності Код кафедри → Назва кафедри та Назва кафедри → Код кафедри, детермінанти яких не є потенційними ключами.

Для того щоб усунути залежність від детермінантів, які не є потенційними ключами, необхідно провести декомпозицію, виносячи ці детермінанти і залежні від них частини в окреме відношення. Відношення КАФЕДРА і ОТРИМАНІ_КНИГИ_1, отримані в результаті декомпозиції знаходяться в НФБК.

Зауваження. Наведена декомпозиція відношень КАФЕДРА_КНИГИ на відношення КАФЕДРА і ОТРИМАНІ_КНИГИ не є єдино можливою. Альтернативною декомпозицією є декомпозиція на наступні відношення:

Код кафедри	Назва кафедри
101	Інформаційно-телекомунікаційні мережі

102	Телекомунікаційні мережі
103	Засоби телекомунікацій

а

Назва кафедри	Код книги	Кількість книг
Інформаційно-телекомунікаційні мережі	111	32
Інформаційно-телекомунікаційні мережі	112	25
Інформаційно-телекомунікаційні мережі	113	30
Телекомунікаційні мережі	121	42
Телекомунікаційні мережі	122	45
Засоби телекомунікацій	131	40

б

Рис. 4.19. Декомпозиція відношення КАФЕДРА_КНИГИ на відношення:
а – КАФЕДРА; б – ОТРИМУВАНІ_КНИГИ_2

На перший погляд, така декомпозиція гірше, ніж перша. Дійсно, назва кафедр як і раніше повторюються, і при зміні назви кафедри, цю назву доведеться міняти одночасно в декількох місцях (тим більше відразу в двох відношеннях!). Здається, що ситуація стала ще гірше, ніж була до декомпозиції. Однак таке відчуття виникає тому, що ми інтуїтивно вважаємо, що назви кафедр можуть змінюватися, а коди - ні. Якщо ж припустити, що коди кафедр теж можуть змінюватися (чому б ні - ректор наказав поміняти коди кафедр!), То перша декомпозиція виходить такою же "поганою" як і друга - повторювані коди доведеться міняти одночасно в декількох місцях і також відразу в двох відношеннях

Насправді ніякого протиріччя тут немає. Відносно ОТРИМУВАНІ_КНИГИ_2 атрибут Назва кафедри є зовнішнім ключем, що служить для зв'язку з відношенням КАФЕДРА. Тому, при зміні назви кафедри, це зміна проводиться щодо КАФЕДРА і каскадно поширюється на відношення ОТРИМУВАНІ_КНИГИ_2 абсолютно так, як зміна коду кафедри каскадно поширюється на відношення ОТРИМУВАНІ_КНИГИ_1. Тому, формально обидві декомпозиції зовсім рівноправні. У реальній роботі розробник вибере, звичайно, першу декомпозицію, але тут важливо підкреслити, що його вибір заснований зовсім на інших міркуваннях, що не мають відношення до формальної теорії нормальних форм.

Зауваження. Відношення ОТРИМУВАНІ_КНИГИ_1, отримане в результаті декомпозиції має всього один потенційний ключ. Тому, для аналізу відношення ОТРИМУВАНІ_КНИГИ_1 не потрібно залучати визначення НФБК, досить визначення ЗНФ. Хоча відношення КАФЕДРА має два потенційних ключа, але, тому що інших атрибутів в ньому немає, воно вже так просто влаштовано, що спростити його далі не можна. Виникає питання, чи є нетривіальні приклади відношень в НФБК, які не перебувають в ЗНФ і не такі прості, як відношення КАФЕДРА?

Припустимо, що як і раніше необхідно враховувати отримання книг, але кожен акт видачі книг повинен мати певний унікальний номер (назовемо його "наскрізний номер видачі книг"). Відношення матиме такий вигляд:

Код кафедри	Код книги	Кількість книг	Наскрізний номер видачі книг
101	111	32	1
101	112	25	2
101	113	30	3
102	121	42	4
102	122	45	5
103	131	40	6

Рис. 4.20. Відношення ВИДАЧА_КНИГ_З_НОМЕРОМ

Одним потенційним ключем даного відношення є, як і раніше, пара атрибутів {Код кафедри, Код книги}. Іншим ключем, в силу унікальності наскрізного номера, є атрибут Наскрізний номер видачі книг. В даному відношенні є наступні функціональні залежності:

Залежність атрибутів від першого ключа відношення:

{Код кафедри, Код книги} → Кількість книг,
 {Код кафедри, Код книги} → Наскрізний номер видачі книг,

Залежність атрибутів від другого ключа відношення:

Наскрізний номер видачі книг → Код кафедри,
 Наскрізний номер видачі книг → Код книги,
 Наскрізний номер видачі книг → Кількість книг,

Залежності, які є наслідком залежностей від ключів відношення:

{Код кафедри, Код книги} → {Кількість книг, Наскрізний номер видачі книг},
 Наскрізний номер видачі книг → {Код кафедри, Код книги},
 Наскрізний номер видачі книг → {Код кафедри, Кількість книг },
 Наскрізний номер видачі книг → {Код книги, Кількість книг },
 Наскрізний номер видачі книг → {Код кафедри, Код книги, Кількість книг }.

Як можна помітити, детермінанти (унікальні ключі) всіх залежностей є потенційними ключами, тому дане відношення знаходиться в НФБК. Особливістю даного відношення є те, що воно має два абсолютно незалежних потенційних ключа.

Алгоритм нормалізації (приведення до НФБК)

Крок 1. Якщо є відношення, що містять кілька потенційних ключів, перевіряємо, чи є функціональні залежності, детермінанти яких не є потенційними ключами.

Крок 2. Якщо функціональні залежності є, проводимо подальшу декомпозицію відношень. Ті атрибути, які залежать від детермінантів, які не є потенційними ключами виносяться в окреме відношення разом з детермінантами.

Четверта Нормальна Форма (4 НФ)

Розглянемо наступний приклад. Нехай потрібно враховувати дані про абітурієнтів, які вступають до ВНЗ. При аналізі предметної області були виділені наступні вимоги:

- Кожен абітурієнт має право скласти іспити на кілька факультетів одночасно.
- Кожен факультет має свій список предметів які потрібно здавати.
- Один і той же предмет може здаватися на кількох факультетах.
- Абітурієнт зобов'язаний здавати всі предмети, зазначені для факультету, на який він надходить, незважаючи на те, що він, може бути, вже здавав такі ж предмети на іншому факультеті.

Припустимо, що нам потрібно зберігати дані про те, які предмети повинен здавати кожен абітурієнт. Спробуємо зберігати дані в одному відношенні АБІТУРІЄНТИ_ФАКУЛЬТЕТИ_ПРЕДМЕТИ:

Абітурієнт	Факультет	Предмет
Іванов	Математичний	Математика
Іванов	Математичний	Інформатика
Іванов	Фізичний	Математика
Іванов	Фізичний	Фізика
Петров	Математичний	Математика
Петров	Математичний	Інформатика

Рис. 4.21. Відношення АБІТУРІЄНТИ_ФАКУЛЬТЕТИ_ПРЕДМЕТИ

В даний момент в відношенні зберігається інформація про те, що абітурієнт Іванов надходить на два факультети (математичний і фізичний), а абітурієнт Петров - тільки на математичний. Крім того, можна зробити висновок, що на

математичному факультеті потрібно здавати математику і інформатику, а на фізичному - математику і фізику.

Здається, що в відношенні є аномалія оновлення, пов'язана з тим, що дублюються прізвища абітурієнтів, найменування факультетів і найменування предметів. Однак ця аномалія легко усувається стандартним способом - винесенням всіх найменувань в окремі відношення, залишаючи в вихідному відношенні тільки відповідні номери:

Номер абітурієнта	Номер факультета	Номер предмета
1	001	101
1	001	102
1	002	101
1	002	103
2	001	101
2	001	102

Рис. 4.22. Модифіковане відношення
АБІТУРІЄНТИ_ФАКУЛЬТЕТИ_ПРЕДМЕТИ

Номер абітурієнта	Прізвище
1	Іванов
2	Петров

а

Номер факультета	Факультет
001	Математичний
001	Фізичний

б

Номер предмета	Предмет
101	Математика
102	Інформатика
103	Фізика

в

Рис. 4.23. а - Відношення АБІТУРІЄНТИ

б - Відношення ФАКУЛЬТЕТИ

в - Відношення ПРЕДМЕТИ

Тепер кожне найменування зустрічається тільки в одному місці.

І все-таки як у вихідному, так і в модифікованому відношенні є аномалії оновлення, що виникають при спробі вставити або видалити кортежі.

Аномалія вставки

При спробі додати в відношення АБІТУРІЄНТИ_ФАКУЛЬТЕТИ_ПРЕДМЕТИ новий кортеж, наприклад (Сидоров, Математичний, Математика), необхідно буде додати також і кортеж (Сидоров, Математичний, Інформатика), тому що всі абітурієнти математичного факультету *зобов'язані* мати один і той же список предметів для. Відповідно, при спробі вставити в модифіковане щодо кортеж (3, 001, 101), ми зобов'язані вставити в нього також і кортеж (3, 001, 102).

Аномалія видалення

При спробі видалити кортеж (Іванов, Математичний, Математика), необхідно буде видалити також і кортеж (Іванов, Математичний, Інформатика) по тій же самій причині.

Таким чином, вставка і видалення кортежів не може бути виконана незалежно від інших кортежів відношення.

Крім того, якщо видалити кортеж (Іванов, Фізичний, Математика), а разом з ним і кортеж (Іванов, Фізичний, Фізика), то буде втрачена інформація про предмети, які повинні здаватися на фізичному факультеті.

Декомпозиція відношення АБІТУРІЄНТИ_ФАКУЛЬТЕТИ_ПРЕДМЕТИ для усунення зазначених аномалій не може бути виконана на основі функціональних залежностей, тому що це відношення *не містить ніяких функціональних залежностей*. Це відношення є повністю ключовим, тобто ключем відношення є вся множина атрибутів. Але ясно, що якийсь взаємозв'язок між атрибутами є. Цей взаємозв'язок описується поняттям **багатозначної залежності**.

Нехай R – відношення, і X, Y, Z – деякі з його атрибутів (або непересічні множини атрибутів). Тоді атрибути (множини атрибутів) Y і Z **багатозначно залежать** від X (позначається $X \twoheadrightarrow Y|Z$), тоді і тільки тоді, коли з того, що в відношенні R містяться кортежі $r_1 = (x, y, z_1)$ і $r_2 = (x, y_1, z)$ виходить, що в відношенні R міститься також і кортеж $r_3 = (x, y, z)$.

Зауваження. Міняючи місцями кортежі r_1 і r_2 у визначенні багатозначної залежності, отримаємо, що в відношенні R повинен міститися також і кортеж $r_4 = (x, y_1, z_1)$. Таким чином, атрибути Y і Z , багатозначно залежать від X , поводяться "симетрично" по відношенню до атрибута X .

В відношенні АБІТУРІЕНТИ_ФАКУЛЬТЕТИ_ПРЕДМЕТИ є багатозначна залежність Факультет $\rightarrow\rightarrow$ Абітурієнт | Предмет.

Словами це можна висловити так - для кожного факультету (для кожного значення з X) кожен абітурієнт, що вступає на нього (значення з Y) здає *один і той же список* предметів (набір значень з Z), для кожного факультету (для кожного значення з X) кожен здається на факультеті іспит (значення з Z) здається *одним і тим же списком* абітурієнтів (набір значень з Y). Саме наявність цієї залежності не дозволяє незалежно вставляти і видаляти кортежі. Кортежі зобов'язані вставлятися і видалятися одночасно *цілими наборами*.

Зауваження. Якщо в відношення R міститься не менше трьох атрибутів X, Y, Z і є функціональна залежність $X \rightarrow Y$, то є і *многозначна залежність* $X \rightarrow\rightarrow Y|Z$.

Дійсно, діючи формально відповідно до визначення багатозначної залежності, припустимо, що у відношенні R містяться кортежі $r_1 = (x, y, z_1)$ і $r_2 = (x, y_1, z)$. В силу функціональної залежності $X \rightarrow Y$ звідси слідує, що $y = y_1$. Але тоді кортеж $r_3 = (x, y, z)$ в точності збігається з кортежем $r_2 = (x, y_1, z)$ і, отже, міститься в відношенні R . Таким чином, є багатозначна залежність $X \rightarrow\rightarrow Y|Z$.

Таким чином, *поняття багатозначної залежності є узагальненням поняття функціональної залежності*.

*Многозначна залежність $X \rightarrow\rightarrow Y|Z$ називається **нетривіальною багатозначною залежністю**, якщо не існує функціональних залежностей $X \rightarrow Y$ і $X \rightarrow Z$.*

В відношенні АБІТУРІЕНТИ_ФАКУЛЬТЕТИ_ПРЕДМЕТИ є саме нетривіальна багатозначна залежність Факультет $\rightarrow\rightarrow$ Абітурієнт | Предмет. В силу нетривіальності цієї залежності ми не можемо скористатися теоремою Хеза для декомпозиції відношення. Однак Фейджином Р. доведена наступна теорема:

Теорема (Фейджина).

Нехай X, Y, Z – непересічні множини атрибутів відношення $R(X, Y, Z)$.

Декомпозиція відношення R на проєкції $R_1 = R[X, Y]$ і $R_2 = R[X, Z]$ буде декомпозицією без втрат тоді і тільки тоді, коли є многозначна залежність $X \rightarrow\rightarrow Y|Z$.

Зауваження. Якщо залежність $X \rightarrow\rightarrow Y|Z$ є тривіальною, тобто існує одна з функціональних залежностей $X \rightarrow Y$ або $X \rightarrow Z$, то отримуємо теорему Хеза.

Відношення R знаходиться в **четвертій нормальній формі (4НФ)** тогдa і тільки тоді, коли відношення знаходиться в НФБК і не містить нетривіальних багатозначних залежностей.

Відношення АБІТУРІЕНТИ_ФАКУЛЬТЕТИ_ПРЕДМЕТИ знаходиться в НФБК, але не в 4НФ. Згідно з теоремою Фейджина, це ставлення можна без втрат декомпонувати на відношення:

Факультет	Абітурієнт
Математичний	Іванов
Фізичний	Іванов
Математичний	Іванов

а

Факультет	Предмет
Математичний	Математика
Математичний	Інформатика
Фізичний	Фізика
Фізичний	Математика

б

Рис. 4.24. а – Відношення ФАКУЛЬТЕТИ_АБІТУРІЄНТИ,

б – Відношення ФАКУЛЬТЕТИ_ПРЕДМЕТИ

В отриманих відношеннях усунені аномалії вставки і видалення, характерні для відношення АБІТУРІЕНТИ_ФАКУЛЬТЕТИ_ПРЕДМЕТИ.

Зауважимо, що отримані відношення залишилися повністю ключовими, і в них, як і раніше немає функціональних залежностей.

Відношення з нетривіальними багатозначними залежностями виникають, як правило, в результаті природного з'єднання двох відношень за загальним полем, яке не є ключовим ні в одному з відношень. Фактично це призводить до спроби зберігати в одному відношенні інформацію про дві незалежні сутності. В якості ще одного прикладу можна навести ситуацію, коли викладач може викладати декілька предметів і керувати низкою наукових проєктів. Зберігання інформації про предмети і наукових проєктах в одному відношенні призводить до виникнення нетривіальної багатозначної залежності Викладач $\rightarrow$ Предмети|Научні проєкти.

Алгоритм нормалізації (приведення до 4 НФ)

Крок 1. Якщо є відношення, що містять кілька потенційних ключів, перевіряємо, чи є функціональні залежності, детермінанти яких не є потенційними ключами.

Крок 2. Якщо функціональні залежності є, проводимо подальшу декомпозицію відношень. Ті атрибути, які залежать від детермінантів, які не є потенційними ключами виносяться в окреме відношення разом з детермінантами.

Крок 3. Якщо в стосунках виявлені нетривіальні багатозначні залежності, проводимо декомпозицію для виключення таких залежностей.

П'ята Нормальна Форма (5 НФ)

Функціональні і багатозначні залежності дозволяють зробити декомпозицію вихідного відношення без втрат на дві проекції. Можна, однак, привести приклади відношень, які не можна декомпонувати без втрат ні на які дві проекції.

Розглянемо наступне відношення R:

X	Y	Z
1	1	2
1	2	1
2	1	1
1	1	1

Рис. 4.25. Відношення R.

Всілякі проекції відношення R, що включають по два атрибути, мають вигляд:

X	Y
1	1
1	2
2	1

Рис. 4.26. Проекція $R_1=R[X,Y]$.

X	Z
1	2
1	1
2	1

Рис. 4.27. Проекція $R_2=R[X,Z]$.

Y	Z
1	2

2	1
1	1

Рис. 4.28. Проекція $R_3=R[Y,Z]$.

Як легко помітити, відношення R не відновлюється ні по одному з *попарних* з'єднань $R_1 \text{ JOIN } R_2$, $R_1 \text{ JOIN } R_3$ або $R_2 \text{ JOIN } R_3$. Дійсно, з'єднання $R_1 \text{ JOIN } R_2$ має вигляд:

X	Y	Z
1	1	2
1	1	1
1	2	2
1	2	1
2	1	1

Рис. 4.29. $R_1 \text{ JOIN } R_2$.

Сірим кольором виділено зайвий кортеж, відсутній в відношенні R . Аналогічно (в силу міркувань симетрії) і інші попарні з'єднання не відновлюють відношення R .

Однак відношення R відновлюється з'єднанням *усіх трьох* проекцій:

$$R_1 \text{ JOIN } R_2 \text{ JOIN } R_3 = R$$

Це говорить про те, що між атрибутами цього відношення також є деяка залежність, але ця залежність не є ні функціональною, ні багатозначною залежністю.

Нехай R буде відношенням, а $A, B, \dots, Z$ – довільними (можливо пересічними) підмножинами множини атрибутів відношення R . Тоді відношення R задовольняє залежності з'єднання

$$* (A, B, \dots, Z)$$

тоді і тільки тоді, коли воно рівносильне з'єднанню всіх своїх проекцій з підмножинами атрибутів $A, B, \dots, Z$, тобто

$$R = R[A] \text{ JOIN } R[B] \text{ JOIN } \dots \text{ JOIN } R[Z]$$

Можна *припустити*, що відношення R в прикладі 3 задовольняє наступній залежності з'єднання: $*(XY, XZ, YZ)$.

Стверджувати, що це саме так поки не можна, тому що визначення залежності з'єднання повинно виконуватися для *будь-якого* стану відношення R , а не тільки для стану, наведеного в прикладі.

Покажемо, що залежність з'єднання є узагальненням поняттям багатозначної залежності. Дійсно, відповідно до теореми Фейджина, відношення $R(X, Y, Z)$ може бути декомпозовано без втрат на проєкції $R_1 = R[X, Y]$ і $R_2 = R[X, Z]$ тоді і тільки тоді, коли є багатозначна залежність $X \twoheadrightarrow Y|Z$. Згідно визначенню залежності з'єднання, теорема Фейджина може бути переформульована наступним чином:

Теорема Фейджина (інше формулювання).

Відношення $R(X, Y, Z)$ задовільняє залежність з'єднання $*$ (XY, XZ) тоді і тільки тоді, коли є багатозначна залежність $X \twoheadrightarrow Y|Z$.

Оскільки теорема Фейджина є взаємно оберненою, то її можна взяти в якості визначення багатозначної залежності. Таким чином, багатозначна залежність є окремим випадком залежності з'єднання, тобто, якщо в плані є багатозначна залежність, то є і залежність з'єднання. Зворотнє, звичайно, невірно.

*Залежність з'єднання $*$ $(A, B, \dots, Z)$ називається **нетривіальною залежністю з'єднання**, якщо виконується дві умови:*

- *Одна з множин атрибутів $A, B, \dots, Z$ не містить потенційного ключа відношення R .*
- *Жодна з множин атрибутів не збігається з усією множиною атрибутів відношення R .*

Для зручності роботи сформулюємо це визначення так само і в негативній формі:

*Залежність з'єднання $*$ $(A, B, \dots, Z)$ називається **тривіальною залежністю з'єднання**, якщо виконується одна із умов:*

- *Або всі множини атрибутів $A, B, \dots, Z$ містять потенційний ключ відношення R .*
- *Або одна з множин атрибутів збігається з усією множиною атрибутів відношення R .*

*Відношення R знаходиться в **п'ятій нормальній формі (5НФ)** тоді і тільки тоді, коли будь-яка наявна залежність з'єднання є тривіальною.*

Визначення 5НФ може стати більш зрозумілим, якщо сформулювати його в негативній формі:

*Відношення R не перебуває у **п'ятій нормальній формі (5НФ)**, якщо в відношенні знайдеться нетривіальна залежність з'єднання.*

Повертаючись до розглянутого прикладу, стає зрозуміло, що не знаючи нічого про те, які потенційні ключі є в відношенні і як взаємопов'язані атрибути, не можна робити висновки про те, чи знаходиться дане відношення в 5НФ (як, втім,¹³⁵

і в інших нормальних формах). За даним конкретного прикладу можна тільки припустити, що відношення в прикладі не знаходиться в 5НФ. Припустимо, що аналіз предметної області дозволив виявити наступні залежності атрибутів у відношенні R:

1. Відношення R є повністю ключовим (тобто потенційним ключем відношення являється вся множина атрибутів).
2. Мітситьна наступна залежність (доволі дивна, з практичної точки зору): якщо в відношенні R є кортежі $r_1 = (x, y, z_1)$, $r_2 = (x, y_1, z)$ і $r_3 = (x_1, y, z)$ то звідси випливає, що в відношенні R є також і кортеж $r_4 = (x, y, z)$.

Затвердження. Доведемо, що при наявності обмежень (1) і (2), відношення знаходиться в 4НФ, але не в 5НФ.

Доведення. Покажемо, що відношення знаходиться в 4НФ. Згідно з визначенням 4НФ, необхідно показати, що відношення знаходиться в НФБК і не містить *нетривіальних* багатозначних залежностей. Оскільки ставлення є повністю ключовим, то воно автоматично знаходиться в НФБК. Якби в відношенні була багатозначна залежність (необов'язково нетривіальна), то, відповідно до теореми Фейджина, ставлення можна було б декомпозувати без втрат на дві проекції. Але приклад 3 показує, що таких декомпозицій немає (тут використовується те, що для доказу можливості декомпозиції необхідно довести її для всіх можливих станів відношення, а для доказу неможливості досить привести *один* контрприклад). Тому в відношенні немає жодних багатозначних залежностей.

Покажемо, що відношення не знаходиться в 5НФ. Для цього потрібно привести приклад нетривіальної залежності з'єднання. Природним кандидатом на неї є: $*$ (XY, XZ, YZ). Якщо це дійсно залежність з'єднання, то вона нетривіальна. Дійсно, жодне з множин атрибутів XY, XZ і YZ не збігається з множиною всіх атрибутів відношення R і не містить потенційного ключа.

Але чи є така декомпозиція саме залежністю з'єднання? Для цього потрібно показати, що декомпозиція на три проекції $R_1 = R[X, Y]$, $R_2 = R[X, Z]$ і $R_3 = R[Y, Z]$ є декомпозицією без втрат для *будь-якого* стану відношення R (саме тут міститься ключова тонкість, зазвичай пропускається при аналізі конкретного стану відношення R в розглянутому прикладі, і саме тут нам знадобляться знання про предметну область, виражені в утвердженні (2)).

Потрібно довести, що $R_1 \text{ JOIN } R_2 \text{ JOIN } R_3 = R$ для будь-якого стану відношення R.

Включення $R_1 \text{ JOIN } R_2 \text{ JOIN } R_3 \supseteq R$ доводиться як в теоремі Хеза. Таке включення виконується завжди для будь-якої декомпозиції відношення R.

Доведемо включення $R_1 \text{ JOIN } R_2 \text{ JOIN } R_3 \subseteq R$.

Нехай кортеж $r = (x, y, z) \in R_1 \text{ JOIN } R_2 \text{ JOIN } R_3$. Це значить, що в проекції $R_1 = R[X, Y]$ є кортеж $r_1 = (x, y)$, в проекції $R_2 = R[X, Z]$ міститься кортеж $r_2 = (x, z)$, а в проекції $R_3 = R[Y, Z]$ міститься кортеж $r_3 = (y, z)$. За визначенням проекції, знайдуться такі значення x_1, y_1, z_1 атрибутів X, Y, Z відповідно, що відношення R містить кортежі $r_1 = (x, y, z_1)$, $r_2 = (x, y_1, z)$ і $r_3 = (x_1, y, z)$. Але тоді за умовою (2) у відношенні R міститься також і кортеж $r = (x, y, z)$. Цим доведено необхідне включення. *Затвердження доведено.*

Продовження алгоритму нормалізації (приведення до 5НФ)

Вище був описаний алгоритм нормалізації як алгоритм приведення відношень до 3НФ. Тепер продовжимо цей алгоритм, довівши його до алгоритму приведення до 5НФ.

Крок 4 (Приведення до НФБК). Якщо є відношення, що містять кілька потенційних ключів, то необхідно перевірити, чи є функціональні залежності, детермінанти яких не є потенційними ключами. Якщо такі функціональні залежності є, то необхідно провести подальшу декомпозицію відношень. Ті атрибути, які залежать від детермінантів, які не є потенційними ключами виносяться в окреме відношення разом з детермінантами.

Крок 5 (Приведення до 4НФ). Якщо у відношеннях виявлені нетривіальні багатозначні залежності, то необхідно провести декомпозицію для виключення таких залежностей.

Крок 6 (Приведення до 5НФ). Якщо у відношеннях виявлені нетривіальні залежності з'єднання, то необхідно провести декомпозицію для виключення і таких залежностей.

4.2 Аналіз критеріїв для нормалізованих і ненормалізованих моделей даних

Зберемо воедино результати аналізу критеріїв, за якими можна оцінити вплив логічного моделювання даних на якість фізичних моделей даних і продуктивність бази даних:

Таблиця 4.1.

Критерій	Відношення слабо нормалізовані (1НФ, 2НФ)	Відношення сильно нормалізовані (3НФ)
Адекватність бази даних предметної області	ГІРШЕ (-)	КРАЩЕ(+)
Легкість розробки і супроводу бази даних	СКЛАДНІШЕ (-)	ПРОСТІШЕ(+)
Швидкість виконання вставки, оновлення, видалення	ПОВІЛЬНІШЕ (-)	БИСТРІШЕ(+)
Швидкість виконання вибірки даних	БИСТРІШЕ(+)	ПОВІЛЬНІШЕ (-)

Як видно з таблиці, більш сильно нормалізовані відношення виявляються краще спроектовані (три плюса, один мінус). Вони більше відповідають предметній області, легше в розробці, для них швидше виконуються операції модифікації бази даних. Це досягається ціною деякого уповільнення виконання операцій вибірки даних.

У слабо нормалізованих відношеннях єдина перевага - якщо до бази даних звертатися тільки до запитів на вибірку даних, то для слабо нормалізованих відношень такі запити виконуються швидше. Це пов'язано з тим, що в таких відношеннях уже як би вироблено з'єднання відношень і на це не витрачається час при вибірці даних.

Таким чином, вибір ступеня нормалізації відношень залежить від характеру запитів, з якими найчастіше звертаються до бази даних.

4.3 Доменно-ключова нормальна форма (ДКНФ)

У 1981 р Фагін опублікував важливу статтю, в якій він визначив доменно-ключову нормальну форму. Він показав, що ставлення в ДКНФ не має аномалій модифікації і, більш того, будь-яке відношення, яке не має аномалій модифікації,

має перебувати в ДКНФ. Це відкриття поклало кінець введення нормальних форм, і тепер в нормальних формах вищого порядку немає необхідності - по крайній мірі, для усунення аномалій модифікації.

Настільки ж важливо і те, що визначення доменно-ключовий нормальної форми зачіпає тільки поняття домену і ключа. Вони безпосередньо підтримуються існуючими СУБД (або, принаймні, можуть підтримуватися).

Поняття ДКНФ - відношення знаходиться в доменно-ключовий нормальній формі, якщо кожне обмеження, що накладаються на це відношення, є логічним наслідком визначення доменів і ключів.

Розглянемо важливі терміни, які фігурують в цьому визначенні: *обмеження, ключ і домен*.

Термін обмеження (constraint) має в даному визначенні навмисно широке тлумачення. Фагін визначає **обмеження** як *будь-яке правило, що регулює можливі статичні значення атрибутів і досить точно для того, щоб можна було встановити, виконується воно чи ні*. Правила редагування, обмеження взаємин і структури відношень, функціональні залежності і багатозначні залежності є прикладами таких обмежень. Фагін явно виключає звідси обмеження, які стосуються змін значень даних, або обмеження, що залежать від часу.

Наприклад, правило «Зарплата викладача за поточний період не може бути менше, ніж за попередній період» не підпадає під визначення обмеження, яке дав Фагін. За винятком обмежень, які залежать від часу, визначення Фагіна є широким і охоплює багато випадків.

Ключ (key) - *це унікальний ідентифікатор кортежу, як його вже визначили*.

Третій важливий термін у визначенні ДКНФ - домен (domain).

Домен - *це опис допустимих значень атрибута. Він складається з двох частин: фізичної опису і семантичного, або логічного, опису. Фізичний опис* - це безліч значень, які може приймати атрибут, а **логічний опис** - це сенс даного атрибута.

Доказ Фагіна відноситься до обох частин.

Говорячи неформально, відношення знаходиться в ДКНФ, якщо виконання обмежень на домени і ключі призводить до виконання всіх обмежень.

Розглянемо приклад доменно-ключової нормальної форми. Дано відношення **СТУДЕНТ** на рис 5.30, має атрибути Номер заліковки, Курс, Гуртожиток і Плата. Тут Плата - це сума, яку студент платить за проживання в гуртожитку.

СТУДЕНТ (Номер заліковки, Курс, Гуртожиток, Плата)

Ключ: Номер заліковки

Обмеження Гуртожиток→Плата

Номер заліковки не повинен починатися з цифри

Рис. 4.30. Приклад ДКНФ.

Номер заліковки функціонально визначає інші три атрибути, тому Номер заліковки є ключем. Припустимо, також знаємо, що, відповідно до вимог користувачів, Гуртожиток→Плата і Номер заліковки не повинен починатися з цифри. Якщо вдасться представити ці вимоги в якості логічних наслідків визначення доменів і ключів, то можна бути впевненим у відповідності з теорією Фрагіна, що аномалії модифікації не виникне. В даному прикладі це буде легко зробити.

Щоб реалізувати обмеження, яке вказує, що номери залікових книжок студентів не повинні починатися з цифри, включимо цю вимогу в визначення домену атрибута Номер заліковки (рис 4.31). Дотримання обмеження домену гарантує, що дана вимога буде виконано.

Постає питання про те, як зробити Гуртожиток ключем. Атрибут Гуртожиток не може бути ключем щодо СТУДЕНТ, так як в одному і тому ж гуртожитку живе більше одного студента, зате він може бути ключем свого власного відношення. Таким чином, можна ввести відношення ГКРТОЖИТОК_ПЛАТА з атрибутами Гуртожиток і Плата. Гуртожиток є ключем цього відношення. Ввівши це нове відношення, можна видалити атрибут Плата з відношення СТУДЕНТ. Остаточні визначення доменів і відношень для цього прикладу представлені на рис.4.31.

Визначення доменів

Номер заліковки	CCDDDD, де С буква українського алфавіту D десятична цифра
Курс	{“C1”, “C2”, “C3”, “C4”, “C5”}
Гуртожиток	CHAR (15)
Плата	DEC (5)

Визначення відношень і ключів

СТУДЕНТ (Номер заліковки, Курс, Гуртожиток)

Ключ: Номер заліковки

ГУРТОЖИТОК_ПЛАТА (Гуртожиток, Плата)

Цей же результат можна отримати, перетворивши відношення з 2 НФ в 3 НФ при видаленні транзитивних залежностей.

Однак, в цьому випадку процес простіше, а результат є більш надійним. Процес простіше тому, що не потрібно знати, що усуваємо транзитивною залежність. Все, що потрібно - це знайти дієві способи перетворення всіх обмеження в логічні слідства визначень доменів і ключів. Результат є більш надійним тому, що при перетворенні відношення в третю нормальну форму знаємо тільки те, що кількість аномалій в ньому менше в порівнянні з другою нормальною формою. Привівши відношення до ДКНФ, точно знаємо, що отримані відношення не будуть мати ніяких аномалій модифікації.

4.4 Синтез відношень

Два атрибути (скажімо, A і B) можуть мати зв'язок трьох видів:

1. Вони можуть визначати один одного: $A \rightarrow B$ і $B \rightarrow A$. В цьому випадку A і B мають атрибутивний зв'язок «один до одного».
2. Один з них може визначати інший: $A \rightarrow B$ або $B \rightarrow A$. В цьому випадку A і B мають атрибутивний зв'язок «багато до одного».
3. Вони можуть бути функціонально не пов'язані: $A \not\rightarrow B$ і $B \not\rightarrow A$. В цьому випадку A і B мають атрибутивний зв'язок «багато до багатьох».

Атрибутивний зв'язок «один до одного»

Якщо A визначає B , а B визначає A , значення атрибутів мають зв'язок «один до одного» (one-to-one relationship). Це повинно бути так, оскільки якщо A визначає B , то між A і B має бути зв'язок «багато до одного». Тим часом, справедливо і інше твердження: якщо B визначає A , то між B і A повинна бути зв'язок «один до одного». Щоб обидва твердження були справедливими одночасно, зв'язок між A і B має в дійсності мати вигляд «один до одного» (що є окремим випадком зв'язку «багато до одного»), а зв'язок між B і A в реальності також має вигляд «один до одного». Тому в даному випадку в наявності зв'язок «один до одного».

Цей випадок можна проілюструвати, наприклад, атрибутами Ідентифікаційний код викладача і Номер паспорта викладача деякого відношення ВИКЛАДАЧ. Кожен з цих атрибутів однозначно визначає викладача. Отже, кожному значенню атрибута Ідентифікаційний код відповідає рівно одне значення атрибута Номер паспорта, і навпаки.

Ці вирази зібрані в першому стовпці на рис. 4.32, а правила визначення записів наведені в урізанні. Якщо A і B мають зв'язок 1: 1, вони можуть перебувати в одному відношенні R . A визначає B , а B визначає A . Ключем відношення може бути або A , або B . Новий атрибут C може бути доданий в відношення R , якщо один з атрибутів A і B функціонально визначає C .

	Тип зв'язку між атрибутами		
Визначення відношення	$R(A,B)$	$S(C,D)$	$T(E,F)$
Залежності	$A \rightarrow B$ $B \rightarrow A$	$C \rightarrow D$ $D \rightarrow C$	$E \rightarrow F$ $F \rightarrow E$
Ключ	Либо A , либо B	C	(E,F)
Правила добавлення атрибута	Либо A , либо $B \rightarrow C$	$C \rightarrow E$	$(E,F) \rightarrow G$

Рис. 4.32. Три типа зв'язку атрибутів

Щодо прикладу з атрибутами Ідентифікаційний код і Номер паспорта можна висловити три еквівалентних твердження:

- Якщо два атрибути функціонально визначають один одного, між їх значеннями є зв'язок «один до одного».
- Якщо два атрибути однозначно визначають одну і ту ж сутність або один і той же об'єкт, між їх значеннями є зв'язок «один до одного».
- Якщо два атрибути мають зв'язок «один до одного», вони функціонально визначають один одного.

Атрибути, що мають зв'язок «один до одного», повинні бути присутніми разом, щонайменше, в одному відношенні, щоб вони могли бути еквівалентними. У загальному випадку, однак, небажано, щоб ці атрибути з'являлися разом більш ніж в одному відношенні, оскільки це призведе до непотрібного дублювання даних. Однак уникнути дублювання повністю неможливо, тому все-таки один з цих атрибутів або обидва вони з'являються в інших відношеннях.

Атрибутивний зв'язок «багато до одного»

Якщо атрибут A визначає B , але B не визначає A , то між їх значеннями є зв'язок «багато до одного» (*many-to-one relationship*). Наприклад, в деякому відношенні КУРАТОР атрибут Номер заліковки студента визначає Табельний номер викладача. Окремо взятий викладач може бути куратором у багатьох студентів, проте кожним студентом має тільки одного куратора. Тому тут є зв'язок «багато до одного».

Застосувати ці твердження до проектування баз даних можна наступним чином: якщо в створюваному щодо А визначає В, то додавати в це відношення можна тільки ті атрибути, які визначаються атрибутом А. Припустимо, наприклад, в відношення під назвою СТУДЕНТ помістили атрибути Номер заліковки і Гуртожиток. У цей показник можна додавати будь-які інші атрибути, які визначаються атрибутом Номер заліковки, наприклад Ім'я студента. Але якщо атрибут Плата визначається атрибутом Гуртожиток, його не можна додати в дане відношення. Атрибут Плата може бути доданий тільки в тому випадку, якщо Номер заліковки → Плата.

Ці твердження наведені в середньому стовпці на рис. 4.32. Якщо С і D мають зв'язок N:1, вони можуть перебувати разом в відношенні. З визначатиме D, але D не визначатиме С. Ключем відношення буде атрибут С. Інший атрибут, Е, може бути доданий в відношення, тільки якщо С визначає Е.

Атрибутивний зв'язок «багато до багатьох»

Якщо А не визначає В, а В не визначає А, то між значеннями цих атрибутів є зв'язок «багато до багатьох» (many-to-many relationship). Для прикладу розглянемо відношення Вивчення, яке включає атрибути Ім'я викладача і Предмет, які мають зв'язок «багато до багатьох». Один викладач веде багато предметів, а один і той же предмет ведеться багатьма викладачами. У зв'язку «багато до багатьох» обидва атрибути повинні бути ключами відношення. В даному прикладі ключем відношення є комбінація (Ім'я викладача, Предмет).

При побудові відношень, у яких ключами є кілька атрибутів, можна додавати нові атрибути, які функціонально залежать від всього ключа. Атрибут Кількість годин, виділених на вивчення предмета, функціонально залежить від кожного з атрибутів в поєднанні (Ім'я викладача, Предмет) і тому може бути доданий в відношення. Атрибут Посада в дане відношення додати не можна, тому що він залежить тільки від атрибута Ім'я викладача, але не від атрибуту Предмет. Якщо потрібно зберігати атрибут Посада в базі даних, його слід додати в відношення, що містить інформацію про професорсько-викладацький склад, а не про вивчення.

Ці висловлювання наведені в колонці праворуч на рис. 4.32. Якщо атрибути Е і F мають зв'язок M: N, то Е не визначає F, а F не визначає Е. Обидва ці атрибута можна помістити в відношення Т, і в цьому випадку ключем відношення Т буде комбінація (Е, F). Новий атрибут G може бути доданий в відношення Т, якщо він визначається обома атрибутами зі сполучення (Е, F). Атрибут G можна додати в відношення Т, якщо він визначається тільки одним з атрибутів в комбінації (Е, F).

Розглянемо аналогічний приклад. Припустимо, хочемо додати в відношення ВИВЧЕННЯ атрибут Номер аудиторії. Чи визначається Номер аудиторії ключем в відношення ВИВЧЕННЯ комбінацією (Ім'я викладача, Предмет)? Швидше за все, ні, оскільки викладач може вести один і той же предмет в різних аудиторіях.

Поєднання (Ім'я викладача, Предмет) і атрибут Номер аудиторії мають зв'язок М: N. Раз це так, можна застосувати правила, наведені на рис. 4.32, але тут роль Е буде грати поєднання (Ім'я викладача, Предмет), а роль F - атрибут Номер аудиторії. Тепер можна побудувати нове відношення Т з атрибутами Ім'я викладача, Предмет і Номер аудиторії. Ключем цього відношення буде комбінація (Ім'я викладача, Предмет, Номер аудиторії). У цій ситуації виходить, що створили нове ставлення з новою темою. Розглянемо відношення Т, що містить відомості про імена викладачів, предметах і номерах аудиторій. Темою цього відношення більше не є ХТО_ЩО_І_ДЕ_ВИКЛАДАЄ.

Зміна теми може бути як доречним, так і недоречним. Якщо Номер аудиторії є значущим атрибутом, тема дійсно повинна бути змінена. У цьому випадку відношення ВИВЧЕННЯ не підходить, і більш доречною є тема ХТО_ЩО_І_ДЕ_ВИКЛАДАЄ.

З іншого боку, в залежності від вимог користувачів, відношення ВИВЧЕННЯ може бути придатним в тому вигляді, в якому воно є. У такому випадку, якщо атрибут Номер аудиторії все ж повинен зберігатися в базі даних, його слід помістити в інше ставлення - наприклад, СЕКЦІЯ_НОМЕР, ПРЕДМЕТ_СЕКЦІЯ або що-небудь на зразок цього.

Багатозначні залежності

Викладені вище міркування, що стосуються атрибутивного зв'язку «багато до багатьох», можуть зробити концепцію багатозначних залежностей більш простий для розуміння. Розглянемо відношення СТУДЕНТ з атрибутами Номер заліковки студента, Науковий проект, Секція. Особливість полягає в тому, що в ньому є дві різних зв'язку виду «багато до багатьох»: одна між атрибутами Номер заліковки і Науковий проект, яка говорить про те, що один студент може працювати з декількома науковими проектами і в одному науковому проекті можуть брати участь кілька студентів; а інша - між атрибутами Номер заліковки і Секція, тобто один студент може відвідувати кілька секцій і одну секцію можуть відвідувати кілька студентів. Ясно, що наукові проекти студента не мають ніякого відношення до секцій, в яких він займається. Однак якщо обидві зв'язку присутні в одному відношенні, виникає враження, що між цими атрибутами існує якась взаємозв'язок.

Атрибути Науковий проект і Секція незалежні, і ніякої проблеми не виникло б, якби студент міг брати участь тільки в одному науковому проекті і займатися тільки в одній секції. Тоді Номер заліковки функціонально визначав би атрибути Науковий проект і Секція, і відношення знаходилося б у ДКНФ. У цьому випадку зв'язок кожного з цих атрибутів з атрибутом Номер заліковки мала б вигляд «багато до одного».

Інший спосіб уявити собі складне становище складається в дослідженні Ключа (Номер заліковки, Науковий проект, секція). Оскільки щодо СТУДЕНТ є зв'язки виду «багато до багатьох», ключ повинен містити в собі кожен з атрибутів. Тепер запитаємо себе: «А яку тему представляє даний ключ?». Можна сказати, що це комбінація про наукові проекти студента і відвідуваних ним секцій. Але така комбінація не одна, їх безліч. Один рядок цього відношення описує тільки частина комбінації і щоб отримати всю картину, потрібні всі рядки, що містять інформацію про конкретного студента. Взагалі кажучи, рядок відношення повинен містити всі дані, що описують окремий екземпляр теми даного відношення.

Вирішення проблеми багатозначних залежностей полягає в тому, щоб розбити вихідне відношення на два нових, кожне з яких буде мати тільки одну тему. Відносно СТУДЕНТ_НАУЧНИЙ_ПРОЕКТ представлені комбінації студентів і наукових проектів. Все, що знаємо про кожну з таких комбінацій, міститься в одному рядку, і не отримаємо будь-якої додаткової інформації про дану комбінації, досліджуваного рядки.

4.5 Правила побудови відношень

Узагальнимо розглянуті правила побудови відносин.

Зв'язок «один до одного»

- Атрибути, що мають зв'язок «один до одного», повинні фігурувати разом, по крайній мірі, в одному відношенні. Нехай відношення носить ім'я А, а його атрибути - А і В.

- Або А, або В повинен бути ключем відношення R.

- Якщо деякий атрибут функціонально визначається атрибутом А чи В, він може бути доданий в відношення R.

- Атрибут, що не може бути визначений функціонально атрибутами А чи В, не може бути доданий в відношення R.

- Атрибути А і В повинні перебувати разом у відношенні R і більше ні в якому іншому відношенні.

- Для подання пари (А, В) у відношеннях, відмінних від R, повинен послідовно використовуватися один з атрибутів, А чи В.

Зв'язок «Багато до одного»

- Атрибути, що мають зв'язок «багато до одного», можуть перебувати разом в одному відношенні. Нехай в відношенні S атрибут С визначає атрибут D.

- Атрибут С повинен бути ключем відношення S.

- Якщо деякий атрибут визначається атрибутом С, він може бути доданий в відношення S.
- Атрибут, що не можуть бути визначений функціонально атрибутом С, не може бути доданий в відношення S.

Зв'язок «багато до багатьох»

- Атрибути, що мають зв'язок «багато до багатьох», можуть співіснувати в одному відношенні. Нехай в відношенні Т є два таких атрибута, Е і F.
- Ключем відношення Т повинна бути комбінація (Е, F).
- Якщо деякий атрибут визначається поєднанням атрибутів (Е, F), він може бути доданий в відношення Т.
- Атрибут, що не можуть бути визначений функціонально поєднанням атрибутів (Е, F), не може бути доданий в відношення Т.
- Якщо додавання нового атрибуту G розширює ключ в відношення до (Е, F, G), це означає, що тема відношення змінилася. Або атрибут G логічно не належить відношенню Т, або слід вибрати інше ім'я для відношення відповідно до зміненої теми.

4.6 Оптимізація

У даній темі розглянута концепція нормалізації, і продемонстровано, як створювати таблиці в доменно-ключовий нормальній формі. Використовувався метод придатний для більшості випадків, однак іноді результат нормалізації не варто витрачених зусиль. Далі розглянемо дві ситуації, в яких це може статися.

Денормалізація

Як вже говорилося, нормалізовані відношення вільні від аномалій модифікації, і з цієї причини вони є кращими, ніж ненормалізовані в відношення. Але якщо судити з інших позицій, іноді нормалізація не варта того, щоб її проводити.

Як приклад розглянемо відношення КАФЕДРА (Назва кафедри, Завідувач, Заступник завідувача). Будемо вважати, що завідувач має трьох заступників. Це відношення не знаходиться в доменно-ключовий нормальній формі, так як обмеження Назва кафедри Завідувач не є логічним наслідком ключа таблиці.

Відношення КАФЕДРА можна нормалізувати, розбивши його на два відношення: ЗАВІДУЮЧИЙ (Назва кафедри, Завідувач) і ЗАМІСНИК_ЗАВІДУЮЧОГО (Назва кафедри, заступник завідувача). Однак тепер, коли з додатком бази даних потрібно буде отримати дані про кафедру, йому₁₄₆

доведеться прочитати мінімум дві, а можливо, і всі чотири рядки даних. В якості альтернативи можна помістити всіх трьох заступників завідувача в таблицю КАФЕДРА, виділивши для кожного індивідуальний атрибут. Отримана таблиця мала б такий вигляд: КАФЕДРА (Назва кафедри, Завідувач, Заступник завідувача 1, Заступник завідувача 2, Заступник завідувача 3).

Відношення КАФЕДРА знаходиться в ДКНФ, оскільки всі його атрибути функціонально залежать від ключа Назва кафедри. Однак тут щось втрачено. Для того щоб побачити, що саме втрачено, припустимо, що хочемо дізнатися назви кафедр, в яких є заступник завідувача по імені Іванов О.Н. Для цього доведеться шукати дане ім'я в кожному з трьох стовпців Заступник завідувача. Запит буде виглядати приблизно так:

```
SELECT Назва кафедри
FROM КАФЕДРА
WHERE Заступник завідувача 1 = 'Іванов А.Н.' OR
Заступник завідувача 2 = 'Іванов А.Н.' OR
Заступник завідувача 3 = 'Іванов А.Н.'
```

Використовуючи нормалізований варіант з відношенням ЗАМІСНИК_ЗАВІДУЮЧОГО, необхідно б було написати лише:

```
SELECT Назва кафедри
FROM ЗАМІСНИК_ЗАВІДУЮЧОГО
WHERE Заступник завідувача = 'Іванов О.Н',
```

У цьому прикладі дані три можливих рішення, у кожного з яких є свої переваги і недоліки. У таких випадках вибір рішень залежить від структури, смислового значення і вимог даних предметної області, а також від їх обсягу: не існує твердого та конкретного правила, що вказує, який з варіантів є найкращим, вибір залежить від характеристик прикладних програм, що використовують цю базу даних.

Загалом, іноді відношення навмисно залишають в ненормалізованому вигляді або нормалізують, а потім денормалізують. Найчастіше це робиться для підвищення продуктивності завжди, коли необхідно комбінувати дані з двох різних взаємини, СУБД повинна виконувати додаткову роботу.

Навмисна (контрольована) надмірність

Одним з переваг нормалізованих відношень є те, що в них мінімізується дублювання даних (тільки значення ключів з'являються більш ніж в одному відношенні). Але в цілях підвищення продуктивності іноді доречним є умисне дублювання даних. *Для гарантії цілісності даних при введенні контрольованої надмірності використовуються програмні засоби СУБД або навіть конкретних прикладних програм.*

Розглянемо, наприклад, додаток, що обробляє замовлення в книжковому магазині, яке звертається до відношення КНИГА, що має такі атрибути:

Код книги
Назва книги
Видавництво
Опис книги
Кількість в наявності
Замовлену кількість
Стандартна ціна
Ім'я покупця

Припустимо, що атрибут Код книги є ключем і що відношення знаходиться в ДКНФ. Припустимо також, що атрибут Опис книги - це потенційно довге текстове поле.

Додатком, що обробляє замовлення, необхідно буде звернутися до цього відношення, щоб отримати значення атрибутів Назва книги, Видавництво, Стандартна ціна і Кількість в наявності. Припустимо, що з додатком не потрібно Опис книги. Залежно від характеристик використовуваної СУБД, можливо, що присутність цього атрибута значно сповільнить обробку. У цій ситуації проектувальники бази даних можуть вирішити продублювати деякі дані в другому відношенні, яке буде містити тільки ту інформацію, яка потрібна для обробки замовлення. Наприклад, можна створити відношення під назвою ЗАМОВЛЕННЯ_КНИГА (Код книги, Назва книги, Видавництво, Стандартна ціна, Кількість в наявності), яке буде використовуватися тільки додатком, що обробляє замовлення.

В цьому випадку проектувальники створюють потенційне джерело серйозних проблем з цілісністю даних. Їм доведеться розробити процедури як програмного, так і ручного контролю, щоб гарантувати, що таких проблем не виникне. Тому до подібного рішення вони вдадуться тільки в тому випадку, якщо, на їхню підвищення, підвищення продуктивності окупить витрати, пов'язані з додатковим контролем, і ризик виникнення проблем з цілісністю даних.

Ще одна причина для введення навмисної надмірності - це створення таблиць, призначених виключно для створення звітів і підтримки прийняття рішень.

Питання:

- 4.1. Дайте визначення поняттю нормалізація і приведіть класифікацію нормальних форм.
- 4.2. Поясніть суть першої нормальної форми і приведіть властиві аномалії. Наведіть приклад.
- 4.3. Сформулюйте поняття функціональної залежності. Що називається детермінантом функціональної залежності?
- 4.4. Дайте визначення другій нормальній формі. Наведіть приклад.
- 4.5. Як перейти від другої нормальної форми до третьої. Наведіть приклад.

4.6. Опишіть призначення нормальної форми Бойса-Кодда. Наведіть приклад.

4.7. Сформулюйте поняття багатозначної залежності і покажіть, як воно пов'язане з четвертою нормальною формою. Наведіть приклад.

4.8. Сформулюйте поняття залежності з'єднання і покажіть, як воно пов'язане з п'ятою нормальною формою. Наведіть приклад.

4.9. Поясніть суть доменно-ключовою нормальної форми.

4.10. Опишіть правила побудови відношень

4.11. В чому полягає суть оптимізації. Поясніть на прикладах.

ТЕМА 5

Методологія проектування баз даних

5.1 Введення в методологію проектування баз даних

Перш ніж приступити до розгляду власне методології, корисно дізнатися, що вона собою являє, зокрема те, як методологія концептуального і логічного проектування бази даних співвідносяться з фізичним проектуванням.

Що таке методологія проектування

Методологія проектування - *структурований підхід, який передбачає використання спеціалізованих процедур, технічних прийомів, інструментів, документації і націлений на підтримку і спрощення процесу проектування.*

Методологія проектування передбачає розбиття всього процесу на декілька фаз, кожна з яких, в свою чергу, складається з декількох етапів. На кожному етапі розробнику пропонується набір технічних прийомів, що дозволяють вирішувати завдання, які стоять перед ним на даній стадії розробки. Крім того, методологія пропонує методи планування, координації, управління, оцінки ходу розробки проекту, а також структурований підхід до аналізу і моделювання всього набору вимог, що висуваються до бази даних і дозволяє виконати ці дії стандартизованим і організованим чином.

Концептуальне, логічне і фізичне проектування бази даних

У запропонованій методології проектування баз даних весь процес розробки розділяється на три основні фази: концептуальне, логічне і фізичне проектування.

Концептуальне проектування бази даних - *процедура конструювання інформаційної моделі підприємства, що не залежить від будь-яких фізичних умов реалізації.*

Фаза концептуального проектування бази даних починається зі створення концептуальної моделі даних підприємства, повністю незалежною від будь-яких деталей реалізації. До останніх відносяться: обраний тип СУБД, склад програм додатку, використовувана мова програмування, конкретна обчислювальна платформа і будь-які інші фізичні особливості реалізації.

Логічне проектування бази даних - *процес конструювання інформаційної моделі підприємства на основі існуючих конкретних моделей даних, що не залежать від використовуваної СУБД і інших фізичних умов реалізації.*

Фаза логічного проектування бази даних полягає в перетворенні концептуальної моделі даних в логічну модель даних підприємства з урахуванням обраного типу СУБД (наприклад, передбачається використання деякої реляційної СУБД). Логічна модель даних є джерелом інформації для фази фізичного проектування. Вона надає розробнику фізичної моделі даних засоби проведення всебічного аналізу різних аспектів роботи з даними, що має

виключно важливе значення для вибору дійсно ефективного проектного рішення.

Фізичне проектування бази даних - процес створення опису конкретної реалізації бази даних, що розміщується у вторинній пам'яті. Передбачає опис структури зберігання даних і методів доступу, призначених для здійснення найбільш ефективного доступу до інформації.

Фаза фізичного проектування бази даних передбачає прийняття розробником остаточного рішення про способи реалізації створюваної бази. Тому фізичне проектування обов'язково проводиться з урахуванням всіх особливостей використовуваної СУБД. Між фазами фізичного і логічного проектування завжди є певний зворотній зв'язок, оскільки рішення, прийняті на етапі фізичного проектування з метою підвищення продуктивності системи, що розробляється, може вимагати певного перегляду логічної моделі даних.

Найважливіші фактори успішного завершення проектування бази даних

Нижче наведені деякі рекомендації, слідувати яким необхідно для успішного завершення процедури проектування бази даних:

- Підтримка постійного та активного зв'язку з майбутніми користувачами додатку.
- При проведенні процедур моделювання даних дотримуватися рекомендацій, наведених під час обговорення пропонованої методології.
- Розробляти систему виходячи з існуючих характеристик даних.
- Створювати модель даних з урахуванням вимог підтримки їх структурної цілісності і узгодженості.
- Доповнювати пропоновані даною методологією процедури технологічними прийомами концептуалізації, нормалізації і перевірки цілісності транзакцій.
- Для представлення моделі даних якомога ширше використовувати діаграми.
- Для опису додаткових семантичних вимог до даних використовувати засоби мови DBDL (Database Design Language).
- В доповнення до діаграм моделей даних розробити словник опису даних.
- Без коливань повертатися до вже виконаних раніше етапів, якщо це потрібно для досягнення оптимальних результатів.

5.2 Загальний огляд процедури проектування бази даних

В цілому, процедура розробки включає наступні етапи:

Концептуальне проектування бази даних

Етап 1. Створення локальної концептуальної моделі даних виходячи з представлень про предметну область кожного з типів користувачів.

Етап 1.1. Визначення типів сутностей.

Етап 1.2. Визначення типів зв'язків.

Етап 1.3. Визначення атрибутів і пов'язання їх з типами сутностей і зв'язків.

Етап 1.4. Визначення доменів атрибутів.

Етап 1.5. Визначення атрибутів, які є потенційними і первинними ключами.

Етап 1.6. Спеціалізація або генералізація типів сутностей (необов'язковий етап).

Етап 1.7. Створення діаграми "сутність-зв'язок".

Етап 1.8. Обговорення локальних концептуальних моделей даних з кінцевими користувачами.

Логічне проектування бази даних (для реляційної моделі)

Етап 2. Побудова і перевірка локальної логічної моделі даних на основі представлень про предметну область кожного з типів користувачів.

Етап 2.1. Перетворення локальної концептуальної моделі даних в локальну логічну модель.

Етап 2.2. Визначення набору відношень виходячи зі структури локальної логічної моделі даних.

Етап 2.3. Перевірка моделі за допомогою правил нормалізації.

Етап 2.4. Перевірка моделі щодо транзакцій користувачів.

Етап 2.5. Створення діаграм "сутність-зв'язок".

Етап 2.6. Визначення вимог підтримки цілісності даних.

Етап 2.7. Обговорення розроблених локальних логічних моделей даних з кінцевими користувачами.

Етап 3. Створення і перевірка глобальної логічної моделі даних.

Етап 3.1. Злиття локальних логічних моделей даних в єдину глобальну модель даних.

Етап 3.2. Перевірка глобальної логічної моделі даних.

Етап 3.3. Перевірка можливостей розширення моделі в майбутньому.

Етап 3.4. Створення остаточного варіанту діаграми "сутність-зв'язок".

Етап 3.5. Обговорення глобальної логічної моделі даних з користувачами.

Фізичне проектування бази даних (з використанням реляційної СУБД)

Етап 4. Перенесення глобальної логічної моделі даних в середовище цільової СУБД.

Етап 4.1. Проектування основних відношень.

Етап 4.2. Розробка способів отримання похідних даних.

Етап 4.3. Реалізація обмежень предметної області.

Етап 5. Проектування фізичного представлення бази даних.

Етап 5.1. Аналіз транзакцій.

Етап 5.2. Вибір файлової структури.

Етап 5.3. Визначення індексів.

Етап 5.4. Оцінка потреби в дисковому просторі.

Етап 6. Проектування користувацьких представлень.

Етап 7. Проектування засобів захисту.

Концептуальне та логічне проектування бази даних включає три основних етапи. Завданням *першого етапу* є розбиття проекту на групу відносно невеликих (і більш простих) завдань виходячи з представлень про предметну область додатку, властивих кожному з типів кінцевих користувачів. Результатом виконання цього етапу є створення локальних концептуальних моделей даних, що представляють собою повне і точне відображення представлень про предметну область додатку окремих типів користувачів.

На *другому етапі* локальні концептуальні моделі даних перетворюються в локальні логічні моделі даних (для реляційної моделі даних). На цьому етапі з

моделей даних видаляються небажані елементи, що утрудняють реалізацію моделей даних в середовищі СУБД. Потім коректність логічних моделей даних перевіряється за допомогою правил нормалізації. Нормалізація являє собою ефективний засіб, що дозволяє переконатися в структурній узгодженості, логічній цілісності і мінімальній надмірності прийнятої моделі даних. Додатково модель даних перевіряється з метою виявлення можливості здійснення транзакцій, які будуть виконуватися користувачами створюваного додатку. Всі ці перевірки дозволяють отримати необхідну впевненість в тому, що прийнята модель даних є цілком коректною. Локальні логічні моделі даних при необхідності можна використовувати для генерації прототипів реалізацій бази даних, призначених для окремих типів користувачів додатку.

На *третьому етапі* виконується інтеграція локальних логічних моделей даних (що відбивають представлення про предметну область окремих типів користувачів) в єдину глобальну логічну модель даних всього підприємства (узагальнююче представлення про предметну область всіх типів користувачів).

У запропонованій методології проектування істотна роль відводиться кінцевим користувачам, які постійно залучаються розробниками для ознайомлення і перевірки створюваних моделей даних та супровідної документації. Проектування баз даних зазвичай є ітеративний процес, який має конкретну точку початку і включає необмежену кількість циклів поліпшень і доробок. Хоча в нашому викладі цей процес виглядає як чітко визначена послідовність процедур, слід особливо підкреслити, що це ні в якій мірі не означає, що розробка бази даних неодмінно повинна виконуватися саме в такій манері. Досить імовірно, що відомості, отримані при виконанні деякого етапу, можуть зажадати змінити рішення, прийняті на одному з попередніх етапів. Тому вважаємо, що практика попереднього аналізу можливих результатів виконання наступних етапів може виявитися корисною при виконанні початкових етапів розробки. Пропоновану методологію слід розглядати як загальну схему, яка дозволить підвищити загальну ефективність роботи з проектування баз даних.

5.3 Методологія концептуального проектування бази даних

В даному розділі буде розглянуто поетапне інструкцію з концептуального проектування баз даних.

Етап 1. Створення локальної концептуальної моделі даних на основі представлення про предметну область кожного з типів користувачів

Мета: створення локальної концептуальної моделі даних підприємства на основі представлення про предметну область кожного окремого типу користувачів.

Перший етап проектування бази даних полягає в розробці концептуальних моделей даних для кожного з існуючих типів користувачів створюваного додатку. Представлення користувача включає в себе дані, необхідні конкретному користувачеві для прийняття рішень або виконання деякого завдання. Зазвичай представлення користувача відображає деяку функціональну область в загальній області діяльності підприємства - наприклад, виробництво,

маркетинг, збут, управління кадрами або складський облік. Користувач може бути як окремим працівником, так і групою осіб, які будуть безпосередньо працювати з створюваним додатком. В альтернативному випадку поняття "користувач" може представляти генерований системою звіт або навіть запит, пов'язаний з отриманням результатів транзакції. Суть в тому, що в будь-якому випадку виконання необхідних користувачеві дій повинне забезпечуватися створюваною системою.

Визначити характеристики представлень користувачів можна за допомогою різних методів. Почати слід з вивчення діаграм потоків даних, які до цього моменту вже повинні бути створені. Вивчення цих діаграм дозволить встановити функціональні області і, можливо, окремі функції. Потім рекомендується провести опитування потенційних користувачів, вивчити ділові процедури, існуючі звіти і форми і/або провести обстеження роботи підприємства.

Для деякого представлення **локальною концептуальною моделлю** даних називатимемо *концептуальну модель даних, яка відображає представлення про предметну область додатку відповідного користувача*. Кожна локальна концептуальна модель даних включає наступне:

- типи сутностей;
- типи зв'язків;
- атрибути;
- домени атрибутів;
- потенційні ключі;
- первинні ключі.

Концептуальна модель даних доповнюється документацією, створюваною в процесі розробки цієї моделі. На першому етапі розробки повинні бути виконані наступні завдання.

Етап 1.1. Визначення типів сутностей.

Етап 1.2. Визначення типів зв'язків.

Етап 1.3. Визначення атрибутів і пов'язання їх з типами сутностей і зв'язків.

Етап 1.4. Визначення доменів атрибутів.

Етап 1.5. Визначення атрибутів, які є потенційними і первинними ключами.

Етап 1.6. Спеціалізація або генералізація типів сутностей (Необов'язковий етап).

Етап 1.7. Створення діаграми "сутність-зв'язок".

Етап 1.8. Обговорення локальних концептуальних моделей даних з користувачами.

Етап 1.1. Визначення типів сутностей

Мета: визначення основних типів сутностей, присутніх в представленні даного користувача про предметну область додатку.

Перший етап в побудові локальної концептуальної моделі даних полягає у визначенні основних об'єктів, які можуть цікавити користувача. Ці об'єкти є типами сутностей, що входять в модель. Один з методів ідентифікації сутностей

полягає у вивченні специфікацій по виконанню конкретних функцій користувача на даному підприємстві. З цих специфікацій слід вилучити всі використовувані в них іменники або поєднання іменника і прикметника (наприклад, "табельний номер", "прізвище викладача", "код кафедри", "назва кафедри", "телефон кафедри"). Потім серед них вибираються найбільші об'єкти (люди, міста) або ті, які представляють інтерес концепції і виключаються всі іменники, які просто визначають інші об'єкти. Наприклад, властивості "табельний номер" і "прізвище викладача" можуть бути об'єднані в зведеному об'єкті під назвою "викладач", тоді як властивості "код кафедри", "назва кафедри" і "телефон кафедри" можна об'єднати в сутності під назвою "кафедра".

Альтернативний спосіб ідентифікації сутностей полягає в пошуку об'єктів, які існують незалежно від інших. Наприклад, об'єкт "викладач" безумовно, є сутністю, тому що будь-який викладач існує незалежно від того, чи відомо його ім'я, адреса і номер телефону чи ні. У цій роботі істотну допомогу можуть надати користувачі створюваного додатка.

У деяких випадках виділення сутностей буває утруднено через спосіб, за допомогою якого вони представлені в специфікаціях. Найчастіше користувачі, викладаючи свої думки, використовують приклади або аналогії. Замість того, щоб вести розмову про деякого узагальненого викладача, вони можуть просто згадати одне або кілька імен. Буває також, що користувачі уточнюють викладачів, приписуючи предмет їх викладання. Наприклад, "викладач математики", "викладач інформатики" і т.д.

Щоб ще більше заплутати стан справ, користувачі часто використовують синоніми або омоніми. Синонімами називаються слова, подібні за змістом, але різні за звучанням і написанням. Омоніми - це слова, однакові за написанням і звучанням, але мають різні смислові значення, причому реальне значення в кожному конкретному випадку можна встановити тільки по контексту. Так, слово "програма" може позначати курс навчання, майбутню серію послідовних подій, план майбутньої роботи і навіть послідовність телепередач.

Далеко не завжди очевидно те, чим є певний об'єкт - сутністю, зв'язком або атрибутом. Наприклад, як слід класифікувати сімейний шлюб? На практиці шлюб можна цілком обґрунтовано віднести до будь-якої зі згаданих категорій. Аналіз є суб'єктивним процесом, тому різні розробники можуть створювати різні, але цілком припустимі інтерпретації одного і того ж факту.

Вибір варіанту в значній мірі залежить від здорового глузду і досвіду виконавця. Розробники баз даних повинні обмежити предметну область рамками того погляду на світ і існуючі в ньому категорії, які задаються контекстом підприємства і створюваної для нього програми. Досить імовірно, що окремої специфікації з набором вимог замовника може виявитися недостатньо для виділення деякого унікального набору сутностей. Однак серія ітеративних процедур аналізу всього комплексу специфікацій на проект, безумовно, дозволить визначити весь набір сутностей, необхідних для задоволення вимог до системи.

Після виділення кожної сутності їй слід привласнити деяке осмислене ім'я, яке обов'язково має бути зрозуміле користувачам. Обране ім'я і опис сутності поміщається в словник даних. Якщо це можливо, слід встановити і внести в документацію очікувану кількість примірників кожної сутності. Якщо сутність відома користувачам під різними іменами, всі додаткові імена рекомендується визначити як синоніми і також занести в словник даних.

Етап 1.2. Визначення типів зв'язків

Мета: визначення найважливіших типів зв'язків, що існують між сутностями, виділеними на попередньому етапі.

Після виділення сутностей наступним етапом розробки буде встановлення всіх існуючих між ними зв'язків. Одним з методів визначення сутностей є вибірка всіх іменників, присутніх в специфікаціях на проект. Аналогічний підхід можна використовувати і при визначенні існуючих зв'язків, проте в цьому випадку вибираються всі вирази, в яких містяться дієслова. Наприклад:

- Кафедра *має* викладачів.
- Викладачі *навчають* студентів.
- Студенти *вивчають* ряд предметів.

Той факт, що текст специфікацій містить інформацію про деякі зв'язки дозволяє припустити, що ці зв'язки є дуже важливими для підприємства. Тому вони обов'язково повинні бути відображені в створюваній моделі.

Інтерес представляють тільки ті зв'язки між сутностями, які необхідні для задоволення вимог до проекту. Ті зв'язки, які цілком припустимі, але в специфікаціях на проект немає жодної вказівки на те, що вони повинні бути відображені в моделі, не використовуються.

У більшості випадків зв'язки є парними - іншими словами, зв'язки існують тільки між двома сутностями. Однак слід проявляти обережність і ретельно перевіряти наявність в проекті комплексних зв'язків, що об'єднують понад дві сутності різних типів, а також рекурсивних зв'язків, що існують між сутностями одного і того ж типу.

Особливу увагу слід приділяти перевірці того, чи були виділені *всі* зв'язки, явно або неявно присутні в специфікаціях на проект. В принципі, кожному з можливих пар сутностей було б корисно перевірити на наявність між ними деякого зв'язку, однак у великих системах, що включають сотні типів сутностей, ця задача може виявитися надзвичайно трудомісткою. Але взагалі відмовлятися від виконання подібних перевірок нерозумно, до того ж відповідальність за сумні наслідки цієї відмови доведеться нести як аналітикам, так і проектувальникам. Так чи інакше, всі пропущені зв'язки будуть обов'язково виявлені пізніше, при проведенні перевірки можливості виконання транзакцій, необхідних користувачам (етап 2.4).

Визначення кардинальності зв'язків і обмежень, що накладаються на його учасників

Встановивши зв'язки, які матимуть місце в створюваній моделі, необхідно визначити кардинальність кожного з них. Кожен зв'язок може мати кардинальність або "один до одного" (1: 1), або "один до багатьох" (1: M), або "багато до багатьох" (N: M). Якщо відомі конкретні значення кардинальності або

хоча б верхню чи нижню межу цих значень, то цю інформацію обов'язково потрібно зафіксувати в документації. Крім того, слід проаналізувати ступінь участі кожної з сутностей в конкретному типі зв'язку. Ступінь участі може бути або повною (тотальною), або приватною.

Модель, що включає відомості про кардинальності зв'язків і ступені участі в ній сторін, більш наочно відображає сенс встановлених зв'язків. Крім того, кардинальність зв'язку та ступінь участі в ній сторін представляють собою певні обмеження, які використовуються для перевірки і підтримки якості даних в базі. Ці обмеження є тими твердженнями про властивості екземплярів сутностей, які можуть бути перевірені при зміні даних в базі з метою визначення того, чи викличуть ці зміни порушення встановлених правил.

Документування типів зв'язків

Після визначення окремих типів зв'язків їм присвоюються осмислені імена, які повинні бути зрозумілі користувачам. Крім того, рекомендується поміщати в словник даних розгорнутий опис кожного зв'язку, що включає відомості про кардинальності і ступеня участі її членів.

Використання коштів ER-модельовання

Робота істотно спрощується, якщо складна система крім великого текстового опису має і деяке візуальне представлення. Для подання сутностей і зв'язків між ними зазвичай використовуються діаграми "сутність-зв'язок" (ER-діаграми). Рекомендується найширше використання ER-діаграм на всіх етапах проектування бази даних. Це дозволить завжди мати під рукою наочний образ частини підприємства, яка моделюється.

Етап 1.3. Визначення атрибутів і зв'язування їх з типами сутностей і зв'язків

Мета: зв'язування атрибутів з відповідними типами сутностей або зв'язків.

На наступному етапі пропонованої методології необхідно виявити всі дані, що описують сутності і зв'язки, виділені в створюваній моделі бази даних. Скористаємося тим же методом, який застосовувався для ідентифікації сутностей: виберемо все іменники і фрази, в яких вони містяться, присутні в специфікаціях на проект. Обраний іменник являє атрибут в тому випадку, якщо він описує властивість, якість, ідентифікатор або характеристику деякої сутності або зв'язку.

Найпростіший метод виділення атрибутів - після ідентифікації чергової сутності або зв'язку в деякій специфікації поставити собі наступне запитання: "Яку інформацію потрібно зберігати про ...". Відповідь на це питання треба шукати в тексті специфікації. У деяких випадках може виявитися корисним попросити користувачів уточнити їхні вимоги. На жаль, користувачі часто дають відповіді, які містять надлишкові концепції, тому кожна отримана відповідь користувача підлягає найбільш суворому аналізу.

Атрибути прості і складені

Важливо відзначити, що кожен атрибут може бути або простим, або складеним. Складені атрибути представляють собою набір простих атрибутів. Наприклад, атрибут "Адреса" може бути простим і представляти всі елементи адреси як єдине значення: "Київ, Малиновська, 7, 23". В іншому варіанті цей же

атрибут може бути представлений як складений, тобто що складається з серії простих атрибутів, що містять різні елементи адреси. У цьому випадку те ж саме значення може бути розділене на такі атрибути, як "Місто" (Київ), "Вулиця" (Малиновська), "Дім" (7), "Квартира" (23). Вибір способу представлення адреси у вигляді простого або складеного атрибута визначається вимогами, що пред'являються до додатку користувачем. Якщо користувач не потребує доступу до окремих елементів адреси, то його доцільно представити як простий атрибут. Але якщо користувачеві буде потрібно незалежний доступ до окремих елементів адреси, то атрибут "Адреса" слід зробити складеним, утвореним з необхідної кількості простих атрибутів.

На даному етапі важливо ідентифікувати всі прості атрибути, які повинні бути представлені в концептуальній моделі бази даних, включаючи і ті, які згодом будуть використані для створення складених атрибутів.

Похідні атрибути

Атрибути, значення яких можуть бути встановлені за допомогою значень інших атрибутів, називаються похідними, або обчислюваними. Прикладами похідних атрибутів є наступні:

- кількість викладачів даного факультету;
- вік викладача;
- загальна сума зарплати всього викладацького складу факультету.

Дуже часто подібні атрибути взагалі не відображаються в концептуальній моделі даних. Однак в деяких випадках може мати місце ризик видалення або модифікації атрибута або атрибутів, значення яких використовуються для обчислення значення похідного атрибута. В цьому випадку похідний атрибут повинен бути представлений в моделі даних, що дозволить попередити про небажану втрату інформації. Однак якщо похідний атрибут показаний в моделі даних, слід неодмінно вказати, що він є саме похідним. Спосіб представлення похідних атрибутів встановлюється на етапі фізичного проектування бази даних. Залежно від того, як цей реквізит застосовується, нове значення похідного реквізиту може обчислюватися або при кожному зверненні до нього, або тільки при зміні значень атрибутів, використовуваних для його розрахунку. Однак ці питання не беруться до уваги на етапі концептуального проектування, тому їх обговорення буде розглянуто нижче в розділі "Методологія фізичного проектування реляційних баз даних".

При визначенні використовуваних в деякому додатку атрибутів дуже часто виявляється, що на попередніх етапах одна або більше сутностей були пропущені. В цьому випадку слід повернутися до вже виконаних етапів і документально оформити знову виявлені сутності, після чого проаналізувати зв'язки, в яких вони беруть участь.

Може виявитися корисним підготувати список всіх атрибутів, використовуваних в специфікаціях на проект. У міру зв'язування чергового атрибута з деякою сутністю або зв'язком, він викреслюється зі списку. Подібний метод дозволяє гарантувати, що кожен з атрибутів буде пов'язаний з сутністю або зв'язком тільки одного типу. Коли зі списку буде викреслено останній атрибут, все ідентифіковані моделі атрибутів виявляться пов'язаними з деякою сутністю або зв'язком.

Слід мати на увазі, що в певних випадках створюється враження, що деякі атрибути повинні бути пов'язані з сутностями кількох різних типів. Подібна ситуація виникає в наступних випадках:

1. Ідентифіковано кілька сутностей (наприклад, "Доцент", "Доктор наук" і "Професор"), які, по суті, являють екземпляри однієї і тієї ж узагальненої сутності (наприклад, "Викладач"). В цьому випадку слід обмірковувати, чи можна об'єднати всі ці сутності в єдину сутність або варто зберегти їх як незалежні сутності, що відображають в додатку різні ролі окремих екземплярів більше загальної сутності. Питання спеціалізації і об'єднання сутностей будуть детально розглянуті під час обговорення етапу 1.6.

2. Виявлений новий зв'язок між сутностями різних типів. В цьому випадку атрибут слід пов'язати з якоюсь однією сутністю, яка буде називатися батьківською, після чого виконати визначення зв'язку, як зазначено в описі етапу 1.2, включаючи поповнення документації детальними даними про новий зв'язок.

Документування атрибутів

Кожному виявленому атрибуту слід привласнити осмислене ім'я, зрозуміле користувачам. Про кожний атрибут в документацію поміщаються такі відомості:

- ім'я атрибута і його опис;
- будь-які аліаси, або синоніми, наявні для даного атрибута;
- тип даних і розмірність значення;
- значення, яке приймається для атрибута за замовчуванням (якщо таке є);
- чи є атрибут обов'язковим (тобто чи може він бути відсутнім або мати значення NULL);
- чи є атрибут складовим і, якщо це так, з яких простих атрибутів він складається;
- чи є даний атрибут похідним і, якщо це так, який метод слід використовувати для обчислення його значення;
- чи є даний атрибут множинним.

Етап 1.4. Визначення доменів атрибутів

Мета: визначення доменів для всіх атрибутів, присутніх в кожній локальній концептуальній моделі даних.

Завдання цього етапу побудови локальної концептуальної моделі даних полягає у визначенні доменів атрибутів для всіх атрибутів, присутніх в моделі. **Доменом** називається деякий пул значень, елементи якого вибираються для присвоєння значень одному або більше атрибутів. Нижче наведено кілька прикладів доменів.

- Домен атрибута, що включає допустимі значення номерів телефонів і факсів. Він складається з рядків довжиною в 13 цифрових символів.
- Припустимими значеннями для атрибута "Стать" сутності "Студент" є "Ч" і "Ж". Домен цього атрибута складається з двох рядків довжиною в один символ, що мають вказані значення.

Повністю розроблена модель даних повинна включати домени для кожного з присутніх в ній атрибутів. Домени повинні містити наступні дані:

- набір допустимих значень для атрибута;
- відомості про розмір і форматі кожного з полів атрибутів.

У доменах може бути вказана й інша додаткова інформація - наприклад, відомості про допустимі операції зі значеннями атрибута, а також дані про те, які атрибути можна використовувати для порівняння з іншими атрибутами або при побудові комбінацій з кількох атрибутів. Однак методологія визначення характеристик доменів атрибутів все ще є предметом досліджень.

Документування доменів атрибутів

Після визначення доменів атрибутів їх імена і характеристики поміщаються в словник даних. Одночасно оновлюються записи словника даних, що відносяться до атрибутів, - в них заносяться імена призначених кожному атрибуту доменів.

Етап 1.5. Визначення атрибутів, які є потенційними і первинними ключами

Мета: визначення всіх потенційних ключів для кожного типу сутності і, якщо таких ключів виявиться кілька, вибір серед них первинного ключа.

Визначення потенційних ключів і вибір первинного ключа

На цьому етапі для кожної сутності встановлюється потенційний ключ (або ключі), після чого здійснюється вибір первинного ключа. **Потенційним ключем називається атрибут або мінімальний набір атрибутів заданої сутності, що дозволяє унікальним чином ідентифікувати кожен її екземпляр.** Для деяких сутностей можлива наявність декількох потенційних ключів. У цьому випадку серед них потрібно вибрати один ключ, який буде називатися **первинним ключем**. Решта всіх потенційних ключів будуть називатися **альтернативними ключами**. При виборі первинного ключа серед кількох потенційних керуйтеся наведеними нижче рекомендаціями.

- Використовується потенційний ключ з мінімальним набором атрибутів.
- Використовується той потенційний ключ, ймовірність зміни значень якого мінімальна.
- Вибирається той потенційний ключ, який має мінімальну ймовірність втрати унікальності значень в майбутньому.
- Використовується потенційний ключ, значення якого мають мінімальну довжину (в разі текстових атрибутів).
- Зупиніть свій вибір на потенційному ключі, з яким буде найпростіше працювати (з точки зору користувача).

У процесі визначення первинного ключа встановлюється, чи є дана сутність сильною або слабкою. *Якщо вибрати первинний ключ для даної сутності виявилось можливим, то таку сутність прийнято називати **сильною**.* І навпаки, *якщо вибрати первинний ключ для заданої сутності неможливо, то її називають **слабкою**.* Таким чином, первинний ключ слабкої сутності можна буде визначити тільки після відображення цієї слабкої сутності та її зв'язку з сутністю-власником у відношення, в якому згаданий зв'язок моделюється за допомогою поміщення в дане відношення відповідного зовнішнього ключа. Процедура відображення сутностей і їх зв'язків у відносини буде розглянута при обговоренні етапу 2.2. Тому визначення первинних ключів

для слабких сутностей може бути виконано тільки після досягнення зазначеного етапу розробки.

Документування первинних і альтернативних ключів

Після вибору первинних і альтернативних (якщо вони існують) ключів сутностей відомості про них необхідно помістити в словник даних.

Етап 1.6. Спеціалізація або генералізація типів сутностей (необов'язковий етап)

Мета: Визначення суперкласів і підкласів для типів сутностей (якщо це необхідно).

На цьому етапі при необхідності можна продовжити розробку ER-моделі, використовуючи процедури спеціалізації або генералізації по відношенню до сутностей, виділених на етапі 1.1. При проведенні спеціалізації робиться спроба виділити відмінності шляхом визначення одного або більше **підкласів** деякої сутності, яка в цьому випадку називається **суперкласом спеціалізації**. При проведенні генералізації робиться спроба виділити загальні властивості деяких сутностей шляхом визначення узагальнюючої сутності, званої **суперкласом генералізації**.

Як приклад розглянемо сутності Книги (читацький зал) і Книги (видача на руки), представлені на рис. 4.1 а. Має бути прийнято рішення, чи є необхідність генералізувати ці сутності в підкласи узагальнюючого суперкласу, представленого сутністю Книги, або ж доцільніше залишити їх незалежними сутностями. Обидві ці сутності мають кілька однакових атрибутів: Назва книги, Видавництво, Кількість сторінок, а також однаковий первинний ключ Код книги. Однак сутність Книги (видача на руки) має ще додатковий атрибут - Термін здачі. Крім того, відзначимо, що обидві розглянуті сутності мають загальний зв'язок Читач_Книга. Однак сутність Книги (читацький зал) має і власну унікальну зв'язок - Зал_Книга.

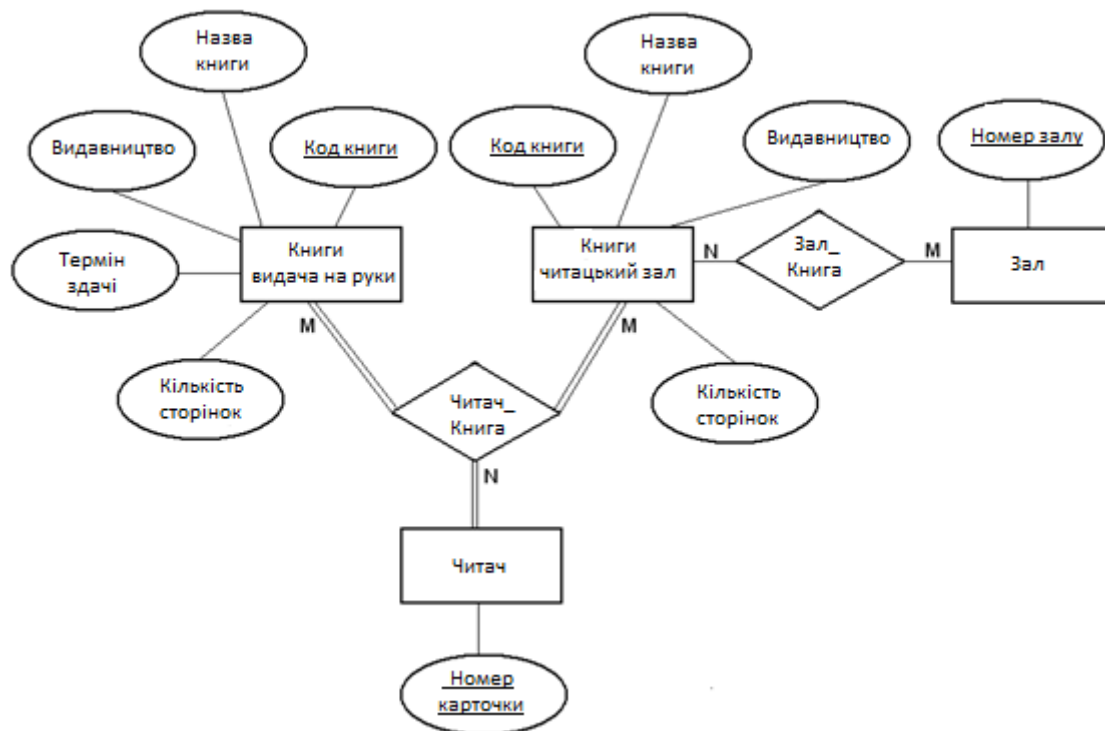
Приймаємо рішення генералізувати сутність Книги (читацький зал) і Книги (видача на руки) виходячи з існуючої між ними спільності атрибутів і зв'язків. В результаті сутність Книги (читацький зал) і Книги (видача на руки) тепер будуть розглядатися як окремі підкласи суперкласу Книги, структура якого показана на рис. 4.1 б. Зв'язок, який суперклас Книги має зі своїми підкласами, є повним і непересічним, оскільки кожен член суперкласу Книги повинен бути членом одного з підкласів (Книги (читацький зал) і Книги (видача на руки)), але ніколи не може належати відразу обом підкласам. Це подання є досить корисним для відображення загальних атрибутів і зв'язків, що відносяться до окремих підкласів.

Немає ніяких чітких рекомендацій з приводу того, в яких випадках ER-модель слід додатково розширювати за допомогою механізмів спеціалізації і генералізації. В кожному окремому випадку прийняте рішення буде чисто суб'єктивним і має прийматися з урахуванням конкретних особливостей модельованої предметної області.

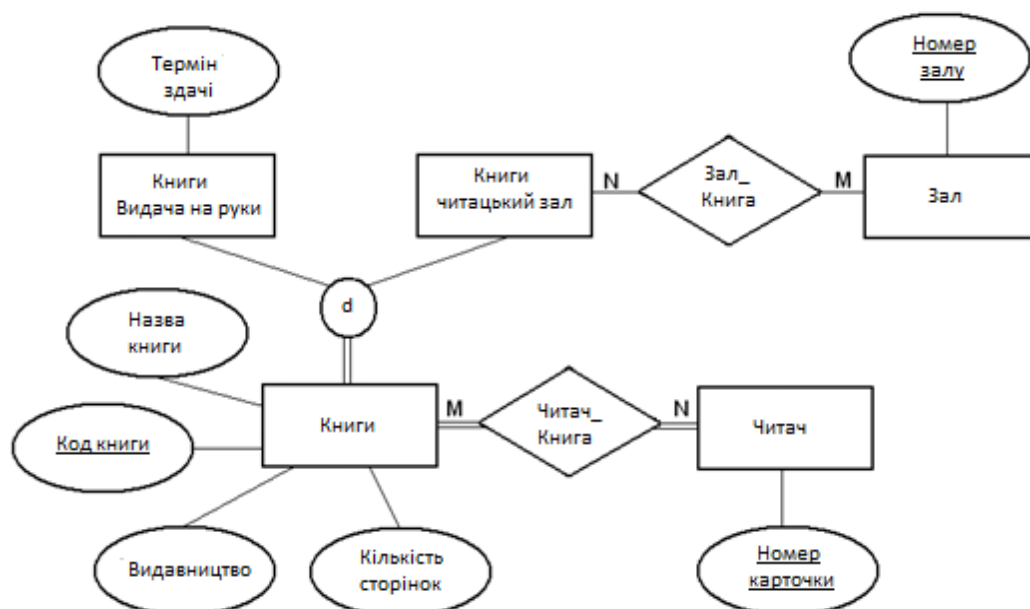
Для оцінки доцільності застосування спеціалізації і генералізації рекомендуємо спробувати, використовуючи ці процедури, представити на ER-моделі найважливіші сутності та їх зв'язку. Ступінь спеціалізації або

генералізації об'єктів ER-діаграми визначається, перш за все, загальною читабельністю цієї діаграми і тією ясністю, з якою на ній моделюються найважливіші типи сутностей і зв'язків додатків.

Застосування концепцій спеціалізації і генералізації дозволяє поліпшити характеристики створеної ER-моделі програми. Однак цей етап є необов'язковим.



а



б

Рис. 5.1: а) сутності Книги (читацький зал) і Книги (видача на руки); б) суперклас Книги з підкласами Книги (читацький зал) і Книги (видача на руки)

Етап 1.7. Створення діаграми "сутність-зв'язок"

Мета: розробка діаграм "сутність-зв'язок" (ER-діаграм), що містять концептуальне відображення представлення користувача про предметну область додатку.

На цьому етапі створюються остаточні варіанти ER-діаграм, що відображають локальні концептуальні моделі даних, що характеризують представлення окремих користувачів про предметну область додатку.

Етап 1.8. Обговорення локальних концептуальних моделей даних з кінцевими користувачами

Мета: обговорення локальних концептуальних моделей даних з кінцевими користувачами з метою отримання підтверджень, що дана модель коректно відображає представлення користувача про додаток та підприємство.

Перш ніж завершити перший етап розробки, необхідно обговорити створені локальні концептуальні моделі даних з кінцевими користувачами. Концептуальна модель даних повинна бути представлена ER-діаграмою і супровідною документацією, що містить опис розробленої моделі даних. Якщо у запропонованій моделі будуть виявлені які-небудь невідповідності, слід внести в неї необхідні зміни (швидше за все, для цього буде потрібно повторно виконати один або кілька попередніх етапів розробки). Цей процес повинен тривати до тих пір, поки користувач не підтвердить, що запропонована йому модель адекватно відображає його особисте представлення про роботу програми та підприємства в цілому.

5.4 Методологія логічного проектування реляційних баз даних

В цьому розділі обговоримо і продемонструємо на конкретних прикладах методологію створення логічної моделі реляційної бази даних.

Відправним пунктом проведеного обговорення є створений на першому етапі пропонованої методології проектування набір локальних концептуальних моделей даних і відповідної документації. Окремі моделі даних відображають представлення про предметну область додатку різноманітних типів його користувачів. Створення цих локальних моделей і необхідної супровідної документації описувалося вище в розділі "Методологія концептуального проектування баз даних". Для побудови локальних концептуальних моделей даних використовувалася ER-технологія моделювання.

У цьому розділі пропонується поетапна методологія логічного проектування баз даних реляційного типу. Процедура проектування починається з перетворення локальних концептуальних моделей даних в локальні логічні моделі даних, які згодом будуть використані для визначення необхідного набору відношень. Побудовані логічні моделі і створений набір відношень перевіряються за допомогою методів нормалізації. Додатково перевіряється можливість виконання всіх необхідних користувачам транзакцій. На останньому етапі цієї фази проектування окремі локальні концептуальні моделі даних зливаються в єдину глобальну модель даних всього підприємства, яка

відображатиме узагальнене представлення всіх користувачів про предметну область додатку.

Етап 2. Побудова і перевірка локальної логічної моделі даних для окремих представлень кожного з типів користувачів

Мета: побудова логічної моделі даних на основі концептуальної моделі даних, що відбиває представлення окремого користувача про предметну область додатку, і перевірка отриманої моделі за допомогою методів нормалізації та контролю виконання транзакцій.

На цьому етапі продовжимо роботу з локальними концептуальними моделями даних, створеними на попередньому етапі. Завдання полягає в доопрацюванні цих моделей з метою видалення з них всіх елементів, що ускладнюють реалізацію даних моделей в середовищі СУБД. В результаті виконання цих дій структури концептуальних моделей даних будуть змінені таким чином, щоб повністю відповідати вимогам, що висуваються реляційною моделлю організації баз даних. Тому нові моделі більш коректно називати логічними моделями даних. Далі створені логічні моделі даних будуть перевірені з використанням правил нормалізації і піддані контролю на можливість виконання всіх необхідних користувачам транзакцій так, як це зазначено в специфікаціях на створюваний додаток. Згодом перевірені логічні моделі даних можна буде використовувати як прототипи, якщо це буде потрібно.

В результаті виконання даного етапу будуть створені коректні, повні і точні моделі представлень користувачів. Це дасть міцну основу, необхідну для виконання наступного етапу, який укладається в об'єднанні окремих локальних логічних моделей даних в єдину глобальну модель даних всього підприємства.

Даний етап включає наступне:

Етап 2.1. Перетворення локальної концептуальної моделі даних в локальну логічну модель.

Етап 2.2. Визначення набору відношень виходячи зі структури локальної логічної моделі даних.

Етап 2.3. Перевірка моделі за допомогою правил нормалізації.

Етап 2.4. Перевірка моделі щодо транзакцій користувачів.

Етап 2.5. Створення діаграм "сутність-зв'язок".

Етап 2.6. Визначення вимог підтримки цілісності даних.

Етап 2.7. Обговорення розроблених локальних логічних моделей даних з кінцевими користувачами.

Етап 2.1. Перетворення локальної концептуальної моделі даних в локальну логічну модель

Мета: доробка локальних концептуальних моделей з метою видалення з них небажаних елементів і перетворення отриманих моделей в локальні логічні моделі даних.

В результаті виконання першого етапу отримаємо набір локальних концептуальних моделей даних, що відбивають представлення окремих користувачів про роботу підприємства. Однак ці моделі даних можуть містити деякі структури даних, реалізація яких в звичайних типах СУБД буде утруднена.

На цьому етапі подібні структури даних перетворюються в таку форму, яка не викличе труднощів при їх реалізації в середовищі існуючих СУБД. Може послідувати зауваження, що ці дії не є елементом логічного проектування баз даних. Однак пропонується процедура змушує розробника ретельніше обдумувати сенс кожного елемента даних, що позитивно позначається на точності відображення в моделі особливостей того чи іншого підприємства.

На даному етапі виконуються наступні дії:

1. Видалення зв'язків типу М: М.
2. Видалення складних зв'язків.
3. Видалення рекурсивних зв'язків.
4. Видалення зв'язків з атрибутами.
5. Видалення множинних атрибутів.
6. Перевірка зв'язків типу 1: 1.
7. Видалення надлишкових зв'язків.

1. Видалення зв'язків типу М: N

Якщо в концептуальній моделі присутні зв'язку типу М: N ("багато до багатьох"), то їх слід усунути шляхом визначення деякої проміжної сутності. Зв'язок типу М:N замінюється двома зв'язками типу 1:М, що встановлюються з новоствореної сутності.

Як приклад розглянемо зв'язок типу М:N: Група вивчає ряд предметів, і певний Предмет може вивчатися декількома групами. Зв'язок представлений на рис. 5.2а. З метою усунення зв'язку "багато до багатьох" (Група_Предмет) визначаємо проміжну сутність Розклад і створюємо дві нових зв'язку типу 1:М (Група_Розклад і Предмет_Розклад).

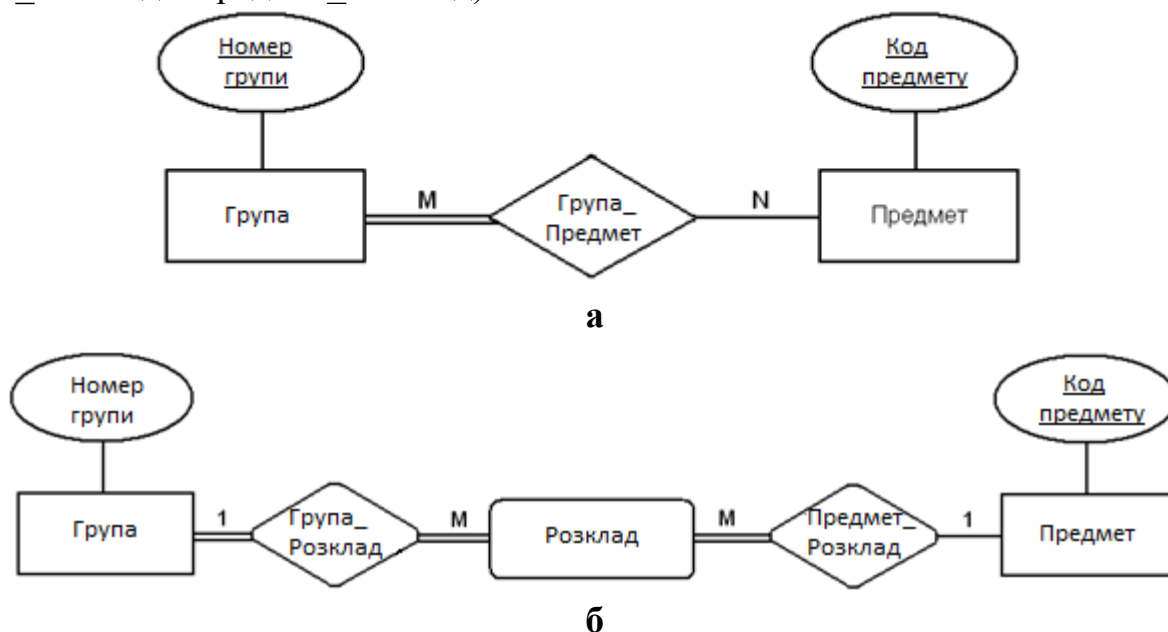


Рис. 4.2: а) зв'язок Група_Предмет типу М:N; б) усунення зв'язку типу М:N за допомогою заміни її двома зв'язками типу 1:М (Група_Розклад і Предмет_Розклад)

Зверніть увагу: сутність Розклад показана як слабка сутність, оскільки її існування залежить від сутностей-власників, а саме від сутностей Група і Предмет.

2. Видалення складних зв'язків

Складним називається зв'язок, що існує між трьома і більше типами сутностей.

Якщо в концептуальній моделі присутній складний зв'язок, його слід усунути за допомогою проміжної сутності. Складний зв'язок замінюється необхідною кількістю бінарних зв'язків типу 1:M, що встановлюються з новоствореної сутністю.

Наприклад, потрійний зв'язок Захист, відображає відносини, що існують між студентом, працюючим над курсовою роботою, керівником і самою курсовою роботою - як показано на рис. 5.3 а. Цей складний зв'язок можна спростити шляхом введення нової сутності і визначення бінарних зв'язків між нею і кожною з вихідних сутностей складного зв'язку. В даному прикладі зв'язок Захист можна усунути за допомогою введення нової слабкої сутності з ім'ям Тема курсової. Новостворена сутність буде пов'язана з вихідними сутностями трьома новими бінарними зв'язками - як показано на рис. 5.3 б.

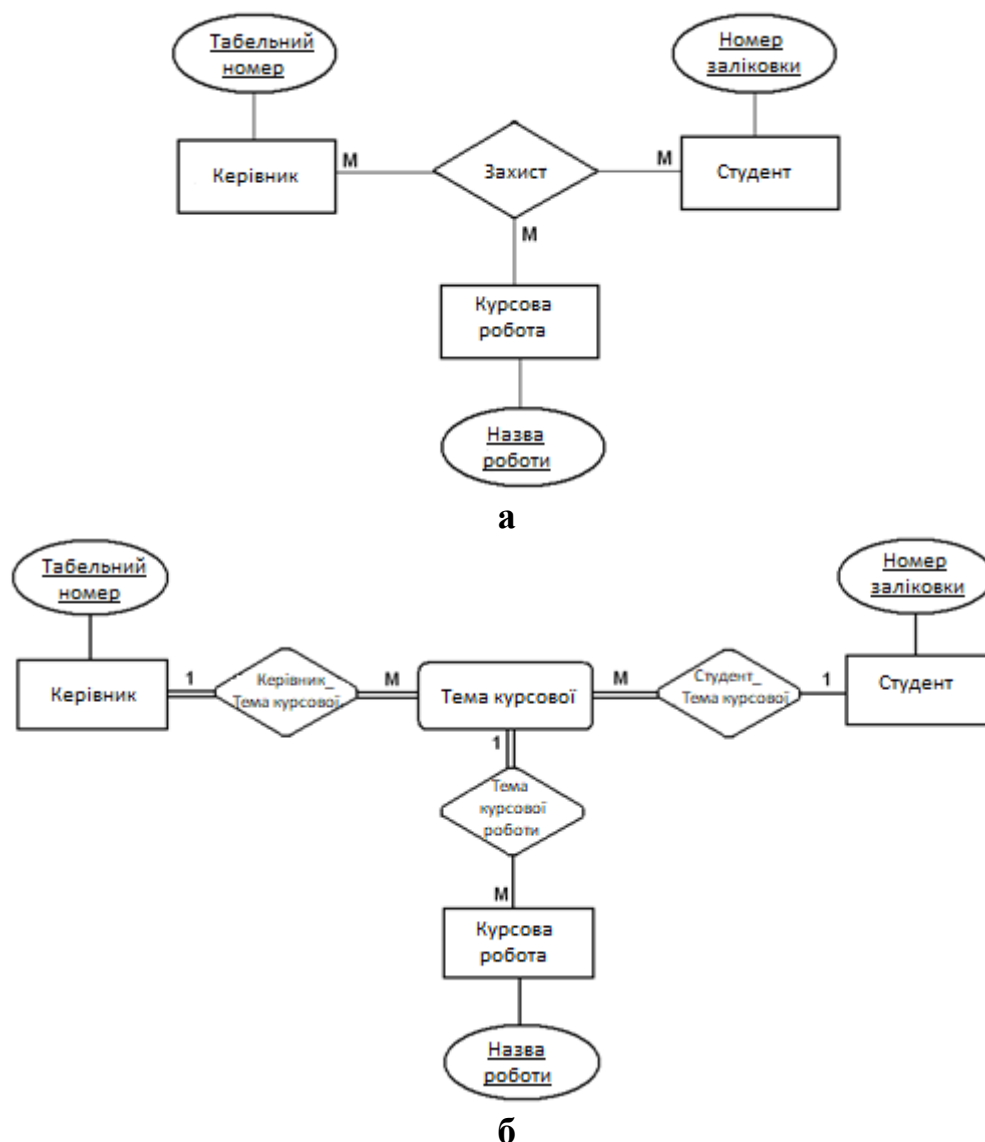


Рис. 5.3: а) складний зв'язок Захист; б) усунення складного зв'язку Захист за допомогою трьох зв'язків типу 1: М

3. Видалення рекурсивних зв'язків

Рекурсивними називаються такі зв'язки, в яких сутність деякого типу взаємодіє сама з собою.

Якщо концептуальна модель містить рекурсивні зв'язки, вони повинні бути усунуті за допомогою визначення деякої проміжної сутності.

Видалення рекурсивної зв'язки типу M:N виконується так само, як і бінарних зв'язків типу M:N, описаних вище.

Наприклад, для відображення ситуації, коли один з викладачів керує групою інших викладачів, може бути встановлений рекурсивний зв'язок типу "один до багатьох" (1:M), як показано на рис. 5.4 а.

Рекурсивна природа цього зв'язку вимагає особливого підходу при роботі з ним як на етапі логічного проектування, так і на етапі фізичної реалізації бази даних. Для спрощення цього рекурсивного зв'язку типу 1:M замінимо його новоствореною слабкою сутністю Керівний Викладач і додатковим зв'язком типу 1:1 з ім'ям Співпраця, як показано на рис. 5.4 б.

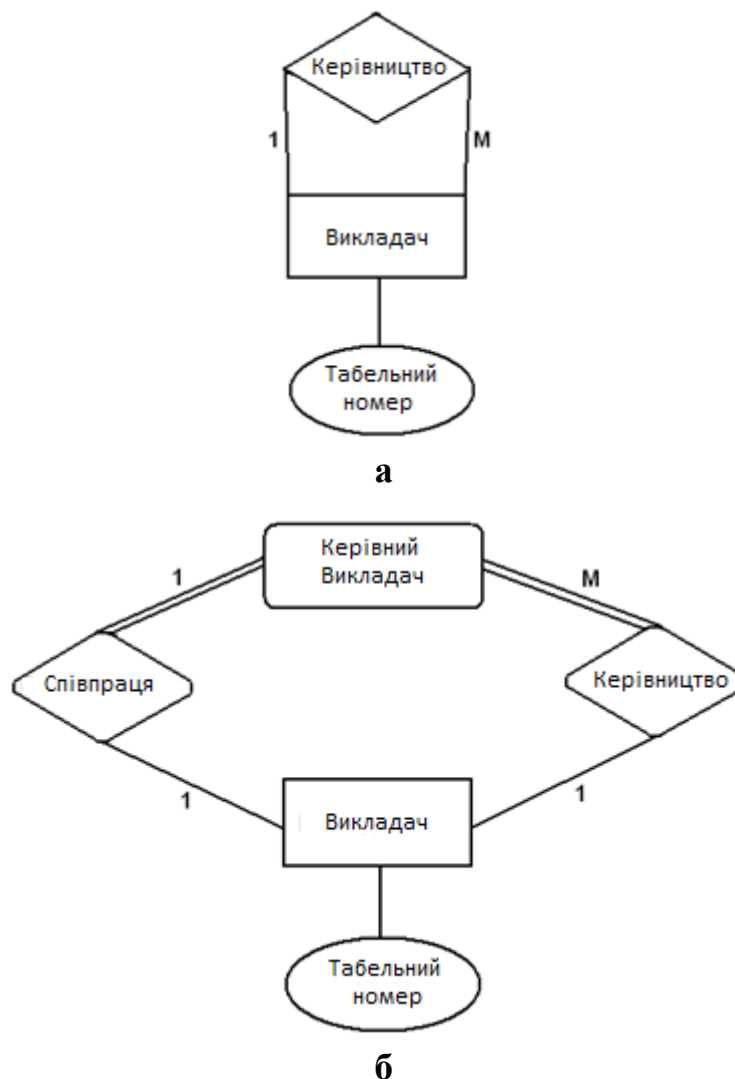


Рис. 5.4; а) рекурсивний зв'язок Керівництво; б) усунення рекурсивного зв'язку Керівництво за допомогою створення додаткової зв'язку Співпраця і сутності з ім'ям Керівний Викладач

4. Видалення зв'язків з атрибутами

Якщо в концептуальній моделі присутні зв'язки, які мають власні атрибути, вони повинні бути перетворені шляхом створення нової сутності.

Наприклад, розглянемо ситуацію, коли потрібно фіксувати оплату студента за відвідувану їм секцію (рис. 5.5 а).

Зв'язок Студент_Секція має атрибут з ім'ям Плата. Перетворимо цей зв'язок в слабку сутність з ім'ям Договір Про Відвідування Занять, якій призначимо атрибут Плата, після чого створимо два нових зв'язки типу 1:М, як показано на рис. 5.5 б.

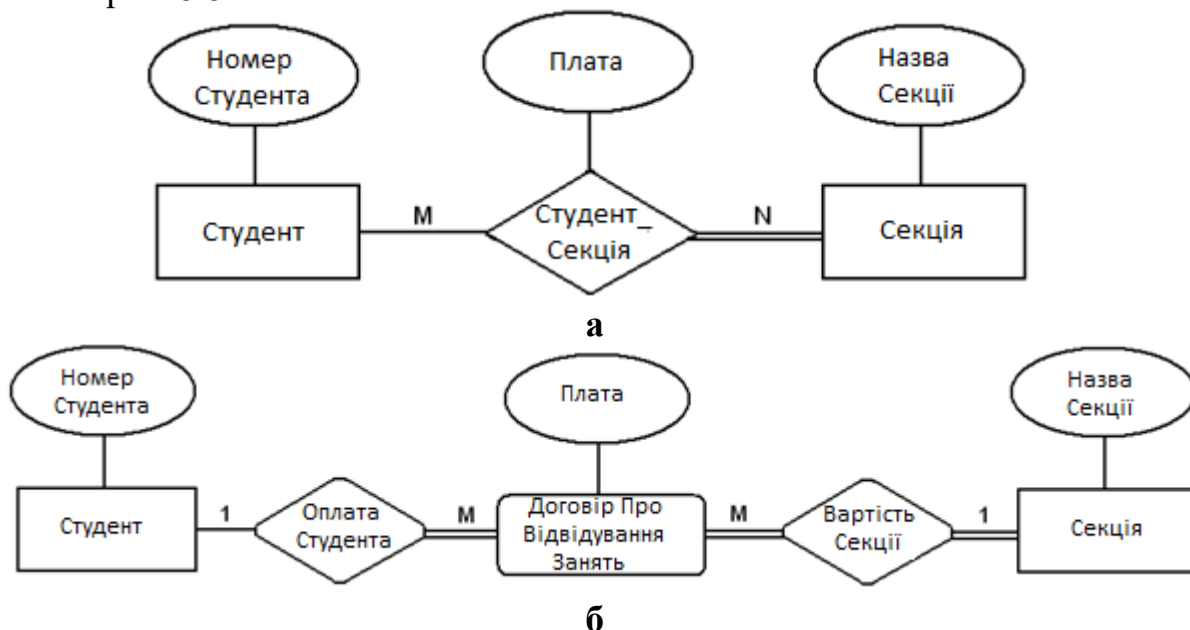


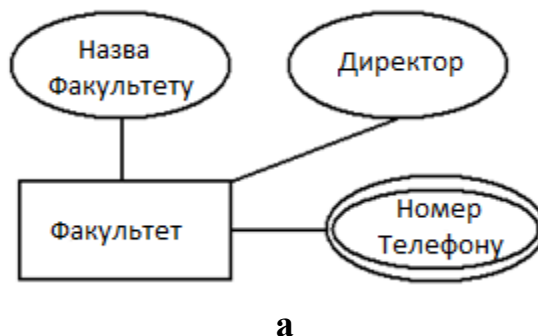
Рис. 5.5: а) зв'язок Студент_Секція має атрибут Плата; б) усунення зв'язку Студент_Секція за допомогою створення сутності Договір Про Відвідування Занять і двох зв'язків типу 1:М (Оплата Студента і Вартість Секції)

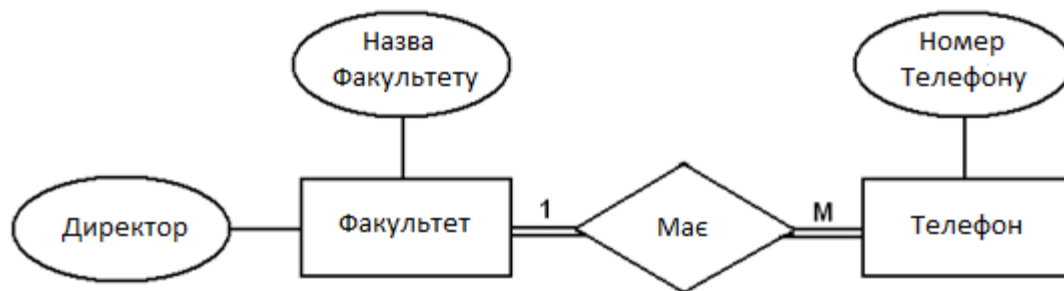
5. Видалення множинних атрибутів

Множинними називають атрибути, які можуть мати одночасно кілька значень для одного і того ж екземпляра сутності.

Якщо в концептуальній моделі присутній множинний атрибут, його слід перетворити шляхом визначення нової сутності.

Наприклад, для відображення ситуації, коли один і той же факультет має кілька телефонних номерів, в концептуальній моделі міг бути визначений множинний атрибут Номер Телефону, що відноситься до сутності Факультет, як показано на рис. 5.6 а. Цей множинний атрибут слід видалити, визначивши нову сутність Телефон, що має єдиний простий атрибут Номер Телефону, і створивши новий зв'язок типу 1:М з ім'ям Має (рис. 5.6 б).





б

Рис. 5.6: а) сутність Факультет має множинний атрибут Номер Телефона; б) усунення множинного атрибуту Номер Телефона за допомогою зв'язку Має типу 1:М і нової сутності Телефон з простим атрибутом Номер Телефона

6. Перевірка зв'язків типу 1: 1

У процесі визначення сутностей могли бути створені дві різні сутності, які насправді представляють один і той же об'єкт в предметній області додатку. У подібному випадку слід об'єднати ці дві сутності в одну. Якщо первинні ключі сутностей, що об'єднуються різні, виберіть один з них в якості первинного, а другий вкажіть як альтернативний ключ.

Наприклад, могли бути створені дві сутності, Книга і Підручник, які насправді можуть представляти один і той же тип об'єкта. Іншими словами, ім'я Книга є синонімом імені Підручник.

7. Видалення надлишкових зв'язків

Зв'язок є надлишковим, якщо одна і та ж інформація може бути отримана не тільки через нього, а й за допомогою іншого зв'язку.

Завжди слід прагнути створювати мінімальні моделі даних, і тому, якщо надлишковий зв'язок не є очевидно необхідним, його слід видаляти.

Встановити, що між двома сутностями є більше одного зв'язку, досить просто. Однак з цього ще не випливає, що один з двох зв'язків обов'язково є надлишковим, оскільки обидва вони можуть представляти різні об'єднання, реально існуючі в предметній області.

При усуненні надлишковості доступу велике значення мають тимчасові показники.

Наприклад, розглянемо ситуацію, коли необхідно змодельовати зв'язки між сутностями Чоловік, Жінка і Дитина, як показано на рис. 5.7. Очевидно, що між сутностями Чоловік і Дитина є два шляхи доступу: один - через безпосередній зв'язок Є Батьком і інший - через зв'язок Одружений З. На перший погляд, здається, що зв'язок Є Батьком є надлишковим. Однак це твердження може виявитися помилковим з двох причин. По-перше, батько може мати дітей від попереднього шлюбу, а моделюємо тільки поточний шлюб батька (через зв'язок 1:1). По-друге, батько і мати можуть бути взагалі неодружені, або батько може бути одружений на жінці, яка не є матір'ю даної дитини (або ж мати може бути одружена з чоловіком, який не є батьком дитини). Тому всі існуючі взаємини не можуть бути змодельовані без використання зв'язку типу Є Батьком.

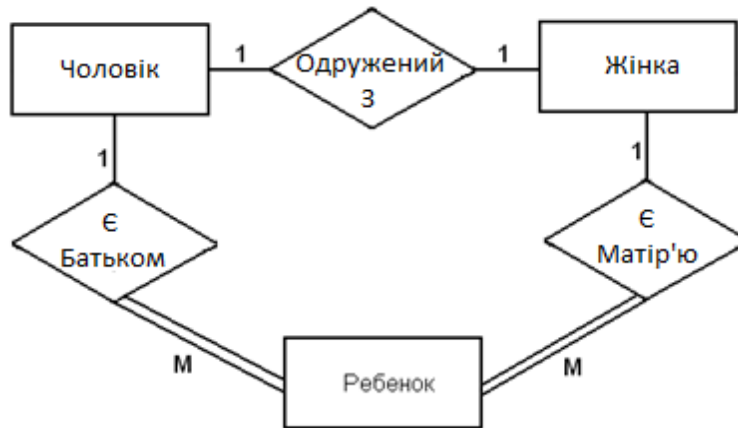


Рис. 5.7. Приклад множинних зв'язків, які не є надлишковими

Суть полягає в тому, що при усуненні надмірності дуже важливо досліджувати значення кожного із зв'язків, що існують між сутностями.

По завершенні цього етапу отримали спрощену локальну концептуальну модель даних, з якої вилучені всі структури, реалізація яких в середовищі реляційних СУБД є важкою.

Тому на даному етапі правильніше буде називати поліпшену локальну концептуальну модель **локальною логічною моделлю даних**.

Етап 2.2. Визначення набору відношеньвиходячи зі структури локальної логічної моделі даних

Мета: визначення набору відношеньна основі локальної логічної моделі даних.

На даному етапі доведеться на основі створених локальних логічних моделей даних визначити набори відносин, необхідні для подання сутностей і зв'язків, що входять в представлення окремих користувачів про предметну область додатку.

Для опису структури створюваних відношенькористаємося мовою DBDL (Database Definition Language), що широко використовується в СУБД. Опис відношення на мові DBDL починається з присвоєного йому імені, за яким йде поміщений в дужки список імен його простих полів. Потім вказується первинний ключ відношення і всі його альтернативні і/або зовнішні ключі. Після опису зовнішніх ключів може вказуватися контрольний первинний ключ, якщо такий є.

Тепер розглянемо, як відношення, що відображають сутності і зв'язки між ними, можуть бути створені виходячи з наявних в логічній моделі різних допустимих структур даних. Для ілюстрації цього процесу буде використовуватися логічна модель, показана на рис. 5.8.

Зв'язки, які сутність має з іншими типами сутностей, представляються за допомогою механізму первинних і зовнішніх ключів. Для прийняття рішення про те, звідки взяти і куди помістити значення атрибута(ів) зовнішнього ключа, попередньо слід встановити, яка з сутностей, що беруть участь в зв'язку є батьківською, а яка - дочірньою. **Батьківською** вважається сутність, яка передає копію набору значень свого первинного ключа в відношення, що

представляє дочірню сутність, де ці значення будуть грати роль зовнішнього ключа.

Сильні типи сутностей

Для кожної сильної (регулярної) сутності в локальній моделі даних створюється, відношення, що включає всі прості атрибути цієї сутності. У разі складових атрибутів (наприклад, адреси) в відношення включаються тільки складові їх прості атрибути (такі, як місто, вулиця, поштовий код).

Нижче наведено приклад складення відношення Викладач, визначеного для логічної моделі, показаної на рис. 5

.8.

Викладач (Табельний номер, ПІБ, Вчений ступінь, Адреса, Телефон, Код кафедри)

Primary Key Табельний номер

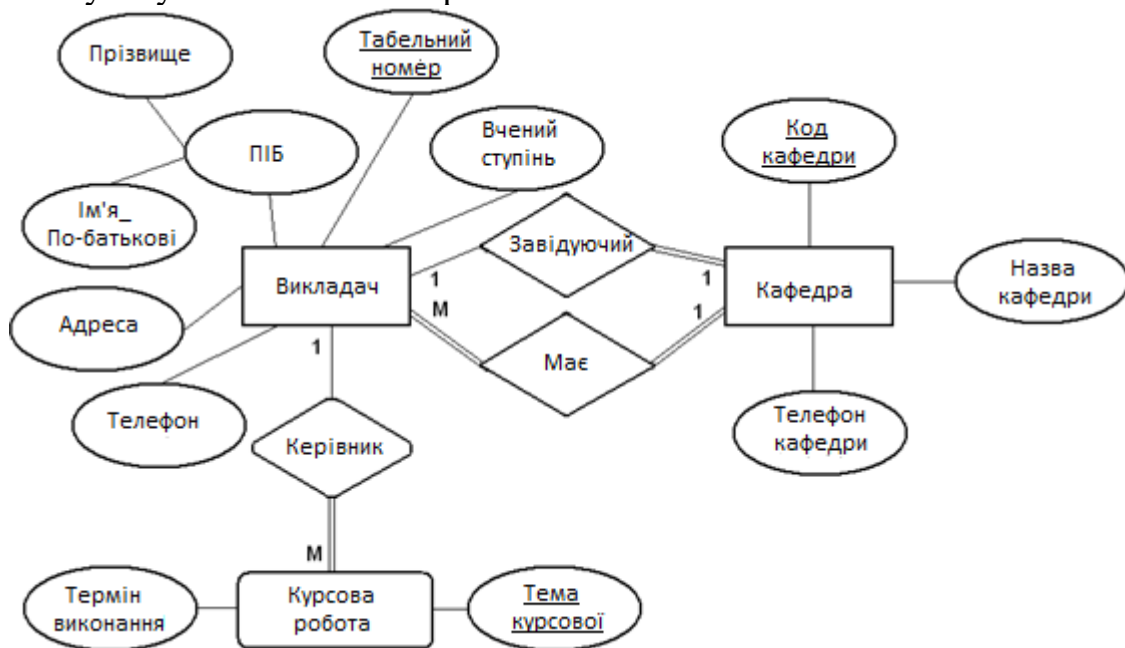


Рис.5.8 Приклад логічної моделі даних

Слабкі типи сутностей

Для кожної слабкою сутності, присутньої в логічній моделі, створюється відношення, що включає всі прості атрибути цієї сутності. Додатково в відношення включається атрибут зовнішнього ключа, що відповідає первинному ключу сутності-власника. Первинний ключ слабкої сутності частково або повністю виводиться з ключа сутності-власника.

Наприклад, показана на рис. 5.8 сутність Викладач є власником слабкої сутності Курсова робота (мається на увазі, що певний викладач керує курсовою роботою). Опис, що представляє цю сутність відношення приведено нижче.

Курсова робота (Табельний номер, Тема курсової, Термін виконання)

Primary Key Табельний номер, Тема курсової

Foreign Key Табельний номер references Викладач (Табельний номер)

Зверніть увагу на те, що атрибут зовнішнього ключа відношення Курсова робота входить до складу первинного ключа цієї сутності. У даній ситуації

первинний ключ відношення Курсова робота неможливо встановити до тих пір, поки в неї не буде поміщений зовнішній ключ, обраний в відношенні Викладач. Таким чином, до кінця цього етапу необхідно визначити будь-які первинні і потенційні ключі, які можуть знадобитися для побудови набору відношень, необхідного для повного представлення вихідної логічної моделі даних.

Бінарні зв'язку типу "один до одного" (1: 1)

Для кожного присутнього в логічній моделі даних бінарного зв'язку типу 1:1, встановленого між сутностями E1 і E2, необхідно переслати атрибути первинного ключа сутності E1 в відношення, що представляє сутність E2. Ці атрибути будуть використовуватися в ньому в якості зовнішнього ключа. Визначення батьківської і дочірньої сутностей залежить від обмежень участі, накладених на члени відношень E1 і E2.

Сутність, яка частково бере участь в зв'язку, визначається як батьківська, а та сутність, яка бере участь в зв'язку повністю (тотально), визначається як дочірня.

Копія набору значень первинного ключа батьківської сутності поміщається в відношення, що представляє дочірню сутність.

Відзначимо, що в тому випадку, коли обидва види сутностей беруть участь в зв'язку типу 1:1 або тотально, або частково, вибір батьківської і дочірньої сутностей може виконуватися довільно. Більш того, якщо обидві сутності беруть участь в зв'язку тотально, можна (на вибір) або представити цей зв'язок за допомогою пари первинного та зовнішнього ключів (як описувалося вище), або злити атрибути обох сутностей в єдине відношення. Злиття в єдине відношення краще в тому випадку, якщо дані сутності не беруть участі в інших типах зв'язків.

Нижче наведено приклад, який ілюструє, як зв'язок типу 1:1 можна представити в відношеннях, створених на базі запропонованої вище логічної моделі даних.

Зв'язок Завідувач, показана на рис. 5.8, є зв'язком типу 1:1, оскільки кожна кафедра має тільки одного завідувача. Сутність Викладач бере участь в цьому зв'язку частково, тоді як сутність Кафедра - повністю. Та сутність, яка бере участь в зв'язку частково (Викладач), визначається як батьківська, а та сутність, яка бере участь в зв'язку повністю (Кафедра), - як дочірня. Отже, копія первинного ключа сутності Викладач (батьківської), представленого атрибутом Табельний номер, пересилається в відношення Кафедра (дочірнє). В результаті відношення Викладач і Кафедра повинні бути визначені в такий спосіб:

Викладач (Табельний номер, ПІБ, Вчений ступінь, Адреса,
Телефон, Код кафедри)

Primary Key Табельний номер

Кафедра (Код кафедри, Назва кафедри, Телефон кафедри, Табельний
номер завідувача)

Primary Key Код кафедри

Foreign Key Табельний номер завідувача references Викладач

(Табельний номер)

Зверніть увагу на те, що атрибут Табельний номер, який представляє завідувача кафедри, був перейменований в Табельний номер завідувача. Це

зроблено з метою більш точної вказівки призначення даного зовнішнього ключа відношення Кафедра.

Бінарні зв'язку типу "один до багатьох" (1:M)

Для кожного бінарного зв'язку типу 1:M, встановленого в логічній моделі даних між сутностями E1 і E2, необхідно переслати копію атрибутів первинного ключа сутності E1 в відношення, що представляє сутність E2, де вони будуть грати роль зовнішнього ключа.

Сутність, яка представляє "одиначну" сторону зв'язку визначається як батьківська, а сутність, що представляє "множинну" сторону, - як дочірня. Як і в попередньому випадку, для представлення даного зв'язку необхідно скопіювати первинний ключ батьківської сутності в відношення, що представляє дочірню сутність, де цей ключ повинен бути описаний як зовнішній.

Нижче наведено приклад, який ілюструє, як зв'язок типу 1:M можна представити у відношеннях, створених на базі запропонованої вище логічної моделі даних.

Зв'язок типу має, показаний на рис. 5.7, має тип 1:M, оскільки до кожної кафедри відносяться кілька викладачів. У цьому випадку відношення Кафедра є "одиначною" стороною зв'язку і представляє батьківську сутність, тоді як відношення Викладач є "множинною" стороною зв'язку і представляє дочірню сутність. Зв'язок між цими двома сутностями встановлюється шляхом розміщення копії первинного ключа сутності Кафедра (батьківської), представленого атрибутом Код кафедри, в відношення Викладач (дочірнє). В результаті остаточний вигляд відношень Викладач і Кафедра повинен бути наступним:

Викладач (Табельний номер, ПІБ, Вчений ступінь, Адреса,
Телефон, Код кафедри)

Primary Key Табельний номер

Foreign Key Код кафедри references Кафедра (Код кафедри)

Кафедра (Код кафедри, Назва кафедри, Телефон кафедри, Табельний
номер завідувача)

Primary Key Код кафедри

Foreign Key Табельний номер завідувача references Викладач

(Табельний номер)

Зв'язки типу "суперклас/підклас"

Для кожного присутнього в логічній моделі даних зв'язку типу "суперклас/підклас" сутність **суперкласу** необхідно визначити як батьківську, а сутність **підкласу** - як дочірню. Існують різні варіанти представлення подібних зв'язків у вигляді одного або декількох відношень. Вибір найбільш відповідного варіанту залежить від обмежень участі і перетину, накладених на учасників зв'язку типу "суперклас/підклас".

Розглянемо, наприклад, зв'язок типу "суперклас/підклас" сутності Property, показаної на рис. 5.1 б. Існують різні способи відображення цього зв'язку.

(Варіант 1)

Всі_Книги (Код книги, Назва книги, Видавництво, Кількість
сторінок, Термін здачі)

Primary Key Код книги

(Варіант 2)

Книги (читацький зал) (Код книги, Назва книги, Видавництво,
Кількість сторінок)

Primary Key Код книги

Книги (видача на руки) (Код книги, Назва книги, Видавництво,
Кількість сторінок, Термін здачі)

(Варіант 3)

Книги (Код книги, Назва книги, Видавництво, Кількість
сторінок)

Primary Key Код книги

Книги (читацький зал) (Код книги)

Primary Key Код книги

Foreign Key Код книги references Книги (Код книги)

Книги (видача на руки) (Код книги, Термін здачі)

Primary Key Код книги

Foreign Key Код книги references Книги (Код книги)

Діапазон можливих варіантів рішень простягається від розміщення всіх атрибутів суперкласу в єдине відношення (варіант 1) до розподілу всіх атрибутів між трьома різними відношеннями (варіант 3). Найбільш відповідний варіант для кожного зв'язку типу "суперклас/підклас" визначається обмеженнями, специфічними для даного конкретного зв'язку. Зв'язки, які суперклас Книги має з його підкласами, є тотальними і непересічними, оскільки кожен член суперкласу Книги обов'язково повинен бути членом одного і тільки одного з підкласів (Книги (читацький зал) або Книги (видача на руки)). Тому самим доречним рішенням для зв'язку типу "суперклас/підклас" з тотальними і непересічними членами є представлення кожного з підкласів у вигляді окремого відношення, що містить копію первинного ключа суперкласу. Це другий варіант в наведеному вище прикладі. Однак на остаточне рішення можуть вплинути й інші фактори - зокрема те, включені чи ні окремі підкласи в різні зв'язки.

Документування створених відношень і атрибутів зовнішніх ключів

Документування складу відношень, створених на базі кожної з логічних моделей даних, здійснюється на мові DBDL. Відзначимо, що синтаксис мови DBDL може бути розширено з метою зазначення обмежень цілісності для зовнішніх ключів (етап 2.6). Крім того, слід оновити вміст словника даних для відображення відомостей про будь-які нові ключові атрибути, визначених на даному етапі.

Етап 2.3. Перевірка моделі за допомогою правил нормалізації

Мета: перевірка локальної логічної моделі даних з використанням технології нормалізації.

Нормалізація використовується для поліпшення моделі даних, для того щоб вона задовольняла різним обмеженням, що дозволяють виключити небажане дублювання даних. Нормалізація гарантує, що отримана в результаті її застосування модель даних буде найкращим чином відображати особливості використання інформації на підприємстві, не містити протиріч, мати мінімальну надмірність і максимальну стійкість.

Нормалізація являє собою процедуру прийняття рішень про те, які саме атрибути повинні бути об'єднані для представлення сутностей кожного типу. В одній з фундаментальних концепцій теорії нормалізації стверджується, що атрибути повинні бути згруповані в відношення відповідно до існуючих між ними логічними зв'язками. У деяких випадках мають місце твердження, що нормалізація розроблюваних баз даних не дозволяє досягти максимальної продуктивності при їх обробці. На це можна відповісти наступними зауваженнями:

- Нормалізація проекту дозволяє організувати розміщення даних відповідно до їх функціональних залежностей. Тому дана процедура повинна виконуватися між етапами концептуального і фізичного проектування.

- На етапі логічного проектування не ставиться завдання досягти остаточного вигляду проекту. Мета цього етапу полягає в наданні проектувальнику більш поглибленого розуміння природи і призначення даних, що використовуються на підприємстві. Якщо до додатка ставляться специфічні вимоги щодо його продуктивності, то вони повинні враховуватися на етапі фізичного проектування. У цьому випадку одним з підходів є денормалізація певних таблиць. Однак це не означає, що час на їх нормалізацію було витрачено даремно, оскільки для правильного виконання нормалізації проектувальник повинен глибоко вивчити семантику і особливості використання даних. Детальніше про денормалізацію мова піде в розділі "Методологія фізичного проектування реляційних баз даних" (етап 5.4).

- Нормалізація проекту дозволяє підвищити його стійкість і позбутися від аномалій оновлення.

- За останні кілька років потужність комп'ютерів істотно зросла. Тому в деяких випадках може бути цілком обґрунтованим рішення реалізувати проект, що дозволяє спростити роботу з даними за рахунок виконання певного обсягу додаткової обробки.

- Виконання нормалізації вимагає від розробника повного розуміння призначення кожного атрибута, присутнього в створюваній базі даних. Досягнуті за рахунок цього переваги можуть виявитися вельми істотними.

- В результаті застосування нормалізації розробник отримує досить гнучкий проект бази даних, що дозволяє легко вносити в неї необхідні розширення.

На попередньому етапі був створений набір відношень, що реалізують локальні логічні моделі даних. На цьому етапі проаналізуємо коректність об'єднання атрибутів в кожному з відношень. Іншими словами, завдання полягає в перевірці правильності складу кожного з створених відношень за допомогою застосування до них процедури нормалізації. Процес нормалізації включає наступні чотири основні етапи:

- приведення до першої нормальної форми (1НФ), що дозволяє видалити з відношень повторювані групи атрибутів;
- приведення до другої нормальної форми (2НФ), що дозволяє усунути часткову залежність атрибутів від первинного ключа;
- приведення до третьої нормальної форми (3НФ), що дозволяє усунути транзитивну залежність атрибутів від первинного ключа;
- приведення до нормальної форми Бойса-Кодда (НФБК), що дозволяє видалити з функціональних залежностей аномалії, що залишилися. Метою виконання цих етапів є отримання гарантій того, що кожне з відношень, створених на підставі логічної моделі даних, відповідає, принаймні, вимогам НФБК. Якщо будуть знайдені відношення, що не відповідають вимогам НФБК, це може вказувати на те, що частина логічної моделі даних невірна або перетворення логічної моделі в набір відношень виконано некоректно. При необхідності потрібно перебудувати модель даних і переконатися, що вона вірно відображає модельовану частину інформаційної структури підприємства.

Етап 2.4. Перевірка моделі в відношенні транзакцій користувачів

Мета: переконатися в тому, що локальна логічна модель даних дозволяє виконати всі транзакції, передбачені даним представленням користувача.

Метою виконання даного етапу є перевірка локальної логічної моделі даних на можливість виконання всіх транзакцій, передбачених даним представленням користувача. Перелік транзакцій визначається відповідно до специфікацій, що описують дії, що виконуються даним користувачем. Використовуючи ER-діаграми, словник даних і встановлені зв'язки між первинними і зовнішніми ключами, зазначені в описі відношень, необхідно виконати всі необхідні операції доступу до даних вручну. Якщо вдасться подібним чином знайти спосіб виконання всіх необхідних транзакцій, то на цьому перевірка логічної моделі даних буде завершена. Однак, якщо будь-яку з транзакцій виконати вручну не вдасться, значить, складена модель даних є неадекватною і містить помилки, які потрібно усунути. Найімовірніше, помилка буде пов'язана з пропуском в моделі даних сутності, зв'язку або атрибуту.

Розглянемо **два можливих підходи**, завдяки яким зможемо переконатися, що локальна логічна модель даних дозволяє виконати всі необхідні транзакції.

Перший підхід передбачає виконання перевірки того, що дана логічна модель надає всю інформацію (сутності, зв'язки та їх атрибути), необхідну для виконання кожної з транзакцій. Практично це реалізується при підготовці опису вимог, висунутих кожній з транзакцій.

Наприклад, транзакції, необхідні завідувачу кожної кафедри, можуть включати наступні операції:

- а) Введення відомостей про нового викладача.

Первинним ключем відношення Викладач є атрибут Табельний номер. Перш за все, переконаємося, що присвоєний новому викладачеві табельний номер є унікальним. Якщо це не так, введення даних слід заборонити, а виконання операції завершити. В іншому випадку введення даних про нового викладача дозволяється. Перевірте, що для кожного елемента відомостей про викладача в відношенні Викладач є відповідний атрибут.

б) Видалення відомостей про викладача, заданому його табельною номером.

Виконується пошук заданого табельного номера у відповідному стовпці відношень Викладач. Якщо він не буде знайдений, то фіксується помилка і ніякі подальші дії не виконуються. В іншому випадку з відношень Викладач видаляється весь знайдений кортеж і оновлюється зовнішній ключ всіх кортежів інших відношень, з якими був пов'язаний викладач, що "видаляється".

Другий підхід до перевірки моделі даних на відповідність необхідним транзакціям полягає в нанесенні безпосередньо на ER-діаграми всіх шляхів, які будуть потрібні для виконання кожної з транзакцій.

Простий приклад використання цього підходу для контролю за виконанням згаданих вище транзакцій (а і б) показаний на рис. 5.9.

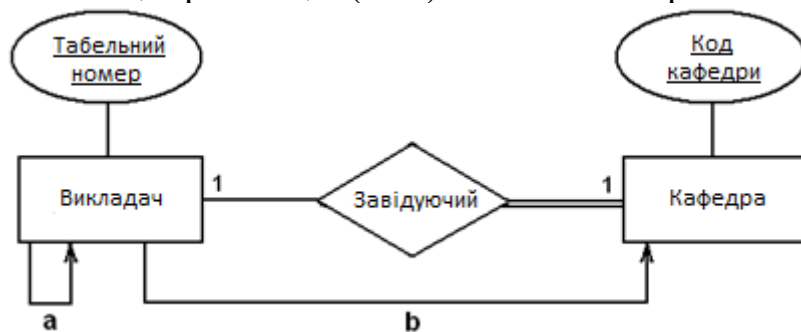


Рис. 5.9. Представлення транзакцій (а і б) на ER-діаграмі

Цей підхід дозволяє візуально виділити ті області моделі, які не використовуються для виконання транзакцій, а також ті області, які найбільш істотні з точки зору виконання транзакцій. У нашому розпорядженні виявляється зручний засіб прямого аналізу підтримки, яку модель даних забезпечує для виконання необхідних транзакцій користувача. Якщо на діаграмі є області, які не використовуються ні в одній з транзакцій, виникає питання про доцільність подання цієї інформації в моделі даних. У той же час, якщо в моделі присутні області, які не дозволяють знайти відповідний метод виконання деякої транзакції, буде потрібно провести аналіз того, яка з обов'язкових для виконання транзакції сутностей або зв'язків була пропущена при складанні моделі.

Етап 2.5. Створення діаграм "сутність-зв'язок"

Мета: створення остаточного варіанту діаграм "сутність-зв'язок" (ER-діаграм), що є локальним логічним поданням даних, що використовуються окремими користувачами додатку.

Тепер все готово для того, щоб створити остаточні варіанти ER-діаграм для окремих представлень кожного з користувачів підприємства. Дані на цих діаграмах були перевірені із застосуванням методів нормалізації, а також проконтрольовані на предмет можливості виконання всіх необхідних транзакцій.

Етап 2.6. Визначення вимог підтримки цілісності даних

Мета: визначення обмежень, що накладаються в представленнях користувачів вимогами збереження цілісності даних.

Обмеження цілісності даних представляють собою такі обмеження, які вводяться з метою запобігти розміщення в базу суперечливих даних. Відзначимо, що, хоча в конкретних СУБД функції контролю цілісності можуть, як підтримуватися, так і не підтримуватися, в даному випадку це не буде цікавити. На цьому етапі виконується проектування тільки на верхньому рівні, де розгляд питань цілісності даних є обов'язковою умовою, не пов'язаним з конкретними аспектами реалізації. Повне і точне відображення представлення користувача можна отримати тільки після визначення обмежень, необхідних з точки зору збереження цілісності даних. При необхідності на основі локальної логічної моделі можна навіть створити попередній фізичний проект бази даних, який згодом може послужити прототипом системи для даного конкретного користувача.

Тут будуть обговорені *п'ять типів обмежень цілісності даних*:

- обов'язкові дані;
- обмеження для доменів атрибутів;
- цілісність сутностей;
- посилювальна цілісність;
- вимоги даного підприємства.

Обов'язкові дані

Деякі атрибути завжди повинні містити одне з допустимих значень. Іншими словами, ці атрибути *не можуть мати порожнього значення*.

Так, кожен викладач повинен викладати той чи інший предмет.

Ці обмеження повинні фіксуватися при занесенні відомостей про атрибут в словник даних (етап 1.3).

Обмеження для доменів атрибутів

Кожен атрибут має домен, який представляє собою набір його допустимих значень.

Наприклад, атрибут "стать" може містити одне з двох допустимих значень - "Ч" або "Ж", тому його домен складається з двох символічних рядків довжиною в один символ, що містять вказані значення.

Дані обмеження встановлюються при визначенні доменів атрибутів, присутніх в моделі даних (етап 1.4).

Цілісність сутностей

Первинний ключ будь-якої сутності не може містити порожнього значення.

Наприклад, кожен рядок відношення Викладач повинен містити унікальне значення атрибута первинного ключа; в даному випадку це - атрибут Табельний номер.

Подібні обмеження повинні враховуватися при визначенні первинних ключів для сутностей кожного типу (етап 1.5).

Посилальна цілісність

Зовнішній ключ пов'язує кожен рядок дочірнього відношення з тим рядком батьківського відношення, яке містить це ж значення відповідного потенційного ключа. Поняття посилювальної цілісності означає, що якщо зовнішній ключ містить деяке значення, то воно обов'язково повинно бути присутнім в потенційному ключі одному з рядків батьківського відношення.

Наприклад, атрибут Код кафедри відношення Викладач пов'язує дані про кожного з викладачів з рядком в відношенні Кафедра, тобто вказується до якої кафедри відноситься викладач. Якщо поле Код кафедри не порожнє, воно повинно містити допустиме значення, яке присутнє у атрибуті Код кафедри одного з рядків відношення Кафедра. В іншому випадку виявиться, що викладач відноситься до неіснуючої кафедри.

Існує кілька важливих моментів, пов'язаних з використанням зовнішніх ключів. По-перше, слід проаналізувати, чи припустимо використання в зовнішніх ключах порожніх значень. Наприклад, чи має сенс зберігати відомості про викладача, якщо не заданий код кафедри, до якої він належить? Питання не в перевірці існування заданого коду кафедри, а в тому, чи обов'язково цей код повинен зазначатися. У загальному випадку, якщо участь дочірнього відношення в зв'язку є тотальною, то рекомендується забороняти використання порожніх значень у відповідному зовнішньому ключі. У той же час, якщо участь дочірнього відношення в зв'язку є частковою, то розміщення порожніх значень в атрибут зовнішнього ключа повинно бути дозволено.

Наступна проблема пов'язана з організацією підтримки посилальної цілісності. Реалізація цієї підтримки здійснюється за допомогою завдання обмежень існування, що визначають умови, при яких може вставлятися, оновлюватися або видалятися кожне значення потенційного або зовнішнього ключа. Для прикладу розглянемо зв'язок типу 1:М між двома відношеннями Факультет і Кафедра. Первинний ключ відношення Факультет - атрибут Назва факультету - є зовнішнім ключем відношення Кафедра. Розглянемо наступні ситуації.

Випадок 1. Вставка нового рядка в дочірнє відношення (Кафедра). Для забезпечення посилальної цілісності необхідно переконатися, що значення атрибута зовнішнього ключа Назва факультету нового рядка відношення Кафедра дорівнює порожньому значенню або деякому конкретному значенню, присутньому в одному з рядків відношення Факультет.

Випадок 2. Видалення рядка з дочірнього відношення (Кафедра). При видаленні рядки з дочірнього відношення ніяких порушень посилальної цілісності не відбувається.

Випадок 3. Оновлення зовнішнього ключа в рядку дочірнього відношення (Кафедра). Цей випадок подібний до випадку 1. Для збереження посилальної цілісності необхідно переконатися, що атрибут Назва факультету в оновленому рядку відношення Кафедра містить або пусте значення, або деяке конкретне значення, яке присутнє у одному з рядків відношення Факультет.

Випадок 4. Вставка рядка в батьківське відношення (Факультет). Вставка рядка в батьківське відношення (Факультет) не може викликати порушення посилальної цілісності. Доданий рядок просто стає батьківським об'єктом, який не має дочірніх об'єктів.

Випадок 5. Видалення рядка з батьківського відношення (Факультет). При видаленні рядка з батьківського відношення посилальна цілісність буде порушена в тому випадку, якщо в дочірньому відношенні існуватимуть рядки,

що посилаються на видалений рядок батьківського відношення. У цьому випадку може бути використана одна з наступних стратегій.

- NO ACTION. Видалення рядка з батьківського відношення забороняється, якщо в дочірньому відношенні існує хоча б один рядок, що посилається на нього.

- CASCADE. При видаленні рядка з батьківського відношення автоматично видаляються всі рядки дочірнього відношення, що посилаються на нього. Якщо будь-який з рядків дочірнього відношення, що видаляється виступає в якості батьківської сторони в деякому іншому зв'язку, то операція видалення застосовується до всіх рядків дочірнього відношення цього зв'язку, і т.д .. Іншими словами, видалення рядка батьківського відношення автоматично поширюється на будь-які дочірні відношення .

- SET NULL. При видаленні рядка з батьківського відношення у всіх рядках дочірнього відношення, що посилаються на нього в атрибут зовнішнього ключа записується порожнє значення. Отже, видалення рядків з батьківського відношення викличе занесення порожнього значення в відповідний атрибут рядків дочірнього відношення. Ця стратегія може використовуватися тільки в тих випадках, коли в атрибуті зовнішнього ключа дочірнього відношення може знаходитись порожнє значення.

- SET DEFAULT. При видаленні рядка з батьківського відношення в атрибут зовнішнього ключа всіх рядків дочірнього відношення, що посилаються на нього автоматично поміщається значення, вказане для цього атрибута як значення за замовчуванням. Таким чином, видалення рядка з батьківського відношення викликає розміщення значення, приймається за замовчуванням в атрибут зовнішнього ключа всіх рядків дочірнього відношення, що посилаються на видалений рядок. Ця стратегія може бути застосована тільки в тих випадках, коли атрибуту зовнішнього ключа дочірнього відношення призначено деяке значення, прийняте за замовчуванням.

- NO CHECK. При видаленні рядка з батьківського відношення ніяких дій по збереженню посилальної цілісності даних не робиться.

Випадок 6. Оновлення первинного ключа в рядку батьківського відношення (Факультет). Якщо значення первинного ключа деякого рядка батьківського відношення буде оновлено, порушення посилальної цілісності матиме місце в тому випадку, якщо в дочірньому відношенні існують рядки, що посилаються на початкове значення первинного ключа.

Для збереження посилальної цілісності може використовуватися будь-яка з описаних вище стратегій. При використанні стратегії CASCADE оновлення значення первинного ключа в рядку батьківського відношення буде відображено в будь-якому рядку дочірнього відношення, що посилається на даний рядок (каскадним чином).

Вимоги даного підприємства

На закінчення потрібно проаналізувати обмеження, які називаються обмеженнями підприємства (або бізнес-правилами).

Наприклад, оновлення сутностей може регламентуватися прийнятими на підприємстві правилами, що описують методи виконання транзакцій, пов'язаних з подібними оновленнями.

Документування всіх обмежень цілісності даних

Помістіть відомості про всі встановлені обмеження цілісності даних в словник даних. Вони будуть потрібні на етапі фізичної реалізації бази даних.

Етап 2.7. Обговорення розроблених локальних логічних моделей даних з кінцевими користувачами

Мета: переконатися, що створені локальні моделі даних точно відображають представлення користувачів про предметну область додатку.

На даний момент робота над локальними моделями даних, що відбивають представлення конкретних користувачів про роботу підприємства, повинна бути закінчена і повністю відображена в документації. Однак перш ніж другий етап розробки можна буде вважати повністю завершеним, необхідно обговорити з користувачами створені логічні моделі даних і всю супровідну документацію.

Перш ніж перейти до третього етапу, передбаченому обговорюваною методологією проектування баз даних, коротко познайомимось з корисним додатковим методом перевірки локальних логічних моделей даних, що виконується за допомогою діаграм потоків даних.

Взаємозв'язок між логічними моделями даних і діаграмами потоків даних

Логічна модель даних відображає структуру даних підприємства, що зберігаються. Діаграма потоків даних (Data Flow Diagram - DFD) відображає переміщення даних в межах підприємства і розміщення їх в сховища інформації. Всі атрибути, які зберігаються на підприємстві, повинні бути оголошені в одному з типів сутностей і, ймовірно, знайти своє місце в потоках даних, які переміщуються в межах підприємства. Якщо обидві ці технології використовуються для моделювання вимог користувачів, кожна з них може застосовуватися для контролю узгодженості та повноти іншої. Правила, які визначають відношення між цими двома технологіями, такі:

- кожне сховище даних має представляти всі безліч множини сутностей;
- атрибути в потоках даних повинні належати сутності того чи іншого типу.

Етап 3. Створення і перевірка глобальної логічної моделі даних

Мета: об'єднання окремих локальних логічних моделей даних в єдину глобальну логічну модель даних, що представляє ту частину підприємства, яка охоплюється даним додатком.

На даному етапі фази логічного проектування баз даних будується глобальна логічна модель даних, створювана за допомогою злиття окремих логічних моделей даних, що відбивають представлення кожного користувача. По завершенні об'єднання локальних моделей необхідно перевірити правильність отриманої глобальної моделі, як щодо правил нормалізації, так і по відношенню до можливості виконання транзакцій, передбачених специфікаціями на функції окремих користувачів. Перевірка виконується з використанням тих же методів, які застосовувалися при виконанні етапів 2.3 і 2.4.

Однак проведення нормалізації потрібно тільки в тому випадку, якщо в процесі злиття були внесені зміни до складу окремих типів сутностей. Аналогічно, перевірка можливості виконання транзакцій виконується тільки для тих областей моделі, які були піддані змінам в ході злиття. У великих системах подібний підхід дозволяє істотно скоротити обсяг необхідних повторних перевірок.

Хоча кожна окрема логічна модель даних передбачається коректною, повною і несуперечливою, будь-яка з них відображає лише сприйняття системи користувачем або групою користувачів. Іншими словами, кожна модель відображає не функції підприємства, а лише представлення про функції підприємства окремих його співробітників. Тому будь-яка з цих моделей є неповною. А це означає, що між окремими моделями в повному наборі представлень можуть існувати несумісність і взаємне перекриття. Отже, при злитті локальних моделей даних в єдину глобальну модель доведеться докладати зусиль для усунення конфліктів між окремими представленнями і брати до уваги їх можливе перекриття.

Процедуру злиття можна вважати найважливішою в ході логічного проектування бази даних, оскільки з її допомогою створюється представлення про підприємство, яке не залежить від будь-яких конкретних користувачів, бізнес-процесів або додатків.

Даний етап передбачає виконання таких дій:

Етап 3.1. Злиття локальних логічних моделей даних в єдину глобальну модель даних.

Етап 3.2. Перевірка глобальної логічної моделі даних.

Етап 3.3. Перевірка можливостей розширення моделі в майбутньому.

Етап 3.4. Створення остаточного варіанту діаграми "сутність-зв'язок".

Етап 3.5. Обговорення глобальної логічної моделі даних з користувачами.

Етап 3.1. Злиття локальних логічних моделей даних в єдину глобальну модель даних

Мета: об'єднати окремі локальні логічні моделі даних в єдину глобальну логічну модель даних підприємства.

У невеликих системах, які налічують два-три користувальницьких представлення з незначною кількістю типів сутностей і зв'язків, завдання порівняння локальних моделей з подальшим злиттям і усуненням будь-яких можливих протиріч є відносно нескладною. Однак у великих системах потрібно використовувати більш систематичний підхід. Нижче вашій увазі запропоновано один з подібних підходів, який можна використовувати для виконання злиття локальних моделей і вирішення будь-яких виникаючих при цьому проблем. Пропонований підхід передбачає виконання таких дій:

1. Аналіз імен сутностей і їх первинних ключів.
2. Аналіз імен зв'язків.
3. Злиття загальних сутностей з окремих локальних моделей.
4. Включення (без злиття) сутностей, унікальних для кожного локального представлення.
5. Злиття загальних зв'язків з окремих локальних моделей.

6. Включення (без злиття) зв'язків, унікальних для кожного локального представлення.
7. Перевірка на наявність пропущених сутностей і зв'язків.
8. Перевірка коректності зовнішніх ключів.
9. Перевірка дотримання обмежень цілісності.
10. Виконання креслення глобальної логічної моделі даних.
11. Оновлення документації.

Ймовірно, найпростіший метод злиття декількох локальних моделей даних в єдину модель складається в злитті двох локальних моделей в одну загальну модель, з подальшим додаванням до неї третьої локальної моделі (і т.д.). Процес додавання до попереднього результату чергової локальної моделі буде продовжуватися до тих пір, поки всі моделі не будуть злиті в єдину глобальну модель. Цей підхід можна вважати більш простим, ніж спроба злити всі локальні моделі за одну операцію.

Дуже важливо відзначити, що, перш ніж приступати до створення глобального представлення інформаційної моделі підприємства, слід переконатися, що кожен з продуктів, які зливаються, локальних логічних моделей був створений в повній відповідності з послідовністю дій, передбачених першим і другим етапами розробки, визначеними в обговорюваній методології проектування баз даних.

1. Аналіз імен сутностей і їх первинних ключів

Може виявитися корисним попередньо проаналізувати імена сутностей присутніх в локальних моделях даних, - ці відомості можна знайти в словнику даних. Проблеми мають місце в наступних випадках:

- якщо дві або більше сутностей мають одне і те ж ім'я, але насправді відрізняються одна від одної;
- якщо дві або більше сутностей ідентичні, але мають різні імена.

Для виявлення можливих проблем слід порівняти між собою склади даних сутностей кожного типу. Зокрема, виявити еквівалентні сутності різними іменами допоможе порівняння їх первинних ключів.

2. Аналіз імен зв'язків

Виконувати дії аналогічні описаним на попередньому етапі.

3. Злиття загальних сутностей з окремих локальних моделей

Тепер слід проаналізувати імена і вміст сутностей кожного типу, присутніх у логічних моделях, що зливаються. Зазвичай ця процедура включає наступні дії:

- злиття сутностей з однаковими іменами та первинними ключами;
- злиття сутностей з однаковими іменами, але з різними первинними ключами;
- злиття сутностей з різними іменами, що мають однакові або різні первинні ключі.

Злиття сутностей з однаковими іменами та первинними ключами.

Як правило, сутності з одним і тим же первинним ключем представляють один і той же об'єкт реального світу і, отже, повинні бути злиті. Об'єднана

сутність включатиме всі атрибути, сутностей, які зливаються, за винятком тих, що дублюються.

Наприклад, в лістингу 5.1 перераховані атрибути, пов'язані з двома сутностями, що носять ім'я Викладач, які визначені в двох різних представленнях, названих View 1 і View 2. В обох, випадках первинним ключем є атрибут Табельний номер. Злиття цих сутностей здійснюється шляхом об'єднання їх атрибутів, тому в зведеній сутності Викладач будуть присутні всі атрибути вихідних сутностей. Відзначимо, що в даному випадку має місце конфлікт, пов'язаний зі способом представлення в кожній з вихідних сутностей імені викладача. У подібній ситуації, перш ніж вибрати необхідний спосіб представлення конфліктних даних, слід (по можливості) проконсультуватися з користувачами кожного з вихідних представлень. Припустимо, що в прикладі вибрано розгорнуте представлення атрибута ПІБ, тобто поділ його в зведеній сутності на атрибути Прізвище та Ім'я_По батькові.

Лістинг 5.1. Злиття сутностей Викладач з представлень View1 і View2

(Представлення View1)

Викладач (Табельний номер, ПІБ, Вчений ступінь, Код кафедри)

Primary Key Табельний номер

Foreign Key Код кафедри references Кафедра (Код кафедри)

(Представлення View2)

Викладач (Табельний номер, Прізвище, Ім'я_По батькові, Адреса, Телефон, Код кафедри)

Primary Key Табельний номер

Foreign Key Код кафедри references Кафедра (Код кафедри)

(Глобальне представлення)

Викладач (Табельний номер, Прізвище, Ім'я_По батькові, Вчений ступінь, Адресу, Телефон, Код кафедри)

Primary Key Табельний номер

Foreign Key Код кафедри references Кафедра (Код кафедри)

Злиття сутностей з однаковими іменами, але з різними первинними ключами.

У деяких ситуаціях можуть бути виявлені дві сутності з одним і тим же ім'ям, в яких використовуються різні первинні ключі, однак є однакові потенційні ключі. В цьому випадку сутності зливаються аналогічно тому, як це було зроблено в попередньому варіанті. Додатково буде потрібно вибрати в результуючій сутності первинний ключ, оголосивши всі інші ключі альтернативними. У лістингу 5.2 перераховані атрибути двох сутностей Викладач, визначених у двох різних представленнях під іменами View 1 і View 2. Первинним ключем сутності Викладач з представлення View 1 є атрибут ПІБ, а первинним ключем сутності Викладач з представлення View 2 - атрибут Табельний номер. Однак, в сутності Викладач з представлення View 1 визначено

альтернативний ключ Табельний номер, а в сутності Викладач з представлення View 2 - альтернативний ключ (Прізвище, Ім'я_По батькові). Хоча первинні ключі в кожній з вихідних сутностей різні, первинний ключ сутності Викладач з представлення View 1 є альтернативним ключем сутності Викладач з представлення View 2, і навпаки. Сутність, отримана в результаті злиття двох вихідних сутностей, аналогічна результуючій сутності, представлений в лістингу 5.1, проте в ній слід додатково визначити альтернативний ключ (FName, LName).

Лістинг 5.2. Злиття еквівалентних сутностей з різними первинними ключами

(Представлення View1)

Викладач (Табельний номер, ПІБ, Вчений ступінь, Код кафедри)

Primary Key ПІБ

Alternate Key Табельний номер

Foreign Key Код кафедри references Кафедра (Код кафедри)

(Представлення View2)

Викладач (Табельний номер, Прізвище, Ім'я_По батькові, Адреса, Телефон, Код кафедри)

Primary Key Табельний номер

Alternate Key Прізвище, Ім'я_По батькові

Foreign Key Код кафедри references Кафедра (Код кафедри)

(Глобальне представлення)

Викладач (Табельний номер, Прізвище, Ім'я_По батькові, Вчена ступінь, адресу, телефон, Код кафедри)

Primary Key Табельний номер

Alternate Key Прізвище, Ім'я_По батькові

Foreign Key Код кафедри references Кафедра (Код кафедри)

Злиття сутностей з різними іменами, що мають однакові або різні первинні ключі.

У деяких випадках можна виявити сутності, які мають різні імена, але призначені для однієї і тієї ж мети. Подібні еквівалентні сутності можна розпізнати за їхніми іменами, які будуть вказувати на їх подібне призначення, по їх змісту і по їх первинним ключам. Крім того, їх можна розпізнати по участі в певних зв'язках. Типовим прикладом подібної ситуації є наявність в моделях сутностей з назвами Персонал і Викладач, які, по суті, є еквівалентними і повинні бути злиті в єдину сутність.

4. Включення (без злиття) сутностей, унікальних для кожного локального представлення

На попередньому етапі були виділені всі сутності, що описують подібні об'єкти. Всі інші сутності просто включаються в глобальну модель без внесення будь-яких змін.

5. Злиття загальних зв'язків з окремих локальних моделей

На цьому етапі аналізуються імена і призначення кожного із зв'язків у всіх представленнях окремих користувачів. Перш ніж об'єднувати зв'язки, дуже важливо вирішити будь-які конфлікти, які можуть мати місце між ними, - наприклад, щодо обмежень участі або кардинальності. Виконувати на цьому етапі дії включають злиття зв'язків з однаковими іменами і призначенням, після чого може знадобитися виконати злиття зв'язків з різними іменами, але які мають одне і те ж призначення.

6. Включення (без злиття) зв'язків, унікальних для кожного локального представлення

На попередньому етапі були виявлені і злиті всі зв'язки, що мали подібне призначення (за визначенням, ці зв'язки повинні існувати між одними і тими ж сутностями, які також повинні бути злиті один з одним). Всі решта зв'язки включаються в глобальну модель без будь-яких змін.

7. Перевірка на наявність пропущених сутностей і зв'язків

Ймовірно, одним з найважливіших завдань при створенні глобальної моделі даних є завдання виявлення пропущених сутностей і зв'язків між елементами представлень різних користувачів. Якщо на підприємстві існує корпоративна модель даних, вона може використовуватися для виявлення сутностей і зв'язків між елементами представлень різних користувачів, яких немає ні в одному з локальних представлень. У той же час, при проведенні опитувань користувачів конкретного представлення в якості превентивного заходу слід попросити їх приділити належну увагу сутностям і зв'язкам, які, на їхню думку, можуть існувати в інших представленнях. Крім того, при аналізі атрибутів сутностей кожного типу можна спробувати виділити посилання на сутності, що належать іншим призначенням для користувача представленням. Досить часто виявляється, що атрибут, пов'язаний з тією чи іншою сутністю в поданні одного користувача, відповідає первинному ключу, альтернативному ключу або навіть простому, не ключовому атрибуту деякої сутності з іншого представлення.

8. Перевірка коректності зовнішніх ключів

На цьому етапі може здійснюватися злиття різних сутностей і зв'язків, зміна первинних ключів і установка нових зв'язків. Переконайтеся, що зовнішні ключі в дочірніх сутностях як і раніше є коректними, і в разі потреби внесіть в модель всі необхідні зміни.

9. Перевірка дотримання обмежень цілісності

Переконайтеся, що встановлені для глобальної логічної моделі обмеження цілісності даних не вступають в протиріччя з тими обмеженнями, які були встановлені для кожного з призначених для користувача представлень. Будь-які конфлікти слід усувати за допомогою проведення консультацій з користувачами.

10. Виконання креслення глобальної логічної моделі даних

На цьому етапі малюється остаточний варіант ER-діаграми, що представляє глобальну логічну модель даних, отриману в результаті злиття всіх локальних моделей.

11. Оновлення документації

Оновлення документації виконується з метою відображення будь-яких змін, що вносяться в процесі створення глобальної логічної моделі даних з набору окремих користувальницьких представлень. Дуже важливо, щоб документація завжди підтримувалася в актуальному стані і точно відображала поточний стан моделі даних. Якщо згодом у модель будуть вноситися зміни, або в ході фізичної реалізації бази даних, або в процесі її супроводу, всі внесені зміни також повинні негайно фіксуватися в документації. Застаріла документація часто є джерелом безлічі прикрих помилок і служить причиною прийняття неправильних рішень.

Етап 3.2. Перевірка глобальної логічної моделі даних

Мета: перевірка глобальної логічної моделі даних за допомогою методів нормалізації та контроль можливості виконання необхідних транзакцій.

На цьому етапі проводяться дії, аналогічні тим, які виконувалися на етапах 2.3 і 2.4 при перевірці кожної з локальних логічних моделей даних.

Етап 3.3. Перевірка можливостей розширення моделі в майбутньому

Мета: визначення ймовірності внесення будь-яких істотних змін у створену модель даних в найближчому майбутньому і оцінка того, наскільки дана модель пристосована для цього.

Дуже важливо, щоб створена глобальна модель була легко розширювана. Якщо модель зможе підтримувати тільки поточні вимоги, то час її існування буде досить обмеженим, а для реалізації підтримки нових або тих, які змінюються вимог буде потрібно докладати значних зусиль. Необхідно так побудувати модель, щоб вона була легко розширювана і дозволяла реалізувати підтримку нових вимог з мінімальними змінами в роботі вже існуючих користувачів.

Отже, має сенс виконати перевірку створеної моделі з точки зору ефективності реалізації нових вимог, які можуть мати місце в майбутньому. Однак не слід вносити в модель будь-яких змін, поки вони не будуть схвалені користувачами.

Етап 3.4. Створення остаточного варіанту діаграми "сутність-зв'язок"

Мета: створення остаточного варіанта діаграми "сутність-зв'язок", що відображає глобальну логічну модель даних підприємства.

Завершивши всі перевірки створеної глобальної логічної моделі, можна приступити до підготовки остаточного варіанту ER-діаграми. Ця діаграма повинна представляти глобальну логічну модель даних тієї частини підприємства, яка моделюється в цьому додатку. Документація, що описує цю

модель (включаючи схему відношень словник даних) повинна бути оновлена і підготовлена в повному обсязі.

Етап 3.5. Обговорення глобальної логічної моделі даних з користувачами

Мета: переконатися, що створена глобальна логічна модель даних адекватно відображає частину інформаційної структури підприємства, що моделюється.

Глобальна логічна модель даних підприємства до цього моменту повинна бути повністю завершена і перевірена. Сама модель і документація, що до неї додається надаються для перегляду та аналізу кінцевим користувачам, які повинні переконатися, що вона точно відображає структуру і функціонування підприємства.

5.4 Методологія фізичного проектування бази даних

У цьому розділі буде описана методологія фізичного проектування реляційної бази даних. За відправну точку проведеного обговорення є глобальна логічна модель даних і супровідна документація, створені на першому-третьому етапах запропонованої методології проектування баз даних. Їх створення починалося з перетворення локальних концептуальних моделей даних для кожного представлення на етапі 1 і підготовки локальних логічних моделей даних для кожного представлення на етапі 2, які згодом використовувалися для розробки набору відповідних відношень. Отримані набори відношень були перевірені на відповідність вимогам правил нормалізації і можливість виконання необхідних транзакцій. Етап логічного проектування бази даних завершився процедурою з'єднання на етапі 3 локальних логічних моделей даних, що відбивають представлення окремих користувачів, в єдину глобальну логічну модель даних, яка відповідає всім призначеним для користувача представленням організації. Безумовно, що етап 3 повинен виконуватися, тільки якщо є кілька користувальницьких представлень.

Відповідно до запропонованої методології, в ході третьої і останньої стадії методології розробки бази даних розробник повинен прийняти рішення про те, як перетворити логічний проект бази даних (тобто сукупність сутностей, атрибутів, зв'язків і встановлених обмежень) в проект фізичної бази даних, яка реалізується в середовищі обраної цільової СУБД. Оскільки багато аспектів фізичного проектування баз даних залежать від типу обраної цільової СУБД, можливе існування декількох способів реалізації будь-якого заданого елемента бази даних. Цілком очевидно, що розробник повинен добре знати функціональні можливості обраної цільової СУБД, а також чітко розуміти всі її переваги і недоліки, що дозволить йому приймати обґрунтовані рішення при виборі того чи іншого методу реалізації бази даних. Крім того, в кожному конкретному випадку розробник повинен вміти вибрати оптимальну стратегію розміщення і зберігання даних.

Порівняння етапів логічного і фізичного проектування баз даних.

Стадія, що передує *фізичному проектуванню*, є стадією *логічного проектування*. Результати її виконання в значній мірі незалежні від особливостей фізичної реалізації проекту. При логічному проектуванні не беруться до уваги конкретні функціональні можливості цільової бази даних і прикладних програм, однак враховуються особливості обраної моделі даних. Результатом логічного проектування є глобальна логічна модель даних, що складається з ER-діаграми або діаграми відношень, а також реляційної схеми, і комплект, що описує її супровідної документації, що включає, зокрема, словник даних. У сукупності ці результати є вихідною інформацією для стадії фізичного проектування бази даних і надають її розробнику все необхідне для прийняття рішень, спрямованих на досягнення максимальної ефективності створюваного проекту.

Образно кажучи, при логічному проектуванні розробник в основному розглядає, *що* має бути зроблено, а при фізичному проектуванні він шукає спосіб, *як це* зробити. У кожному випадку потрібна наявність різних навичок, якими найчастіше володіють різні фахівці. Так, фахівець з фізичного проектування баз даних повинен чітко уявляти, як функціонує в комп'ютерній системі та чи інша СУБД, а також добре знати всі функціональні можливості цільової СУБД. Оскільки функціональні можливості різних СУБД досить сильно відрізняються один від одного, фізичне проектування завжди тісно пов'язане з особливостями конкретної обраної системи. Однак етап фізичного проектування бази даних не є абсолютно ізольованим від інших. Як правило, між фізичним, логічним проектуванням і розробкою додатків завжди є зворотний зв'язок. Наприклад, рішення, прийняті на етапі фізичного проектування з метою підвищення продуктивності системи (зокрема, щодо об'єднання відношень), можуть вплинути на структуру логічної моделі даних, а це може позначитися на проектах додатків.

Етап 4. Перенесення глобальної логічної моделі даних в середовище цільової СУБД

Мета: створення базової функціональної схеми реляційної бази даних на основі глобальної логічної моделі даних, яка може бути реалізована в цільовій СУБД.

Найпершим завданням на етапі фізичного проектування баз даних є перетворення відношень, створених на основі глобальної логічної моделі даних, в таку форму, яка може бути реалізована в середовищі цільової СУБД. Перша частина цього процесу передбачає перевірку інформації, зібраної на етапі логічного проектування бази даних та вміщеній в словник даних. Друга частина процесу полягає в використанні цієї інформації для розробки проекту основних відношень. Цей процес вимагає наявності глибоких знань про функціональні можливості, що надаються цільовою СУБД. Зокрема, розробник повинен знати наступне:

- способи створення основних відношень;
- чи підтримує система визначення первинних, зовнішніх і альтернативних ключів;

- чи підтримує система визначення обов'язкових даних (тобто чи допускає система вказувати у визначенні атрибута, що для нього заборонено використання значення NULL);
- чи підтримує система визначення доменів;
- чи підтримує система реляційні обмеження цілісності;
- чи підтримує система визначення обмежень предметної області.

На етапі 4 процедури розробки баз даних виконуються наступні дії:

Етап 4.1. Проектування основних відношень.

Етап 4.2. Розробка способів отримання похідних даних.

Етап 4.3. Реалізація обмежень предметної області.

Етап 4.1. Проектування основних відношень

Мета: визначення способу представлення в цільовій СУБД відношень, визначених у глобальній логічній моделі даних.

Приступаючи до фізичного проектування, перш за все, необхідно проаналізувати і добре засвоїти інформацію про відношення, зібрану на етапі побудови логічної моделі бази даних. Ця інформація може міститися в словнику даних і в визначеннях відношень, записаних на мові DBDL. Визначення кожного виділеного в глобальній логічній моделі даних відношення включає наступні елементи:

- ім'я відношення;
- список простих атрибутів, взятий у круглі дужки;
- визначення первинного ключа і (якщо такі існують) альтернативних (АК) і зовнішніх (ФК) ключів;
- список похідних атрибутів і опис способів їх обчислення;
- визначення вимог посилальної цілісності для будь-яких зовнішніх ключів.

Для кожного атрибута в словнику даних повинна бути наступна інформація:

- визначення його домену, що включає вказівку типу даних, розмірність внутрішнього представлення атрибута і будь-які необхідні обмеження на допустимі значення;

- прийняте за замовчуванням значення атрибута (необов'язково);
- допустимість значення NULL для даного атрибута.

При створенні проекту основних відношень використовується розширений формат виразів мови DBDL, що дозволяє вказувати домени, прийняті за замовчуванням значення і індикатори допустимості значення NULL. Як приклад в лістингу 5.3 наведено визначення відношення Книги (читацький зал).

Лістинг 5.3. Опис відносини Книги (читацький зал) на мові DBDL

Domain Коди книг:	integer, in the range 0001-9999
Domain Назви книг:	variable length character string, length 25
Domain Видавництва книг:	variable length character string, length 25
Domain Кількість сторінок:	integer, in the range 1-999
Domain Номери залів:	integer, in the range 1-30
Книги (читацький зал) (	

Код книги	Коди книг NOT NULL,
Назва книги	Назви книг NOT NULL,
Видавництво	Видавництва книг NOT NULL,
Кількість сторінок	Кількість сторінок NOT NULL,
Номер залу	Номери залів

PRIMARY KEY (Код книги),
FOREIGN KEY (Номер залу) REFERENCES Зал (Номер залу) ON UPDATE
CASCADE ON DELETE SET NULL);

Реалізація основних відношень

Тепер необхідно прийняти рішення про спосіб реалізації основних відношень. Це рішення залежить від типу обраної цільової СУБД - при визначенні основних відношень деякі системи надають більше можливостей, ніж інші.

Документальне оформлення проекту основних відношень

Підготовлений проект основних відношень повинен бути детально описаний в супровідній документації із зазначенням причин, за якими був обраний даний конкретний проект. Зокрема, необхідно вказати, чому був обраний саме цей підхід, якщо є цілий ряд інших варіантів.

Етап 4.2. Розробка способів отримання похідних даних

Мета: визначення способу представлення в цільовій СУБД всіх похідних даних, які включені в глобальну логічну модель даних.

Похідними, або **розрахунковими** називаються *атрибути, значення яких можна визначити з використанням значень інших атрибутів*. Наприклад, похідними є всі перераховані нижче атрибути:

- кількість студентів, які навчаються на третьому курсі;
- загальна сума щомісячної зарплати всіх викладачів;
- кількість гуртожитків, виділених для даного факультету.

Іноді похідні атрибути не включають в логічну модель даних, а описують в словнику даних. А якщо похідний атрибут включений в модель, для вказівки на те, що він є похідним, перед його ім'ям ставиться коса риска (/). На першому етапі проектування вивчається логічна модель даних і словника даних, а також готується список всіх похідних атрибутів. На етапі фізичного проектування бази даних необхідно визначити, чи повинен похідний атрибут зберігатися в базі даних або обчислюватися кожен раз, коли в ньому виникає необхідність. Проектувальник повинен розрахувати наступне:

- додаткові витрати на зберігання похідних даних та підтримання їх узгодженості з реальними даними, на основі яких вони обчислюються;
- витрати на обчислення похідних даних, якщо їх обчислення виконується в міру необхідності.

З цих двох варіантів вибирається найменш дорогий з урахуванням вимог до продуктивності. В останньому з перерахованих вище прикладів може бути прийнято рішення про зберігання в відношенні Викладач додаткового атрибута, що позначає кількість курсових робіт, якими керує викладач в даний час.

Додаткові витрати пам'яті для цього нового похідного атрибута не надто великі. Значення атрибута оновлюється кожного разу, коли викладач починає

керувати новою курсовою роботою, або припиняє керувати. У будь-якому з цих випадків значення атрибута Кількість курсових для відповідного викладача повинно бути збільшена або зменшена на 1. В процесі проектування необхідно забезпечити, щоб ці зміни відбувалися в кожному із зазначених випадків і кількість об'єктів, що враховуються залишалися правильними, оскільки це гарантує цілісність бази даних. При зверненні до цього атрибуту за допомогою будь-якого запиту його значення є безпосередньо доступним і не вимагає обчислення. З іншого боку, якщо цей атрибут не зберігався б безпосередньо в відношенні Викладач, то його значення доводилося б обчислювати кожен раз, коли воно буде потрібно. Для цього необхідно виконати з'єднання відношення Викладач і Курсова робота. Таким чином, якщо запит зазначеного типу виконується часто або вважається дуже важливим з точки зору продуктивності, похідний атрибут більш доцільно зберігати в базі даних, а не обчислювати при кожному зверненні до його значення.

Рішення, що передбачає зберігання похідних атрибутів, є більш прийнятним і в тому випадку, якщо мова запитів цільової СУБД не дозволяє легко реалізувати алгоритм обчислення похідних атрибутів. Наприклад, в мові SQL є лише обмежений набір функцій агрегування, який не дозволяє легко реалізувати рекурсивні запити.

Документальне оформлення проекту отримання похідних даних

Спосіб отримання похідних даних повинен бути повністю описаний в документації із зазначенням причин вибору запропонованого проекту. Зокрема, необхідно перерахувати всі причини, за якими було обрано саме цей підхід, якщо є цілий ряд інших варіантів.

Етап 4.3. Реалізація обмежень предметної області

Мета: реалізація обмежень предметної області для цільової СУБД.

Оновлення інформації в відношеннях може регламентуватися обмеженнями предметної області, що регулюють виконання тих реальних транзакцій, які пов'язані з проведенням таких оновлень. Спосіб реалізації зазначених обмежень знову-таки буде залежати від типу обраної цільової СУБД, оскільки одні системи для реалізації обмежень предметної області надають ширші можливості, ніж інші. Як і на попередньому етапі, якщо цільова СУБД підтримує стандарт мови SQL, то реалізувати певні типи обмежень буде набагато простіше. Наприклад, для заданого навчального процесу існує правило, згідно з яким кожен викладач може одночасно керувати не більше ніж дев'ятьма курсовими роботами. Це обмеження можна включити в оператор CREATE TABLE мови SQL для таблиці Курсова робота за допомогою наступної конструкції:

CONSTRAINT Табельний номер (обмеження по керівництву)

```
CHECK (NOT EXIST (SELECT Табельний номер
                    FROM Курсова робота
                    GROUP BY Табельний номер
                    HAVING COUNT (*) > 9))
```

Альтернативним методом реалізації обмежень є застосування тригерів. У деяких системах відсутня підтримка частини або навіть всіх обмежень

предметної області і тому такі обмеження доводиться передбачати безпосередньо в самому додатку. Наприклад, далеко не всі реляційні СУБД (якщо взагалі такі є) дозволяють реалізувати тимчасові обмеження.

Документальне оформлення проекту реалізації обмежень предметної області

Всі рішення, прийняті в зв'язку з реалізацією обмежень предметної області, повинні бути повністю описані в супровідній документації. Крім того, в документації повинні бути вказані причини вибору саме даного варіанта з декількох можливих.

Етап 5. Проектування фізичного представлення бази даних

Мета: визначення оптимальної файлової організації для зберігання основних відношень, а також індексів, необхідних для досягнення прийнятної продуктивності; іншими словами, визначення способу зберігання відношень і рядків у зовнішній пам'яті.

Однією з найважливіших цілей фізичного проектування бази даних є організація ефективного зберігання даних. Існує кілька показників, які можуть бути використані для оцінки досягнутої ефективності.

- *Продуктивність виконання транзакцій.* Цей показник являє собою кількість транзакцій, які можуть бути оброблені за заданий інтервал часу. У деяких системах, наприклад в службах резервування авіаквитків, забезпечення високої продуктивності виконання транзакцій є вирішальним фактором успішної експлуатації всієї системи.

- *Час відповіді.* Характеризує часовий проміжок, необхідний для виконання однієї транзакції. З точки зору користувача бажано зробити час відповіді системи мінімальним. Однак існують деякі фактори, які впливають на швидкість системи, але не можуть контролюватися розробниками, наприклад, рівень завантаження системи або час, що витрачається на передачу даних.

- *Дискова пам'ять.* Цей показник являє собою обсяг дискового простору, необхідного для розміщення файлів бази даних. Розробник повинен прагнути мінімізувати обсяг використовуваної дискової пам'яті.

Однак жоден з цих чинників не є самодостатнім. Як правило, розробник змушений шукати компроміс між цими показниками для досягнення прийнятного балансу. Наприклад, збільшення обсягу збережених даних може викликати збільшення часу відповіді системи або зменшення продуктивності виконання транзакцій. Початковий варіант фізичного проекту бази даних не слід розглядати як щось незмінне, а потрібно застосовувати як засіб оцінки можливого рівня продуктивності системи. Після реалізації початкового варіанту проекту необхідно вести спостереження за показниками роботи системи і відповідно до отриманих результатів виконувати її налаштування з метою поліпшення показників роботи і враховуючи мінливі вимоги користувачів. Багато типів СУБД надають в розпорядження адміністратора бази даних (Data Base Administrator - DBA) комплект утиліт, призначений для поточного контролю над функціонуванням системи і її налаштуванням. Пізніше дізнаємося, що існують певні структури організації зовнішньої пам'яті, що дозволяють ефективно завантажувати в базу великі обсяги даних, але мало

придатні для інших цілей. Іншими словами, спочатку має сенс вибрати такі структури зберігання даних, які будуть дуже ефективні при масовому завантаженні даних в процесі створення бази, після чого їх можна буде замінити іншими структурами, що дозволяють ефективно її експлуатувати.

І знову-таки діапазон вибору можливих типів організації файлів залежить від цільової СУБД, оскільки різні системи підтримують різні набори допустимих структур зберігання інформації. Дуже важливо, щоб розробник фізичного проекту бази даних мав повне представлення про всі типи структур зберігання даних, підтримуваних цільовою СУБД, а також про всі особливості використання цих структур в системі. Зокрема, бажано, щоб розробник ясно розумів принципи роботи оптимізатора запитів системи. Наприклад, можуть виникнути ситуації, в яких оптимізатор запитів не використовуватиме індекси, навіть якщо вони доступні. В результаті просте додавання індексу не дозволить підвищити ефективність обробки запитів, а лише викличе додаткове непотрібне навантаження на систему.

Визначення поняття системних ресурсів

Щоб досягти високої продуктивності системи, розробник фізичного проекту бази даних повинен знати, яким чином взаємодіють між собою і впливають на продуктивність системи наступні чотири основні компоненти апаратних засобів.

- *Оперативна пам'ять.* Доступ до даних в оперативній пам'яті здійснюється набагато (в десятки або навіть в сотні і тисячі разів) швидше, ніж до даних у зовнішній пам'яті. У загальному випадку, чим більший обсяг оперативної пам'яті доступний для СУБД і додатків бази даних, тим швидше виконуються програми. Досвід показує, що корисно постійно підтримувати в системі такий режим, при якому близько 5% її оперативної пам'яті залишається вільною. Однак нерозумно підтримувати рівень вільної пам'яті вище 10%, оскільки в цьому випадку оперативна пам'ять буде використовуватися неефективно. Якщо в системі не вистачає оперативної пам'яті для задоволення потреб всіх процесів, операційна система звільняє частину цієї пам'яті, передаючи окремі сторінки пам'яті деяких процесів на диск. Ці сторінки будуть прочитані з диска, як тільки знову буде потрібно доступ до даних, які в них містяться. Така операція називається *свопінгом*, або *сторінковим обміном*. У певних випадках для отримання необхідного обсягу вільної пам'яті системі доводиться переміщати на диск всі сторінки пам'яті, відведені цілому процесу, а потім повертати їх назад. Але якщо інтенсивність свопінга (сторінкового обміну) стає занадто високою, це вказує на брак оперативної пам'яті в системі.

- *Процесор.* Управляє функціонуванням інших апаратних компонентів системи і підтримує призначені для користувача процеси. Однією з умов ефективної роботи цього компонента є запобігання конкуренції за право його використання, що зазвичай супроводжується перекладом процесів в стан очікування. Процесор стає вузьким місцем системи в тих випадках, якщо операційна система або програми користувачів створюють дуже велике

навантаження на процесор. Така ситуація часто виникає в результаті різкого зростання інтенсивності сторінкового обміну.

- *Дисковий ввід-вивід.* У будь-якій досить потужній СУБД процеси збереження і вибірки даних пов'язані з виконанням безлічі дискових операцій вводу-виводу. Як правило, виробники дискових пристроїв вказують рекомендовану кількість операцій введення-виведення в секунду. Якщо реальний показник перевищує дане значення, дискова підсистема перетворюється в вузьке місце системи. На загальну продуктивність дискової пам'яті дуже великий вплив справляє спосіб організації зберігання даних. Рекомендується рівномірно розподіляти збережені дані між усіма доступними в системі пристроями, що знижує ймовірність появи проблем. На рис. 9.9 проілюстровано приклад реалізації перерахованих нижче основних принципів розподілу даних по дисковим пристроїв.

- Файли операційної системи повинні бути відокремлені від файлів бази даних.
- Основні файли бази даних повинні бути відокремлені від індексних файлів.
- Журнал відновлення повинен бути відділений від іншої частини бази даних.

- *Мережа.* У мережі виконуються розподілені по різних її вузлах запити (або група запитів), звані транзакціями. Мережа може стати вузьким місцем всієї системи при надмірному зростанні мережевого трафіку або великій кількості мережевих колізій. Кожен з цих ресурсів здатний впливати на інші системні ресурси. Тому поліпшення показників використання одного ресурсу може позитивно позначитися на стані всіх інших системних ресурсів, наприклад:

- збільшення обсягу оперативної пам'яті викликає зниження інтенсивності сторінкового обміну і, як наслідок, зменшення навантаження на процесор;
- більш ефективне використання оперативної пам'яті призводить до зменшення кількості операцій дискового вводу-виводу.

Взявши до уваги все вищесказане, приступимо до обговорення тих дій, які повинні бути виконані на п'ятому етапі розробки бази даних.

Етап 5.1. Аналіз транзакцій.

Етап 5.2. Вибір файлової структури.

Етап 5.3. Визначення індексів.

Етап 5.4. Визначення вимог до дискової пам'яті.

Етап 5.1. аналіз транзакцій

Мета: визначення функціональних характеристик транзакцій, які будуть виконуватися в базі даних, що проектується, і виділення найбільш важливих з них.

Для того щоб розроблюваний фізичний проект бази даних володів необхідним рівнем ефективності, необхідно отримати максимум відомостей про ті транзакції і запити, які будуть виконуватися в базі даних. Будуть потрібні як

якісні, так і кількісні характеристики. Для успішного планування кожної транзакції необхідно знати наступне:

- транзакції, що виконуються найбільш часто і роблять істотний вплив на продуктивність;
- транзакції, найбільш важливі для роботи організації;
- періоди часу протягом діб/тижнів, в які навантаження бази даних зростає до максимуму (звані *періодами пікового навантаження*).



Рис. 5.10. Типова схема розподілення файлів по дисках

Ця інформація використовується для визначення компонентів бази даних, які можуть викликати проблеми продуктивності. Крім того, необхідно визначити такі характеристики транзакцій високого рівня, як атрибути, які модифікуються в транзакціях поновлення, або критерії, які служать для обмеження кількості рядків, що повертаються за запитом. Ця інформація використовується для визначення найбільш відповідної файлової організації і створення індексів.

У багатьох випадках проаналізувати всі очікувані транзакції просто неможливо, тому необхідно тим чи іншим чином вибрати найбільш "важливі" з них (це слово взято в лапки, бо критерії вибору найчастіше є суб'єктивними). Існує емпіричне правило, згідно з яким виконання близько 20% найбільш активних запитів користувачів створює приблизно 80% загального навантаження на базу даних. Це правило "80/20" може використовуватися як рекомендація щодо проведення аналізу. Для визначення того, які з транзакцій підлягають детальному аналізу, скористаємося таблицею відповідності транзакцій і відношень, в якій показані відношення, доступ до яких відбувається при виконанні кожної транзакції, а також діаграмою частоти виконання транзакцій, яка схематично показує відношення, ймовірність використання яких в транзакціях найбільш висока. Для виділення областей, які з найбільшою ймовірністю можуть бути джерелом проблем, необхідно виконати такі дії.

- Підготовка схеми відповідності шляхів виконання транзакцій і відношень.

- Визначення відношень, найбільш часто використовуваних при виконанні транзакцій.

- Аналіз інтенсивності доступу до даних в деяких з транзакцій, що використовують ці відношення.

Підготовка схеми відповідності шляхів виконання транзакцій і відношень

Етапи 1.8, 2.4 і 3.2 методології концептуального/логічного проектування бази даних передбачають оцінку моделей даних для визначення того, чи підтримують вони всі транзакції, необхідні для роботи користувачів; для цього встановлюється відповідність між шляхами виконання транзакцій і сутностями/відношеннями. З іншого боку, якщо перевірка транзакцій здійснювалася якимось іншим способом, може знадобитися підготувати таблицю відповідності транзакцій і відношень. Ця таблиця наочно показує, які транзакції потрібні в додатку і до яких відношень вони звертаються.

Таблиця стає ще більш зручною, якщо в кожній її клітинці зазначено кількість випадків доступу протягом деякого інтервалу часу (наприклад, за годину, на добу або на тиждень).

Можна зробити висновок, що потрібно більш уважне вивчення і самих транзакцій, і цих відносин.

Отримання інформації про частоту використання відношень в транзакціях

При вивченні кожної транзакції необхідно знати не тільки середню і максимальну частоту її виконання на годину, але також визначити, в які дні та години виконується ця транзакція і, зокрема, коли ймовірність пікового навантаження найбільш велика.

Якщо для деяких транзакцій потрібно частий доступ до певних відношень, то набуває особливої важливості завдання вивчення характеру їх виконання. Наприклад, якщо в певний час потрібно тільки одні транзакції, а потім інші, то ризик виникнення проблем, пов'язаних зі зниженням продуктивності, зменшується. А якщо періоди максимальної частоти застосування різних транзакцій збігаються, потенційні проблеми продуктивності можна вирішити, більш ретельно вивчаючи такі транзакції для визначення того, які зміни можуть бути внесені в структуру відношень для підвищення продуктивності. Ще одне рішення може передбачати зміну графіка виконання деяких транзакцій таким чином, щоб вони не створювали додаткове навантаження в періоди інтенсивного виконання інших транзакцій (наприклад, іноді є можливість запланувати проведення деяких підсумкових транзакцій на більш відповідний час, скажімо, виконувати їх ввечері або вночі).

Аналіз використання даних

Після визначення найбільш важливих транзакцій детально аналізується кожна з них. Для кожної транзакції необхідне з'ясувати наступне:

- Відношення і атрибути, до яких здійснюється доступ в процесі виконання транзакції, а також тип доступу; це означає визначення того, чи виконується в цій транзакції вставка, оновлення, видалення або вибірка даних (транзакції останнього типу називаються також *запитами*).

- При вивченні транзакції оновлення необхідно визначити, які атрибути оновлюються в даній транзакції, оскільки ці атрибути можуть зажадати застосування допоміжних структур доступу (таких як додаткові індекси).

- Атрибути, які використовуються в будь-яких предикатах (в мові SQL *предикатами* є умови, зазначені в конструкції WHERE). Перевірка того, чи передбачають ці предикати наступне:

- зіставлення з шаблоном, наприклад `name LIKE '% Сидоров%'`;

- пошук в діапазоні, наприклад salary BETWEEN 4000 AND 5000;
 - вибірка по точному значенням ключа, наприклад salary = 6000.
 - Ця задача повинна бути виконана не тільки по відношенню до запитів, але і до транзакцій оновлення та видалення, оскільки в них також може бути обмежена кількість рядків, оновлюваних або тих, що видаляються в деякому відношенні.
 - Слід враховувати, що такі атрибути можуть зажадати створення допоміжних структур доступу.
 - Запити, в яких використовуються атрибути, які беруть участь в з'єднанні двох або декількох відношень.
 - Ці атрибути також можуть зажадати застосування допоміжних структур доступу.
 - Очікувана частота виконання транзакції; наприклад, може бути встановлено, що транзакція виконується приблизно 50 разів на добу.
 - Встановлені показники продуктивності для транзакції; наприклад, вимога, щоб транзакція виконувалася протягом 1 секунди.
- Найбільший пріоритет при визначенні допоміжних структур доступу повинні мати атрибути, які застосовуються у всіх предиката найбільш часто, або найбільш важливі транзакції.

Етап 5.2. Вибір файлової структури

Мета: визначення найбільш ефективного файлового представлення для кожного з основних відношень.

Одним із головних завдань фізичного проектування бази даних є забезпечення ефективного зберігання даних. Наприклад, якщо вибірка рядків з даними про викладачів повинна виконуватися по іменах в алфавітному порядку, то найбільш придатною структурою для зберігання цих даних є файл, відсортований за іменами викладачів. А якщо має виконуватися вибірка даних про всіх викладачів, зарплата яких знаходиться в певному діапазоні, файл, відсортований за іменами викладачів, не підходить для виконання такого завдання. Становище ще більше ускладнюється у зв'язку з тим, що деякі способи організації файлів добре підходять для масового завантаження даних в базу даних, але їх застосування в подальшому стає неефективним. Іншими словами, завдання полягає в використанні ефективної структури зберігання даних на зовнішніх пристроях, яка забезпечує швидке завантаження бази даних, а потім її перетворення в структуру, більш підходящу для повсякденної експлуатації.

Тому мета справжнього етапу полягає у визначенні оптимальної файлової організації для кожного відношення, якщо цільова СУБД це дозволяє. У багатьох випадках реляційна СУБД може надавати лише обмежений вибір або навіть взагалі не дозволяти вибрати певну файлову організацію, але в деяких СУБД на організацію розміщення даних на зовнішніх пристроях можна певною мірою вплинути шляхом задання індексів. Проте, в даній темі наведені рекомендації щодо вибору файлової організації на основі типів файлів, перерахованих нижче. Це зроблено для того, щоб читач міг краще зрозуміти особливості файлової організації та використання індексів.

- Послідовні файли.

- Хешовані файли.
- Індексно-послідовні файли (ISAM - Indexed Sequential Access Method).
- Вдосконалені збалансовані дерева (B + -Tree).
- Кластери.

Якщо цільова СУБД не дозволяє вибрати певну файлову організацію, цей етап може бути пропущений.

Послідовні (невпорядковані) файли

Послідовний файл є найбільш зручною структурою для зберігання даних наступних випадках:

1. Дані завантажуються в таблицю великими блоками. Наприклад, при заповненні новоствореної таблиці в неї одночасно може завантажуватися велика кількість даних. Якщо для новоствореної таблиці як вихідний тип файлової організації була обрана послідовна організація, то реструктуризація файлу таблиці після завантаження основного обсягу даних може зробити помітний вплив на ефективність її використання.

2. Весь файл таблиці займає всього кілька сторінок. В цьому випадку час пошуку будь-якого запису буде невелика, навіть при послідовному перегляді всіх рядків таблиці.

3. При кожному зверненні до таблиці вибірці підлягають всі її рядки (в будь-якому порядку). Приклад - вибірка адрес викладачів, які стосуються певної кафедри.

4. Таблиця має додаткові структури пошуку, наприклад індекс по ключу. У цьому випадку використання файлів послідовної організації дозволяє домогтися економії дискового простору.

Файли послідовної організації неефективні, якщо доступ виконується тільки до деяких рядків таблиці.

Хешовані файли

Застосування хешованого файлу в якості структури організації пам'яті для таблиці доцільно в тих випадках, коли вибір рядків здійснюється по точному значенні поля, використаного для хешування, особливо якщо доступ до рядків відбувається випадковий чином. Наприклад, якщо виконати хешування таблиці Книги (читацький зал) по атрибуту Кількість сторінок, вибірка даних за принципом "Кількість сторінок дорівнює 100" буде ефективною. Хешовані файли не рекомендується використовувати в наступних випадках:

1. Вибірка рядків з таблиці здійснюється шляхом зіставлення з шаблоном ключа хешованого поля.

2. Вибірка рядків з таблиці здійснюється по заданому діапазону значень поля, яке входить в значення поля хешування.

3. Вибірка рядків з таблиці здійснюється за значенням поля, відмінного від поля хешування.

4. Доступ до рядків необхідно виконувати тільки за частиною поля хешування.

5. Часто відбувається оновлення поля хешування. При виконанні такої операції в СУБД повинний бути вилучений весь рядок і в разі необхідності розміщений за новою адресою (якщо в результаті обчислення за допомогою

хеш-функції формується нова адреса). Тому часте оновлення поля хешування призводить до зниження продуктивності.

Індексно-послідовні файли

У порівнянні з хешуванням метод файлу з індексно-послідовної організації (ISAM) являє собою більш гнучку структуру зберігання даних. Він підтримує вибірку даних за точним збігом значення ключа, за шаблоном підстановки, за діапазоном значень і по частині основного ключа. Однак структура індексу файлу ISAM залишається незмінною після її формування при створенні самого файлу. Тому продуктивність доступу до даних файлу ISAM знижується по мірі відновлення його даних. Оновлення також можуть викликати порушення послідовності ключів файлу ISAM, тому вибірка даних в порядку значень ключів все більше і більше сповільнюється. Дві останні проблеми усуваються при використанні файлів, організованих за принципом збалансованого дерева. Однак на відміну від збалансованих дерев, конкурентний доступ до індексу файлу ISAM легко забезпечується, оскільки його індекс залишається незмінним.

Збалансовані дерева

У порівнянні з хешованими файлами збалансовані дерева (B + -Tree) також представляють собою значно більш гнучкі структури зберігання даних. Вони дозволяють виконувати вибірку даних за точним збігом ключового значення, за шаблоном підстановки, за діапазоном значень і по частково заданому ключу. Індекс збалансованих дерев є динамічним, що збільшується в міру зростання файлу таблиці. Завдяки цьому, на відміну від файлів ISAM, ефективність доступу в збалансованих деревах не знижується по мірі відновлення даних таблиці. Файли структури B + -Tree постійно зберігають впорядкованість доступу по ключу, навіть при оновленні їх даних. Тому швидкість вибірки рядків в порядку значень ключа тут вище, ніж у файлів ISAM, і є постійною. Але якщо інформація в таблиці не піддається постійним змінам, то використання структури збалансованого дерева може виявитися менш ефективною в порівнянні з індексно-послідовними файлами. Справа в тому, що індекси файлів ISAM є послідовними, а файлів збалансованого дерева - багаторівневими (в яких лист-вузли містять вказівники на фізичні рядки, а не самі рядки).

Кластеризовані таблиці

Деякі СУБД, наприклад Oracle, підтримують кластеризовані таблиці. Необхідність використання кластеризованих таблиць обумовлена тим, як відбувається доступ до таблиць, об'єднаним в кластер. Така інформація може бути отримана на основі попереднього аналізу транзакцій. Але слід враховувати, що необгрунтоване рішення по об'єднанню таблиць в кластер може привести до зниження продуктивності. Рекомендації по використанню кластеризованих таблиць наведені нижче. Слід зазначити, що в даному розділі застосовується термінологія компанії Oracle, в якій *відношення* розглядається як *таблиця зі стовпцями і рядками*.

Кластери являють собою групи з однієї або декількох таблиць, які фізично зберігаються разом, оскільки мають загальні стовпці і часто використовуються одночасно. При спільному фізичному зберіганні взаємопов'язаних рядків скорочується час доступу до диска. Взаємопов'язані стовпці таблиць в кластері

називаються *ключем кластера*. Ключ кластера зберігається тільки в одному екземплярі, тому організація зберігання набору таблиць в кластері є більш ефективною в порівнянні з роздільним зберіганням тих же таблиць (не об'єднаних в кластер). СУБД Oracle підтримує кластери двох типів: *індексовані* і *хешовані*.

А. Індексовані кластери

У індексованому кластері рядки з однаковим ключем кластера зберігаються разом. Корпорація Oracle рекомендує використовувати індексовані кластери за таких умов:

- в запитах найчастіше відбувається вибірка рядків в діапазоні значень ключа кластера;

- кластеризовані таблиці можуть рости непередбачуваним чином.

При визначенні того, чи слід застосовувати кластеризовані таблиці, можна скористатися рекомендаціями, наведеними нижче:

- Передбачити можливість кластеризації таблиць, доступ до яких часто відбувається в операціях з'єднання.

- Не кластеризувати таблиці, якщо їх з'єднання відбувається тільки час від часу або часто оновлюються значення в загальних стовпцях. (Модифікація значення ключа кластера в рядку вимагає більше часу в порівнянні з модифікацією значення в некластеризованій таблиці, оскільки в СУБД Oracle може знадобитися перемістити модифікований рядок в інший блок в процесі оновлення даних в кластері).

- Не кластеризувати таблиці, якщо часто потрібне виконання повного пошуку в одній з таблиць. (Повний пошук в кластеризованій таблиці може вимагати більше часу в порівнянні з повним пошуком в некластеризованій таблиці. В СУБД Oracle зростає ймовірність того, що при виконанні цієї операції доведеться прочитати більше блоків, оскільки таблиці зберігаються разом).

- Передбачити кластеризацію таблиць, що беруть участь в зв'язку "один до багатьох", якщо часто відбувається вибірка рядку з батьківської таблиці, а потім - відповідних рядків з дочірньою таблиці. (Дочірні рядки зберігаються в тому ж блоці (блоках) даних, що і батьківський рядок, тому велика ймовірність того, що вони вже будуть перебувати в оперативній пам'яті, коли до них знадобиться доступ, тому в СУБД зменшується загальний обсяг операцій вводу-виводу).

- Передбачити зберігання в кластері тільки дочірньої таблиці, якщо після вибірки одного батьківського рядка вибирається багато дочірніх рядків. (Це дозволяє підвищити продуктивність запитів, в яких вибираються дочірні рядки, відповідні одному батьківському рядку, і разом з тим не знижує продуктивність повного пошуку в батьківській таблиці).

- Не кластеризувати таблиці, якщо рядки всіх таблиць з однаковим значенням ключа кластера займають більше одного або двох блоків Oracle. (Для доступу до рядка кластеризованої таблиці СУБД Oracle зчитує всі блоки, які містять рядки з цим значенням. Тому якщо рядки займають кілька блоків, для доступу до одного рядка може знадобитися виконати більше операцій читання, ніж при здійсненні доступу до того ж рядка в некластеризованій таблиці).

Б. Хешовані кластери

Хешовані кластери також дозволяють кластеризувати дані таблиць у формі, аналогічній індексованим кластерам. Але вибір місця зберігання рядка в хешованому кластері здійснюється з урахуванням результатів застосування хеш-функції до значення ключа кластера розглянутого рядка. Всі рядки з однаковим значенням ключа хешу зберігаються на диску разом. Корпорація Oracle рекомендує використовувати хешовані кластери за таких умов:

- в запитах вибірка рядків відбувається з використанням умов рівності, в яких враховуються всі стовпці ключа кластера;
- Групові таблиці змінюються рідко, або ще до створення таблиці можна визначити максимальну кількість рядків і максимальний обсяг простору, який потрібно для кластера.

При визначенні доцільності використання хешованих кластерів можна скористатися рекомендаціями, наведеними нижче:

- Передбачити можливість застосування хешованих кластерів для зберігання таблиць, доступ до яких часто відбувається з використанням критеріїв пошуку, що містять умови рівності, в яких розглядається один і той же стовпець (стовпці). Цей стовпець (стовпці) повинен бути оголошений як ключ кластера.

- Зберігати таблицю в хешованому кластері, якщо можна легко визначити, який обсяг простору потрібно для зберігання всіх рядків з певним значенням ключа кластера, як в даний час, так і в майбутньому.

- Не використовувати хешовані кластери, якщо простір обмежений і немає можливості відвести додатковий простір для рядків, які можуть бути вставлені в подальшому.

- Не використовувати хешований кластер для зберігання постійно зростаючої таблиці, якщо процес періодичного створення нового, більшого хешованого кластера для зберігання цієї таблиці неможливо здійснити на практиці.

- Не зберігати таблицю в хешованому кластері, якщо часто потрібен пошук у всій таблиці і для хешованого кластера повинен бути відведений значний обсяг простору з урахуванням необхідності подальшого зростання таблиці. (В подібних випадках повний пошук вимагає читання всіх блоків, розподілених для хешованого кластера, навіть незважаючи на те, що деякі блоки можуть містити лише невелику кількість рядків. Тому окреме зберігання такої таблиці дозволяє зменшити кількість блоків, зчитуваних при повному пошуку в таблиці).

- Не зберігати таблицю в хешованому кластері, якщо значення ключа кластера часто змінюються.

- Іноді може бути виправдане зберігання в хешованому кластері навіть однієї таблиці, незалежно від того, чи часто відбувається з'єднання цієї таблиці з іншими таблицями, але за умови, що наведені вище рекомендації показують, що для цієї таблиці доцільно застосовувати хешування.

Документальне оформлення результатів вибору структури файлів таблиць

Результати вибору файлової структури для всіх таблиць повинні бути ретельно зафіксовані в документації. У кожному випадку слід вказати причини

зробленого вибору. Зокрема, обов'язково вкажіть причини вибору конкретного варіанту, якщо існував альтернативний варіант (один або кілька).

Етап 5.3. Визначення індексів

Мета: визначення того, чи буде додавання індексів сприяти підвищенню продуктивності системи.

Один з варіантів вибору відповідної файлової структури для відношення полягає в тому, що рядки залишаються невідсортованими і створюється будь-яка необхідна кількість додаткових індексів. Ще один варіант передбачає упорядкування рядків у відношенні з використанням первинного або кластеризованого індексу. В цьому випадку для вибору атрибута, за яким виконується впорядкування або кластеризації рядків, застосовуються такі критерії:

- вибирається атрибут, найбільш часто вживаний в операціях з'єднання, оскільки саме він дозволяє підвищити ефективність таких операцій;
- вибирається атрибут, найбільш часто вживаний для доступу до рядків в відношенні з урахуванням послідовності значень цього атрибута.

Якщо обраний атрибут, за яким відбувається впорядкування, є ключем відношення, то створюваний індекс служить в якості первинного індексу, а якщо атрибут, за яким відбувається впорядкування, не є ключем, то створюваний індекс служить кластеризованим індексом. Слід враховувати, що в кожному відношенні повинен застосовуватися або первинний, або кластеризований індекс.

Індекс зазвичай створюється засобами мови SQL; для цього застосовується оператор CREATE INDEX. Наприклад, для створення первинного індексу на відношенні Книги (читацький зал) з урахуванням атрибута Код книги можна використовувати наступний оператор SQL:

```
CREATE UNIQUE INDEX Код_книгиInd ON Книги (читацький зал) (Код книги);
```

Для створення кластеризованого індексу на відношенні Книги (читацький зал) з урахуванням атрибута Номер залу служить наступний оператор SQL:

```
CREATE INDEX Номер_залуInd ON Книги (читацький зал) (Номер залу) CLUSTER.
```

Як було зазначено вище, в деяких системах файлова структура є заздалегідь визначеною. Наприклад, до недавнього часу в СУБД Oracle підтримувалися тільки збалансовані дерева, але, як зазначено вище, тепер в цій СУБД передбачена підтримка кластерів. З іншого боку, така СУБД, як INGRES, підтримує широкий набір різних індексних структур, які можна вибрати шляхом зазначення наступної необов'язковою конструкції в операторі CREATE INDEX:

```
[STRUCTURE = BTREE | ISAM | HASH | HEAP]
```

Вибір додаткових індексів

Додаткові індекси надають можливість визначати додаткові ключі для базового відношення, які можуть застосовуватися для підвищення ефективності вибірки даних.

Супровід і застосування додаткових індексів приводить до збільшення витрат, тому необхідно визначити, чи виправдовують вони підвищення

продуктивності при вибірці даних, що досягається завдяки їх використанню. Основні витрати, пов'язані із застосуванням додаткових індексів, перераховані нижче:

- Введення індексного запису в кожен додатковий індекс при вставці рядка в відношення.
- Оновлення додаткового індексу при оновленні відповідного рядка в відношенні.
- Збільшення потреби в дисковому просторі у зв'язку з необхідністю поранення додаткового індексу.
- Можливе зниження продуктивності процесу оптимізації запитів, оскільки оптимізатор запитів повинен врахувати наявність всіх додаткових індексів і тільки після цього вибрати оптимальну стратегію виконання запитів.

Рекомендації по вибору списку вимог до індексів

Один із способів визначення необхідної кількості додаткових індексів полягає в підготовці списку вимог до атрибутів, які розглядаються для визначення необхідності застосування їх для індексації, а потім вивченні витрат на супровід кожного з цих індексів. Нижче наведені рекомендації по підготовці такого списку вимог.

1. Не створювати індекс на невеликих відношеннях. Може виявитися більш ефективним пошук в відношенні, дані якого зберігаються в оперативній пам'яті, ніж зберігати додаткову індексний структуру.

2. Як правило, слід створювати індекс на первинному ключі відношення, якщо він не застосовується в якості ключа файлової структури. Хоча в стандарті SQL передбачена конструкція, що дозволяє задати специфікацію первинного ключа, слід зазначити, що застосування цієї конструкції не завжди гарантує створення індексу на первинному ключі.

3. Ввести додатковий індекс на зовнішньому ключі, якщо з його допомогою часто відбувається доступ до відношення. Але слід враховувати, що в деяких СУБД індекси на зовнішніх ключах створюються автоматично.

4. Ввести додаткові індекси на всіх атрибутах, які часто застосовуються в якості додаткового ключа.

5. Ввести додаткові індекси на атрибутах, які часто застосовуються в наступних конструкціях:

- критерії вибірки або з'єднання;
- конструкції ORDER BY;
- конструкції GROUP BY;
- інші операції, що вимагають сортування (такі як UNION і DISTINCT).

6. Ввести додаткові індекси на атрибутах, які застосовуються у вбудованих функціях агрегування, поряд з усіма атрибутами, використовуваними для цих вбудованих функцій. Наприклад, щоб визначити середню зарплату викладачів кожної кафедри, можна застосувати наступний запит SQL:

```
SELECT Код кафедри, AVG (salary)
FROM Викладач
GROUP BY Код кафедри;
```

Згідно з наведеною вище рекомендацією, можна передбачити створення індексу на атрибуті Код кафедри, оскільки він бере участь в конструкції GROUP BY. Але

більш ефективне рішення може передбачати створення індексу і на атрибуті Код кафедри, і на атрибуті salary. Це дозволяє виконати в СУБД весь запит за даними тільки з самого індексу, без необхідності доступу до файлу даних. Такий план виконання запиту іноді називають планом, що передбачає використання тільки індексу, оскільки необхідні результати можуть бути отримані за допомогою тільки даних, що містяться в індексі.

7. Узагальнюючи наведену вище рекомендацію, можна вказати, що необхідно вводити додатковий індекс на всіх атрибутах, які можуть привести до створення плану, що передбачає застосування тільки індексів.

8. Не індексувати атрибут або відношення, які часто оновлюються.

9. Не індексувати атрибут, якщо в запитах з використанням цього атрибута зазвичай відбувається вибірка значної частини (наприклад, 25%) рядків кортежів у відношенні. В такому випадку може виявитися більш ефективним пошук у всьому відношенні, ніж пошук з використанням індексу.

10. Не індексувати атрибути, які складаються з довгих символьних рядків.

Якщо в умовах пошуку передбачено кілька предикатів і одна з цих умов містить конструкцію OR, а в самій умові не застосовується індекс і не передбачається сортування, то додавання індексів для інших атрибутів не дозволяє підвищити швидкість виконання такого запиту, оскільки все одно буде потрібний послідовний пошук в відношенні.

З іншого боку, якщо предикати в конструкції WHERE з'єднані за допомогою оператора AND, навіть ці два індекси можуть застосовуватися для оптимізації запиту.

Видалення індексів зі списку вимог

Після підготовки списку вимог до потенційних індексів необхідно врахувати вплив кожного з них на транзакції оновлення. Якщо потреба в супроводі індексу може привести до уповільнення важливих транзакцій оновлення, слід передбачити можливість виключення цього індексу зі списку. Але необхідно враховувати, що визначені індекси можуть також сприяти підвищенню ефективності операції оновлення.

Доцільно по можливості провести експерименти для визначення того, чи сприяє створення індексу підвищення продуктивності, майже не впливає на продуктивність або призводить до її зниження. Якщо буде виявлено зниження продуктивності, цей індекс, безумовно, повинен бути вилючений зі списку вимог. А якщо в результаті введення додаткового індексу спостерігається лише незначне підвищення продуктивності, може знадобитися подальше дослідження для визначення того, за яких обставин цей індекс може виявитися корисним і чи так часто виникають ці обставини, щоб створення індексу дійсно було виправданим.

Деякі системи дозволяють вивчати стратегію, обрану оптимізатором для виконання конкретного запиту або оновлення; така стратегія називається планом виконання запиту (Query Execution Plan - QEP). Наприклад, в СУБД Microsoft Access передбачена програма Performance Analyzer, в СУБД Oracle - діагностична утиліта EXPLAIN PLAN, в СУБД DB2 - утиліта EXPLAIN, а в СУБД INGRES - інтерактивний засіб перегляду плану виконання запиту. Якщо запит виконується повільніше, ніж очікувалося, має сенс скористатися такою

утилітою для визначення причин уповільнення і знайти іншу стратегію, що дозволяє підвищити продуктивність запиту.

Якщо у відношеннях з одним або декількома індексами відбувається вставка великої кількості рядків, може виявитися більш ефективним рішення спочатку видалити індекси, виконати вставку, а потім знову створити індекси. В якості емпіричного правила можна вказати, що якщо в результаті вставки загальний обсяг даних в відношенні збільшується щонайменше на 10%, доцільно видалити на певний час індекси цього відношення.

Оновлення статистичної інформації в базі даних

Оптимізатор запитів для вибору оптимальної стратегії використовує статистичну інформацію бази даних, яка зберігається в системному каталозі. При створенні будь-якого індексу СУБД автоматично вводить в системний каталог інформацію про цей індекс. Але слід враховувати, що в деяких СУБД для оновлення в системному каталозі статистичної інформації, яка відноситься до певного відношення або індексу, необхідно викликати на виконання певну утиліту.

Документальне оформлення результатів вибору індексів

Вибрані індекси повинні бути повністю відображені в документації із зазначенням причин того, чому був зроблений саме цей вибір. Зокрема, якщо деякі атрибути були проіндексовані з урахуванням ймовірності зниження продуктивності, це також повинно бути вказано в документації.

Етап 5.4. Оцінка потреби в дисковому просторі

Мета: визначити обсяг дискового простору, який потрібен для бази даних.

У багатьох проектах висувається вимога, щоб фізична реалізація бази даних могла бути здійснена на основі існуючої конфігурації апаратних засобів. Але навіть при відсутності такої вимоги проектувальник повинен оцінити обсяг дискового простору, який потрібно для зберігання файлів бази даних, хоча б на той випадок, що в зв'язку з цим буде потрібно придбання нових апаратних засобів. Мета даного етапу полягає в оцінці обсягу дискового простору, необхідного для підтримки реалізації бази даних у зовнішній пам'яті. Як і на попередніх етапах, проведення оцінки потреби в дисковому просторі в значній мірі залежить від цільової СУБД і від апаратних засобів, які застосовуються для підтримки функціонування бази даних. Як правило, така оцінка заснована на інформації про середній розмір кожного рядка і кількості рядків в відношенні. Передня оцінка повинна показувати максимальну кількість, але може також виявитися доцільним проведення аналізу інтенсивності зростання розмірів відношення і коригування підсумкових відомостей про обсяг диска з урахуванням отриманого коефіцієнта зростання для визначення можливих розмірів бази даних в майбутньому.

Етап 6. Проектування користувацьких представлень

Мета: спроектувати користувацькі представлення, необхідність в створенні яких була виявлена на стадії збору і аналізу вимог, що доставляє частина життєвого циклу додатку реляційної бази даних.

Перший етап методології проектування бази даних передбачає підготовку локальних концептуальних моделей даних для кожного представлення, визначеного на стадії аналізу вимог до бази даних. Кожне представлення складається з одного або декількох користувальницьких представлень.

На етапі 2 ці локальні концептуальні моделі даних перетворюються в локальні логічні моделі даних, засновані на реляційній моделі, а для додатків з декількома представленнями на етапі 3 локальні моделі об'єднуються в єдину глобальну логічну модель даних. Призначення цього етапу полягає в проектуванні користувальницьких представлень, які були визначені раніше. В автономній СУБД на персональному комп'ютері користувача представлення зазвичай застосовуються тільки для зручності, оскільки вони часто дозволяють спростити запити до бази даних. Але в багатокористувацькій СУБД такі представлення грають головну роль при визначенні структури бази даних і дотриманні правил захисту.

Документальне оформлення проекту користувальницьких представлень

Проекти окремих користувальницьких представлень повинні бути повністю відображені в документації.

Етап 7. Проектування засобів захисту

Мета: проектування засобів захисту для бази даних відповідно до вимог користувачів.

База даних являє собою виключно важливий корпоративний ресурс і тому захист цього ресурсу має виняткове значення. На стадії збору і аналізу вимог, що становить частину життєвого циклу програми бази даних, повинні бути враховані і відображені в специфікації вимог до системи конкретні вимоги до захисту. Призначення цього етапу полягає у визначенні способів реалізації вимог до захисту. Склад засобів захисту залежить від конкретної системи. Тому проектувальник повинен знати, які можливості пропонує розглянута їм цільова СУБД. Реляційні СУБД, як правило, надають засоби захисту бази даних наступних типів:

- захист системи;
- захист даних.

Засоби захисту системи регламентують доступ і експлуатацію бази даних на рівні системи; до них, зокрема, відноситься автентифікація користувача по імені і пароллю. Засоби захисту даних регламентують доступ і використання об'єктів бази даних (таких як відношення та представлення), а також дії, які можуть бути виконані користувачами з конкретними об'єктами. І в цьому випадку проект реалізації правил доступу залежить від цільової СУБД, оскільки можливості реалізації правил доступу залежать від системи. Наприклад, можна привести три конкретних способи реалізації правил доступу з використанням стандарту ISO SQL, СУБД Microsoft Access і СУБД Oracle.

Документальне оформлення проекту реалізації засобів захисту

Проект реалізації засобів захисту повинен бути повністю відображений в документації. А якщо створений при цьому фізичний проект вимагає внесення змін в окремі локальні логічні моделі даних, необхідно внести поправки також і в схеми цих моделей.

Питання:

- 5.1. У чому полягає загальне призначення пропонованої методології проектування?
- 5.2. Які основні етапи проектування баз даних?
- 5.3. Вкажіть найважливіші фактори успішного завершення процесу розробки бази даних.
- 5.4. У чому полягає основна мета етапу концептуального проектування бази даних?
- 5.5. Які основні етапи концептуального проектування бази даних?
- 5.6. Як відбувається виявлення типів сутностей і зв'язків виходячи з специфікації вимог користувача?
- 5.7. Який спосіб застосовується для визначення атрибутів на основі специфікації вимог користувача, а потім - для визначення приналежності цих атрибутів до конкретних типів сутностей і зв'язків?
- 5.8. Опишіть спосіб визначення потенційних ключів і вибір первинного ключа.
- 5.9. Що розуміється під спеціалізацією або генералізацією типів сутностей?
- 5.10. Поясніть необхідність обговорення локальних концептуальних моделей даних з кінцевими користувачами.
- 5.11. У чому полягає основна мета етапу логічного проектування бази даних?
- 5.12. Які основні етапи логічного проектування бази даних?
- 5.13. Опишіть послідовність дій, які виконуються при перетворенні концептуальної моделі даних в логічну модель.
- 5.14. Назвіть правила утворення відношень, що становлять сильні сутності, слабкі сутності, бінарні зв'язку типу "один до одного" і типу "один до багатьох", множинні атрибути та зв'язку типу "суперклас/підклас".
- 5.15. Поясніть, як методи нормалізації можуть використовуватися для перевірки логічної моделі даних і набору відношень, створених на основі цієї моделі.
- 5.16. Опишіть два підходи, які можуть використовуватися для перевірки надання логічною моделлю можливостей виконання всіх необхідних транзакцій.
- 5.17. Поясніть призначення обмежень цілісності і назвіть п'ять основних типів подібних обмежень.
- 5.18. Опишіть всі існуючі типи стратегій, які можуть застосовуватися для обробки спроб видалення рядка батьківського відношення, на яке є посилання в дочірньому відношенні.
- 5.19. Назвіть типові задачі, які доводиться вирішувати при злитті локальних логічних моделей даних в єдину глобальну логічну модель.
- 5.20. Опишіть склад вхідної інформації і результати виконання процедури фізичного проектування бази даних.

- 5.21. Яке призначення основних етапів фізичного проектування бази даних, що визначаються розглянутою методологією.
- 5.22. Однією з найважливіших цілей етапів фізичного проектування бази даних є організація ефективного зберігання даних. Як можна виміряти досягнутий рівень ефективності в зазначеному контексті?

Вправи:

Виберіть будь-яку предметну область, крім навчального процесу, і створіть базу даних, виконуючи описані вище етапи методології проектування бази даних.

ТЕМА 6

Архітектура систем БД. СУБД

Основна мета системи управління базами даних (СУБД) полягає в тому, щоб *запропонувати користувачеві абстрактне уявлення даних, приховавши конкретні особливості зберігання та управління ними.*

Отже, відправною точкою при проектуванні бази даних повинний бути *абстрактний і загальний опис інформаційних потреб організації*, які повинні знайти своє відображення в створюваній базі даних.

Наприклад, для навчального процесу становить інтерес моделювання таких понять

- сутностей "реального світу", таких як Студент, Група, Викладач, Кафедра і Предмет;
- атрибутів, що описують властивості або якості кожної суті, наприклад, сутність Студент (Student) володіє атрибутами: ПІБ студента (FIO), вік (Age), адреса студента (Address), середній бал (Sr.Ball), номер групи (Group_No), номер Студента в групі (Student_No);
- зв'язків між цими сутностями, наприклад, кожен студент перебуває в певній групі і кожна група містить певного студента.

Більше того, оскільки база даних є загальним ресурсом, то кожному користувачеві може знадобитися своє, відмінне від інших, представлення про характеристики інформації, що зберігається в базі даних.

Для задоволення потреби колективного використання структур даних при їх індивідуальному представленні розроблена архітектура ANSI-SPARC, на основі якої в тій чи іншій мірі побудована архітектура більшості сучасних комерційних СУБД.

6.1 Трирівнева архітектура ANSI-SPARC

Перша спроба створення стандартної термінології та загальної архітектури СУБД була зроблена в 1971 році групою, з назвою DBTG (data base technology group).

Вона була створена після конференції CODASYL (Conference on Data System and Languages - Конференція з мов і системам, даних), що відбулася в тому ж році.

Група DBTG визнала необхідним використання дворівневого підходу, побудованого на основі використання системного представлення, тобто схеми (schema), і представлень для користувача, тобто підсхем (subscheme).

Подібна термінологія та архітектура були запропоновані в 1975 році Комітетом планування стандартів і норм SPARC (Standards Planning and Requirements Committee) Національного Інституту Стандартизації США, ANSI / X3 / SPARC (ANSI, 1975).

Комітет ANSI / SPARC визнав необхідність використання трирівневого підходу.

У цих матеріалах відображені пропозиції, які були зроблені організаціями GUIDE / SHARE, до яких входять користувачі продуктів корпорації IBM, і опубліковані за кілька років до цього. Основна увага в них було сконцентровано на необхідності втілення незалежного рівня для ізоляції програм від особливостей подання даних на більш низькому рівні (Guide / Share, 1970). Хоча модель ANSI / SPARC не стала стандартом, тим не менш, вона все ще являє собою основу для розуміння деяких функціональних особливостей СУБД.

Найбільш фундаментальним моментом у цих і наступних звітах дослідницьких груп є *ідентифікація трьох рівнів абстракції*, тобто трьох різних рівнів опису елементів даних. Ці рівні формують *трирівневу архітектуру*, яка охоплює зовнішній, концептуальний і внутрішній рівні, як показано на рис. 6.1.

Мета трирівневої архітектури полягає у відділенні представлення користувача бази даних від її фізичного представлення.

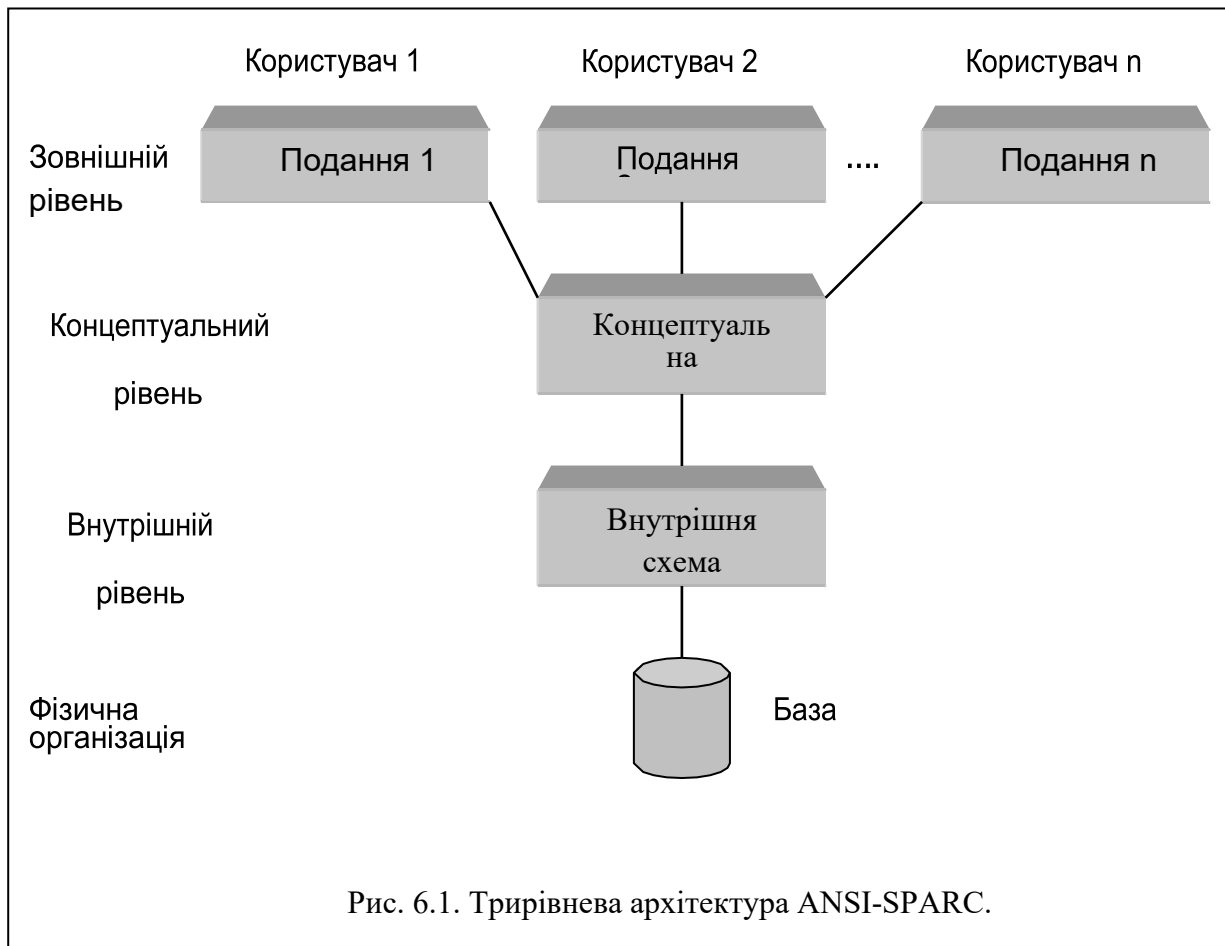
Причини поділу на три рівні;

- *Кожен користувач повинен мати можливість звертатися до одних і тих же даних, використовуючи своє власне представлення про них.* Кожен користувач повинен мати можливість змінювати своє представлення про дані, причому ця зміна не повинна впливати на інших користувачів.
- *Користувачі не повинні безпосередньо мати справу з такими деталями фізичного зберігання даних в базі, як індексування і хешування, тобто взаємодія користувача з базою не повинно залежати від особливостей розміщення в ній даних.*
- *Адміністратор бази даних (АБД) повинен мати можливість змінювати структуру зберігання даних в базі, впливаючи на представлення користувача.*
- *Внутрішня структура бази даних не повинна залежати від таких змін фізичних аспектів зберігання інформації, як перемикання на новий пристрій зберігання.*
- *АБД повинен мати можливість змінювати концептуальну або глобальну структуру бази даних без будь-якого впливу на будь-якого з користувачів.*

*Рівень, на якому сприймають дані користувачі, називається **зовнішнім рівнем (external level)**, тоді як СУБД і операційна система сприймають дані на **внутрішньому рівні (internal level)**.*

Саме на внутрішньому рівні дані реально зберігаються з використанням конкретних структур даних і файлової організації.

Концептуальний рівень (conceptual level) представлення даних призначений для відображення зовнішнього рівня на внутрішній і забезпечення необхідної незалежності їх один від одного.



Зовнішній рівень

Зовнішній рівень - *представлення бази даних з точки зору користувачів, описує ту частину бази даних, яка відноситься до кожного користувача.*

Зовнішній рівень складається з декількох різних зовнішніх представлень бази даних, має справу з представленням "реального світу", вираженим в найбільш зручній для нього формі.

Зовнішнє представлення містить тільки ті сутності, атрибути та зв'язки "реального світу", які цікаві користувачеві, а інші також можуть бути представлені в базі даних, але користувач може навіть не підозрювати про їх існування.

Крім цього, різні представлення можуть по-різному відображати одні й ті ж дані. Наприклад, один користувач може переглядати дати у форматі (день, місяць, рік) а інший - у форматі (рік, місяць, день).

Деякі зовнішні представлення можуть включати похідні або обчислювані дані, що не зберігаються в базі даних як такі, а створюються в міру потреби.

Представлення можуть також включати комбіновані або похідні дані з декількох об'єктів.

Концептуальний рівень

Концептуальний рівень - узагальнююче представлення бази даних, описує те, які дані зберігаються в базі даних, а також зв'язки, що існують між ними.

Проміжним рівнем в трирівневій архітектурі є концептуальний рівень. Цей рівень містить логічну структуру всієї бази даних (з точки зору

АБД), є повним представленням вимог до даних з боку організації, яке не залежить від будь-яких міркувань щодо способу їх зберігання.

На концептуальному рівні представлені наступні компоненти:

- всі сутності, їх атрибути і зв'язки;
- обмеження, що накладаються на дані;
- семантична інформація про дані;
- інформація про заходи безпеки і підтримки цілісності даних.

Концептуальний рівень підтримує кожне зовнішнє обмеження, в тому сенсі, що будь-які доступні користувачеві дані повинні міститися (або можуть бути обчислені) на цьому рівні.

Цей рівень не містить жодних відомостей про методи зберігання даних.

Наприклад, опис сутності повинен містити відомості про типи даних атрибутів (цілочисельний, дійсний або символьний) та їх довжини (кількості значущих цифр або максимальній кількості символів), але не повинен включати відомостей про організацію зберігання даних, наприклад про обсяг зайнятого простору в байтах.

Внутрішній рівень

Внутрішній рівень - це низькорівневе подання всієї бази даних як бази, що складається з деякої множини екземплярів кожного з існуючих типів внутрішніх записів.

Внутрішнє представлення, так само як зовнішнє і концептуальне, відокремлене від фізичного рівня, оскільки в ньому не розглядаються фізичні записи, зазвичай звані блоками або сторінками, і фізичні області пристрою зберігання, такі як циліндри і доріжки.

Блоки (або сторінки) пристрою введення-виведення - це кількість даних, переданих з вторинної пам'яті (пам'яті накопичувача) в основну (оперативну) пам'ять за одну операцію введення-виведення.

Іншими словами, внутрішнє представлення передбачає наявність нескінченного лінійного адресного простору.

Внутрішнє представлення описується за допомогою **внутрішньої схеми**. Яка визначає не тільки різні типи збережених записів, але також існуючі індекси, способи представлення збережених полів, фізичну впорядкованість збережених записів і т.д., **внутрішня схема** формується з використанням ще однієї мови визначення даних - внутрішньої.

Нижче внутрішнього рівня знаходиться **фізичний рівень (physical level)**, який контролюється операційною системою, але під керівництвом СУБД.

Незалежність від даних

Основним призначенням трирівневої архітектури є забезпечення незалежності від даних, яка означає, що зміни на нижніх рівнях ніяк не впливають на верхні рівні.

Розрізняють два типи незалежності від даних: логічну і фізичну.

Логічна незалежність від даних означає повну захищеність зовнішніх схем від змін, що вносяться до концептуальної схеми.

Такі зміни концептуальної схеми, як додавання або видалення нових сутностей, атрибутів або зв'язків, повинні здійснюватися без необхідності внесення змін у вже існуючі зовнішні схеми або переписування прикладних програм.

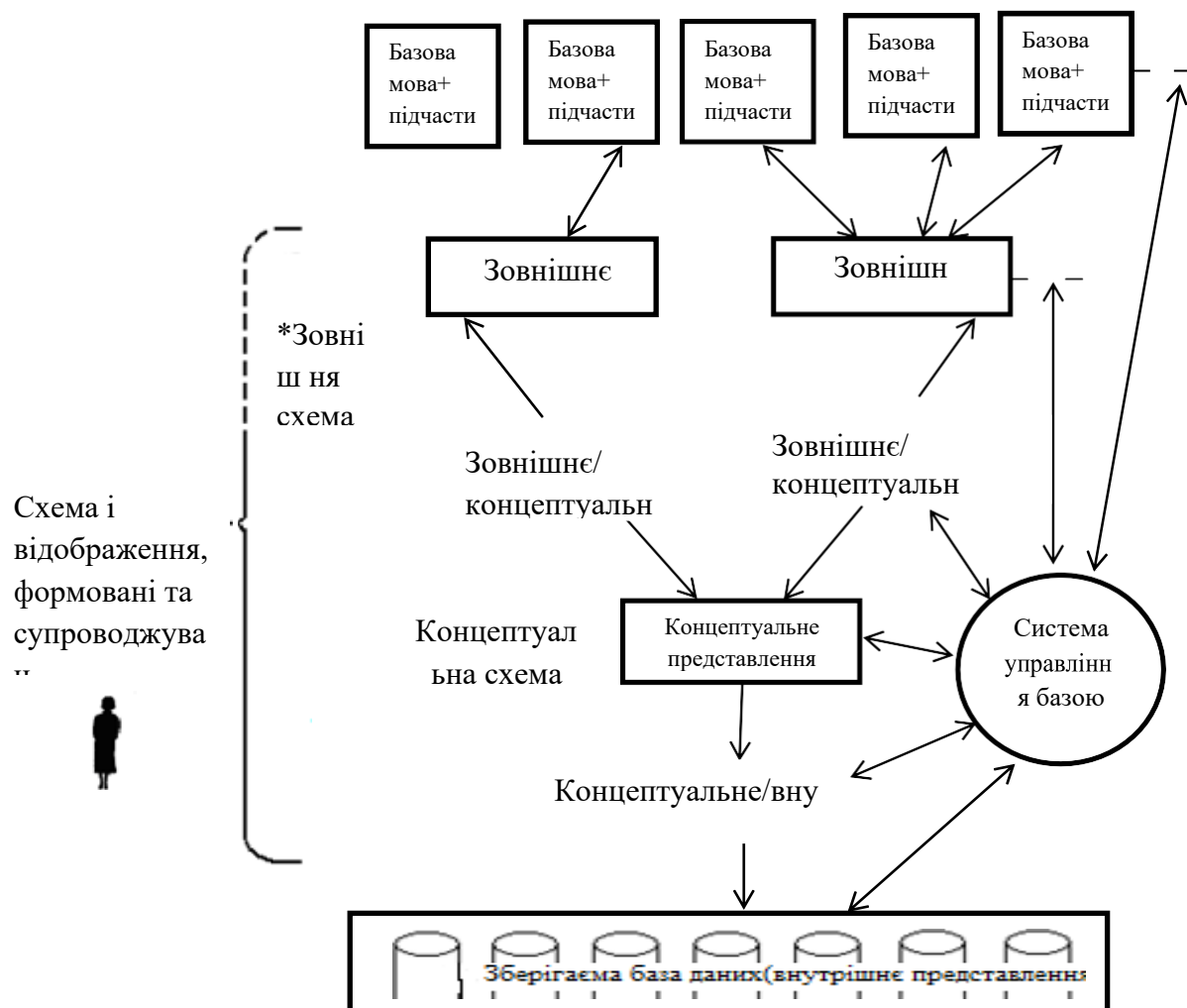
Фізична незалежність від даних означає захищеність концептуальної схеми від змін, що вносяться у внутрішню схему.

Такі зміни внутрішньої схеми, як використання різних файлових систем або структур зберігання, різних пристроїв зберігання, модифікація *індексів* або хешування, повинні здійснюватися без необхідності внесення змін до концептуальної або зовнішньої схеми.

Користувачем можуть бути помічені зміни тільки в загальній продуктивності системи.

Насправді найбільш поширеною причиною внесення змін у внутрішню схему є саме недостатня продуктивність виконання операцій. На рис. 6.2 показано місце перерахованих вище типів незалежності від даних в трирівневій архітектурі СУБД.

Прийняте в архітектурі ANSI-SPARC двоетапне відображення може позначатися на ефективності роботи, але при цьому воно забезпечує більш високу незалежність від даних. Для підвищення ефективності в моделі ANSI-SPARC допускається використання прямого відображення зовнішніх схем на внутрішні, без звернення до концептуальної схемою. Однак це знижує рівень незалежності від даних, оскільки при кожній зміні внутрішньої схеми потрібно внесення певних змін у зовнішню схему і всі залежні від неї прикладні програми.



*Користувачський інтерфейс

Рис. 6.2. Реалізація незалежності від даних в тривірневій архітектурі

ANSI-SPARC.

Відображення

Архітектура системи баз даних, крім елементів самих трьох рівнів, включає певні відображення: *відображення* концептуального рівня на внутрішній і кілька відображень зовнішніх рівнів на концептуальний.

Відображення
«концептуальний-внутрішній»
встановлює відповідність між концептуальним поданням і збереженою базою даних, тобто описує, як концептуальні записи і поля представлені на внутрішньому рівні.

При зміні структури збереженої бази даних відображення «концептуальний-внутрішній» також змінюється, з урахуванням того, що концептуальна схема залишається незмінною

Відображення «зовнішній-концептуальний» визначає відповідність між деякими зовнішнім представленням і концептуальним представленням.

Відображення «концептуальний-внутрішній» служить основою *фізичної незалежності від даних*, а відображення «зовнішній- концептуальний» є *ключем до логічної незалежності від даних*.

Слід зазначити, що більшість систем дозволяє виражати одне визначення зовнішнього представлення через інше, не вимагаючи обов'язкового явного визначення відображення кожного зовнішнього представлення на концептуальний рівень. Ця можливість дуже корисна, якщо кілька представлень подібні один одному. Зокрема, аналогічна можливість передбачена в багатьох реляційних СУБД.

Схеми

*Загальний опис бази даних називається **схемою бази даних**.*

Існує *три різних типи схем бази даних*, які визначаються згідно з рівнями абстракції трирівневої архітектури, як показано на рис. 6.1.

На *найвищому рівні* є декілька **зовнішніх схем** або *підсхем*, які відповідають різним представленням даних.

На *концептуальному рівні* опис бази даних називають *концептуальною схемою*, а на *найнижчому рівні абстракції* - *внутрішньою схемою*.

Концептуальна схема описує всі елементи даних і зв'язки між ними, із зазначенням необхідних обмежень підтримки цілісності даних.

Для кожної бази даних є тільки одна концептуальна схема. На нижньому рівні знаходиться **внутрішня схема**, яка є повним описом внутрішньої моделі даних, містить визначення збережених записів, методи представлення, описи полів даних, відомості про індекси і вибрані схеми хешування. Для кожної бази даних існує тільки одна внутрішня схема.

СУБД відповідає за встановлення відповідності між цими трьома типами схем, а також за перевірку їх несуперечності.

СУБД повинна переконатися в тому, що кожну зовнішню схему можна вивести на основі концептуальної схеми.

Для встановлення відповідності між будь-якими зовнішньою і внутрішньою схемами СУБД повинна використовувати інформацію з концептуальної схеми.

Приклади різних рівнів наведено на рис. 6.3. На ньому показані два різних зовнішніх представлення інформації про студента: одне складається з ПІБ студента (FIO), віку (Age), адреси студента (Address), середнього балу (Sr.Ball). Інше представлення включає номер групи (Group_No), номер Студента в групі (Student_No), ПІБ (FIO) студента. Ці зовнішні представлення зливаються воедино в одному концептуальному

представленні. Особливістю даного процесу злиття є те, що поле віку студента (Age) перетворюється, в поле дати його народження (DOB). Потім концептуальний рівень відображається на внутрішній рівень, який містить фізичний опис структури запису концептуального представлення. На цьому рівні визначення структури формулюється мовою високого рівня. Ця структура містить покажчик (Next), який дозволяє фізично зв'язати всі записи про студентів в єдиний ланцюжок. Зверніть увагу, що порядок полів на внутрішньому рівні відрізняється від порядку атрибутів, прийнятого на концептуальному рівні. Такий механізм, за допомогою якого СУБД здійснює концептуально внутрішнє відображення.

Важливо розрізняти опис бази даних і саму базу даних. **Описом бази даних** є *схема бази даних, яка створюється в процесі її проектування, причому передбачається, що вона змінюється досить рідко.*

Інформація, що міститься в базі даних міняється часто.

Наприклад, щоразу при вставці відомостей про нового співробітника або новий об'єкт нерухомості, що здається в оренду.

*Сукупність інформації, що зберігається в базі даних у будь-який визначений момент часу, називається **станом бази даних**.*

Одній і тій же схемі бази даних може відповідати безліч її різних станів.

Схема бази даних іноді називається *вмістом бази даних*, а її стан - *деталізацією*.

6.2 Система управління Базами Даних

Система управління базою даних (СУБД) - це програмне забезпечення, за допомогою якого користувачі можуть визначати, створювати і підтримувати базу даних, а також здійснювати до неї контрольований доступ. Концептуально це відбувається таким чином (див. рис. 6.2).

1. Користувач видає запит на доступ до даних, застосовуючи певну внутрішню мову даних (зазвичай це мова SQL).

2. СУБД перехоплює цей запит і аналізує його.

3. СУБД переглядає зовнішню схему (її об'єктну версію) для цього користувача, відповідне відображення «зовнішній-концептуальний», концептуальну схему, відображення «концептуальний-внутрішній» і визначення структур зберігання.

4. СУБД виконує необхідні операції в збереженій базі даних. Рис. 6.4. Компоненти системи бази даних.

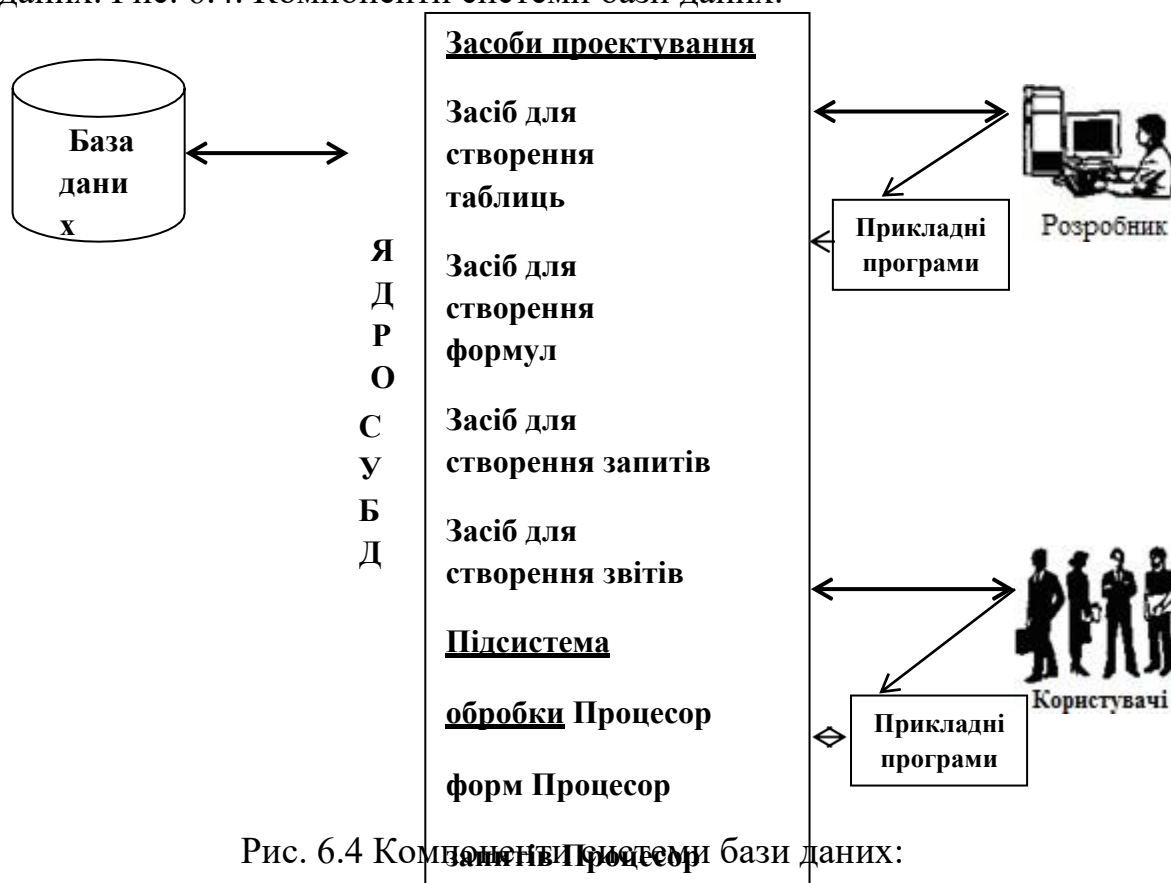


Рис. 6.4 Компоненти системи бази даних:

Компоненти системи бази даних:

1. **Підсистема засобів проектування.** Підсистема засобів проектування являє собою набір інструментів, що спрощують проектування і реалізацію баз даних та їх додатків. Як правило, цей набір включає в себе

засоби для створення таблиць, форм, запитів і звітів. У СУБД є також мови програмування та інтерфейси для них.

2. Підсистема обробки. Підсистема обробки займається обробкою компонентів програми, створених за допомогою засобів проектування. Наприклад, у Access 2002 існує компонент, що матеріалізує форми і зв'язуючі елементи форм з даними таблицями. Уявіть собі форму з текстовим полем, де відображається значення стовпця Номер Студента з таблиці СТУДЕНТ. У процесі роботи програми при відкритті форми процесор форм витягує значення поля Номер Студента з поточного рядка таблиці і відображає його у формі. Все це робиться автоматично - ні користувачеві, ні розробнику не потрібно нічого робити, якщо є готова форма.

Інші процесори підсистеми обробки призначені для виконання запитів і виведення звітів. Крім того, в підсистемі обробки є компонент, що обробляє запити прикладних програм на читання і запис даних в базу.

3. Ядро СУБД. Третій компонент СУБД - це її ядро, яке виконує функцію посередника між підсистемою засобів проектування та обробки даних. Ядро СУБД отримує запити від двох інших компонентів, виражених в термінах таблиць, рядків і стовпців, і перетворює ці запити в команди операційної системи, що виконують запис і читання даних з фізичного носія. Крім того, ядро СУБД бере участь в управлінні транзакціями, блокуванні, резервного копіювання та відновлення. Ядро СУБД допомагає координувати дії з тим, щоб або виконувалися всі дії в групі, або не виконалось жодного.

СУБД - це програмне забезпечення, яке взаємодіє з прикладними програмами користувача і базою даних та володіє наведеними нижче можливостями:

1. Дозволяє визначати базу даних, що зазвичай здійснюється за допомогою мови визначення даних (*DDL - Data Definition Language*). DDL призначений для надання користувачам засобів вказівки типу даних та їх структури, а також засобів задання обмежень для інформації, що зберігається в базі даних.

2. Дозволяє вставляти, оновлювати, видаляти і витягувати інформацію з бази даних, що зазвичай здійснюється за допомогою мови керування даними (*Data Manipulation Language*). Наявність централізованого сховища всіх даних та їх описів дозволяє використовувати мову DML як загальний інструмент організації запитів, який іноді називають мовою запитів (*query language*). Мова запитів дозволяє усунути властиві файловим системам обмеження, при яких користувачам доводиться мати справу тільки з фіксованим набором запитів або постійно зростаючою кількістю програм, що породжує інші, більш складні проблеми управління програмним забезпеченням.

Існує два різновиди мов DML - *процедурна (procedural)* і *непроцедурна (non-procedural)* мови, - які відрізняються між собою способом вилучення даних. **Процедурні мови** звичайно обробляють інформацію в базі даних послідовно, запис за записом, а **непроцедурні** оперують відразу цілими наборами записів. З допомогою процедурних мов DML зазвичай вказується, як можна отримати бажаний результат, тоді як непроцедурні мови DML використовуються для опису того, що слід отримати.

Найбільш поширеним типом *процедурної мови* є **мова структурованих запитів (Structured Query Language-SQL)**, яка в даний час визначається спеціальним стандартом і фактично є обов'язковою мовою для будь-яких реляційних СУБД.

3. **СУБД** надає контрольований доступ до бази даних за допомогою перерахованих нижче засобів:

- *системи забезпечення безпеки*, що запобігає несанкціонований доступ до бази даних з боку користувачів;
- *системи підтримки цілісності даних*, що забезпечує несуперечливий стан збережених даних;
- *системи управління паралельною роботою прикладних програм*, яка контролює процеси їх спільного доступу до бази даних;
- *системи відновлення*, що дозволяє відновити базу даних до попереднього несуперечливого стану, порушеного в результаті збою апаратного або програмного забезпечення;
- *доступного користувачам каталогу*, що містить опис інформації, що зберігається в базі даних.

Компоненти середовища СУБД

У середовищі СУБД можна виділити наступних п'ять основних компонентів:

- апаратне забезпечення,
- програмне забезпечення,
- дані,
- процедури,
- користувачі.

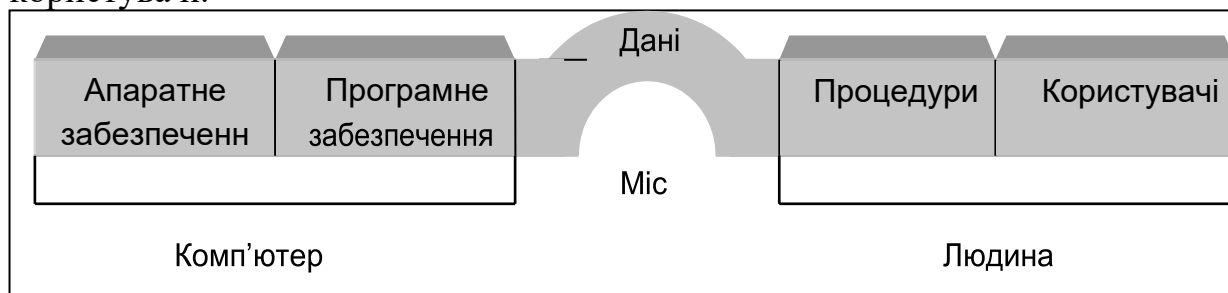


Рис. 6.5. Середовище СУБД.

Апаратне забезпечення

До апаратного забезпечення системи відноситься наступне:

- томи вторинної (зовнішньої) пам'яті (зазвичай це магнітні диски), що використовуються для зберігання інформації, а також відповідні пристрої введення-виведення і т. д.;
- апаратний процесор (або процесори) разом з оперативною (первинною) пам'яттю, призначений для підтримки роботи програмного забезпечення системи баз даних.

Для роботи СУБД і прикладних програм необхідне деяке апаратне забезпечення. Воно може варіюватись в дуже широких межах - від єдиного персонального комп'ютера або одного мейнфрейма до мережі з багатьох комп'ютерів.

На *центральному комп'ютері працює серверна частина СУБД* (backend), яка обслуговує і контролює доступ до бази даних.

На *інших комп'ютерах працюють клієнтські частини СУБД* (frontend), які здійснюють взаємодію з користувачами.

Подібна архітектура носить назву клієнт / сервер (client-server), де сервером є комп'ютер з серверною частиною СУБД, а клієнтами - комп'ютери з клієнтськими частинами СУБД.

Програмне забезпечення

Цей компонент охоплює *програмне забезпечення самої СУБД і прикладних програм, разом з операційною системою, включаючи і мережеве програмне забезпечення, якщо СУБД використовується в мережі*.

СУБД може мати свої власні інструменти четвертого покоління, призначені для швидкої розробки додатків з використанням вбудованих непроцедурних мов запитів, генераторів звітів, форм, графічних зображень і навіть повномасштабних додатків.

Використання інструментів четвертого покоління може істотно підвищити продуктивність системи і сприяти створенню більш зручних, для обслуговування програм.

Дані

Найважливішим компонентом середовища СУБД (з точки зору кінцевих користувачів) є дані.

База даних містить як робочі дані, так і метадані, тобто "Дані про дані". Структура бази даних називається **схемою** (schema).

У загальному випадку дані в базі даних (принаймні, у великих системах) є *інтегрованими і розділюваними*. Ці два аспекти, *інтеграція і розділювання* даних, являють собою найбільш важливіше переваги використання систем баз даних на «великому» обладнанні і, щонайменше, один з них - інтеграція - є перевагою їх застосування і на «малому» обладнанні.

Під поняттям **інтеграції даних** мається на увазі можливість представити базу даних як об'єднання декількох окремих файлів даних, повністю або частково виключає надмірність зберігання інформації.

Під поняттям **розділюваність даних** мається на увазі можливість використання кількома різними користувачами окремих елементів, що зберігаються в базі даних, кожен з користувачів зможе отримати доступ до одних і тих же даних, можливо, навіть одночасно (*паралельний доступ*). Такий поділ даних, з паралельним або послідовним доступом, частково є наслідком того факту, що база даних має інтегровану структуру. (Якщо база даних не піддається поділу, то її іноді називають *особистою базою* або *базою даних спеціального призначення*).

У системному каталозі містяться такі відомості:

- імена, типи і розміри елементів даних;
- імена зв'язків;
- обмеження цілісності даних;
- імена зареєстрованих користувачів, яким надані деякі права доступу до даних;
- індекси та структури зберігання, що використовуються - наприклад, інвертовані файли або дерева B +.

Процедури

До процедур відносяться інструкції та правила, які повинні враховуватися при проектуванні та використанні бази даних.

Користувачам і обслуговуючому персоналу бази даних надається документація, яка містить докладний опис процедур використання та супроводу даної системи, включаючи інструкції про правила виконання наведених нижче дій.

- Реєстрація в СУБД.
- Використання окремого інструменту СУБД або програми.
- Запуск і зупинка СУБД.
- Створення резервних копій СУБД.
- Обробка збоїв апаратного та програмного забезпечення, включаючи процедури ідентифікації компонента, який вийшов з ладу, виправлення компонента, що вийшов з ладу (наприклад, за

допомогою виклику фахівця з ремонту апаратного забезпечення), а також відновлення бази даних після усунення несправності.

- Зміна структури таблиці, реорганізація бази даних, розміщеної на кількох дисках, способи поліпшення продуктивності і методи архівації даних на вторинних пристроях зберігання.

Користувачі

Серед користувачів можна виділити *чотири різні групи*:

- адміністратори даних і баз даних,
- розробники баз даних,
- прикладні програмісти
- кінцеві користувачі.

Адміністратори даних і адміністратори баз даних

База даних і СУБД є корпоративними ресурсами, якими слід керувати так само, як і будь-якими іншими ресурсами.

Зазвичай управління даними і базою даних передбачає управління і контроль над СУБД і поміщеними в неї даними.

Адміністратор даних (DA - data administration) відповідає за управління даними, включаючи планування бази даних, розробку і супровід стандартів, бізнес-правил і ділових процедур, а також за концептуальне і логічне проектування бази даних.

Адміністратор бази даних, або АБД (Database Administrator - DBA), відповідає за фізичну реалізацію бази даних, включаючи *фізичне проектування та втілення проекту, за забезпечення безпеки і цілісності даних, за супровід операційної системи, а також за забезпечення максимальної продуктивності додатків і користувачів.*

Розробники баз даних

У проектуванні великих баз даних беруть участь *два різних типи розробників: розробники логічної бази даних і розробники фізичної бази даних.*

Розробник логічної бази даних займається ідентифікацією даних (тобто сутностей та їх атрибутів), зв'язків між даними і встановлює обмеження, що накладаються на збережені дані, повинен всебічно і в повному обсязі розуміти структури даних організації та її бізнес-правил.

Бізнес-правила описують основні характеристики даних з точки зору організації або описуваної предметної області.

Нижче наводяться приклади типових бізнес-правил.

- Будь-який викладач не може проводити, більш ніж однозначно встановлену кількість лекцій з даного предмета.
- Будь-який студент не має права без поважної причини пропускати лекції.
- Викладач не може бути одночасно завідувачем декількох кафедр в одному університеті.

Для ефективної роботи розробник логічної бази даних повинен якомога раніше залучити всіх можливих користувачів бази даних в процес створення моделі даних.

Робота розробника логічної бази даних ділиться на два етапи:

1. Концептуальне проектування бази даних, яке абсолютно не залежить від таких деталей її втілення, як конкретна цільова СУБД, додатки, мови програмування або будь-які інші фізичні характеристики.

2. Логічне проектування бази даних, яке проводиться з урахуванням особливостей обраної моделі даних: реляційної, мережевої, ієрархічної або об'єктно-орієнтованої.

Розробник *фізичної бази даних* отримує готову логічну модель даних, займається її *фізичною реалізацією*, в тому числі:

- перетворенням логічної моделі даних в набір таблиць і обмежень цілісності даних;
- вибором конкретних структур зберігання і методів доступу до даних, що забезпечують необхідний рівень продуктивності при роботі з базою даних;
- проектуванням будь-яких необхідних заходів захисту даних.

Багато етапів фізичного проектування бази даних в значній мірі залежать від обраної цільової СУБД, а тому може існувати кілька різних способів втілення необхідної схеми. Отже, розробник фізичної бази даних повинен розбиратися у функціональних можливостях цільової СУБД і розуміти переваги і недоліки кожного можливого варіанту втілення. Розробник фізичної бази даних повинен вміти вибрати найбільш підходящу стратегію зберігання даних з урахуванням усіх існуючих особливостей їх використання. Якщо концептуальне і логічне проектування бази даних відповідає на питання "що?", То фізичне проектування відповідає на питання "як?".

Прикладні програмісти

Після створення бази даних слідує розробка додатків, що надають користувачам необхідні їм функціональні можливості, яку і виконують *прикладні програмісти*.

Вони працюють на основі *специфікацій*, створених системними аналітиками.

Користувачі

Користувачі є клієнтами бази даних - вона проектується, створюється і підтримується для того, щоб обслуговувати їх інформаційні потреби.

Користувач може отримувати доступ до бази даних, застосовуючи одну з *інтерактивних програм* або ж *інтерфейс* (вбудований додаток), інтегрований в програмне забезпечення самої СУБД. Більшість СУБД включає, принаймні, один вбудований додаток, а саме - **процесор мови запитів**, що дозволяє користувачеві в діалоговому режимі вводити запити до бази даних. Мова SQL являє собою типовий приклад мови запитів до бази даних.

Крім мови запитів, в більшості систем додатково надаються *спеціалізовані вбудовані інтерфейси*, в яких користувач в явному вигляді не використовує пропозиції, або команди з такими операторами, як SELECT та INSERT. Робота з базою даних здійснюється за рахунок вибору користувачем

необхідних елементів меню або заповнення необхідних полів у поданих формах. Такі **некомандні інтерфейси**, полегшують роботу з базами даних для тих, хто не має досвіду роботи з інформаційними технологіями. **Командний інтерфейс**, тобто мова запитів, навпаки, вимагає деякого професійного досвіду роботи з інформаційними технологіями. Він більш гнучкий, ніж некомандний, до того ж мови запитів зазвичай включають певні функції, відсутні в інтерфейсах, заснованих на використанні меню або форм.

Таким чином, спрощена схема взаємодії всіх компонент середовища СУБД представлена на рис. 6.6. Процес взаємодії реалізується за допомогою програмного забезпечення, розміщеного на апаратній платформі, суть взаємодії полягає в доступі до різних наборів даних різними користувачами

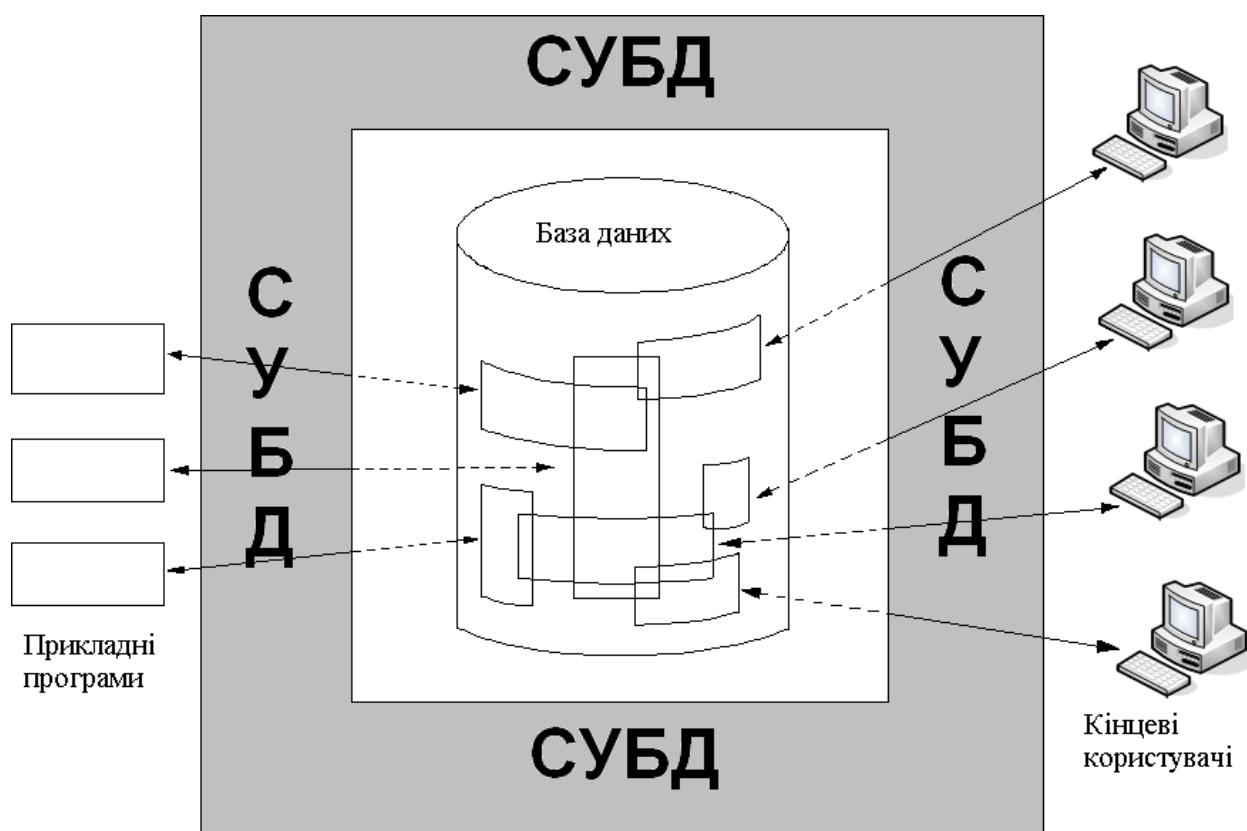


Рис. 6.6. Спрощена схема системи баз даних

Функції СУБД

У цьому розділі розглянемо *типи функцій і служб (сервісів)*, які повинна забезпечувати типова СУБД. У свій час Кодд запропонував перелік з *восьми сервісів*, які повинні бути реалізовані в будь-якій повномасштабній СУБД (Codd 1982).

1.Зберігання, вилучення та оновлення даних

СУБД повинна надавати користувачам можливість зберігати, витягати і оновлювати дані в базі даних.

2. Каталог доступний кінцевим користувачам

СУБД повинна мати доступний кінцевим користувачам каталог, в якому зберігається опис елементів даних.

Ключовою особливістю архітектури ANSI-SPARC є наявність інтегрованого системного каталогу з даними про схеми користувачів, додатки тощо. передбачається, що каталог доступний як користувачам, так і функціям СУБД. Системний каталог, або словник даних, є сховищем інформації, яка описує дані в базі даних (по суті, це "дані про дані", або метадані).

Залежно від типу СУБД, використовується кількість інформації і способи її застосування можуть варіюватися.

У системному каталозі зберігаються такі відомості:

- імена, типи і розміри елементів даних;
- імена зв'язків;
- накладаються на дані обмеження підтримки цілісності;
- імена санкціонованих користувачів, яким надано право доступу до даних;
- зовнішня, концептуальна і внутрішня схеми, відображення між ними;
- статистичні дані, наприклад частота транзакцій і лічильники звернень до об'єктів бази даних.

Системний каталог дозволяє досягти певних переваг, перерахованих нижче:

- Інформація про дані може бути централізовано зібрана і збережена, що дозволить контролювати доступ до цих даних, як і до будь-якого іншого ресурсу.
- Можна визначити сенс даних, що допоможе іншим користувачам зрозуміти їх призначення.
- Спрощується повідомлення, так як зберігаються точні визначення сенсу даних. У системному каталозі також можуть бути вказані один або кілька користувачів, які є власниками даних або володіють правом доступу до них.
- Завдяки централізованому зберіганню надмірність і суперечливість опису окремих елементів даних можуть бути легко виявлені.
- Внесені в базу даних зміни можуть бути за протокольовані.
- Наслідки будь-яких змін можуть бути визначені ще до їх внесення, оскільки в системному каталозі зафіксовані всі існуючі елементи даних, встановлені між ними зв'язки, а також усі їхні користувачі.
- Заходи забезпечення безпеки можуть бути додатково посилені.
- З'являються нові можливості організації підтримки цілісності даних.
- Може виконуватися аудит інформації, що зберігається.

Деякі автори, крім системного каталогу, виділяють каталог даних (data directory), в якому знаходиться інформація про місце та спосіб зберігання даних.

3. Підтримка транзакцій

Транзакція являє собою набір дій, що виконуються окремим користувачем або прикладною програмою, з метою доступу або зміни вмісту бази даних.

СУБД повинна мати механізм, який гарантує виконання або всіх операцій оновлення даної транзакції, або жодної з них.

Прикладом простих транзакцій у проекті «Навчальний процес» може служити додавання в базу даних відомостей про нового викладача, оновлення відомостей про його зарплату або надання відомостей про деякий предмет, який він викладав. Прикладом більш складною транзакції може бути видалення відомостей про викладача з бази даних і передача відповідальності за всі групи, у яких він був викладачем іншому співробітникові. У цьому випадку в базу даних буде потрібно внести відразу декілька змін.

4. Сервіси управління паралельністю

СУБД має механізм, який гарантує коректне оновлення бази даних при

паралельному виконанні операцій оновлення багатьма користувачами.

Одна з основних цілей створення і використання СУБД полягає в тому, щоб безліч користувачів могло здійснювати паралельний доступ до спільно-оброблюваних даних.

Паралельний доступ порівняно просто організувати, якщо всі користувачі виконують тільки читання даних, оскільки в цьому випадку вони не можуть перешкодити один одному. Однак коли два або більше користувачів одночасно отримують доступ до бази даних, конфлікт з небажаними наслідками легко може виникнути, наприклад, якщо хоча б один з них спробує оновити дані.

Розглянемо паралельно виконувані транзакції T_1 і T_2 , послідовність виконання яких представлена в табл. 6.1.

Таблиця 6.1. Приклад виникнення проблеми "втраченого оновлення"

Час	Транзакція T_1	Транзакція T_2	Стовпчик bal_x
t_1		Зчитування (bal_x)	100
t_2	Зчитування (bal_x)	$bal_x \square bal_x \square 10$	100
t_3	$bal_x \square bal_x \square 10$	Запис (bal_x)	200
t_4	Запис (bal_x)		90
t_5			90

T1 10 грн. знімається з рахунку, баланс якого зберігається в стовпці bal_x , а в транзакції T2 100 грн. поміщається на цей же рахунок. Якби транзакції виконувалися послідовно, тобто одна за одною, без чергування окремих операцій, то в кінцевому підсумку баланс був би рівний 190 грн., незалежно від порядку виконання транзакцій. Інакше може виникнути наступна ситуація. Припустимо, що транзакції T1 і T2 стартували практично одночасно і прочитали вихідне значення балансу, рівне 100 грн. Потім транзакція T2 збільшує це значення на 100 грн. (До 200 грн.) І зберігає отримане значення в базі даних. Негайно після цього транзакція T1 зменшує свою копію ліченого вихідного значення на 10 грн. (До 90 грн.) І зберігає отримане значення в базі даних замість щойно записаного результату попереднього оновлення, внаслідок чого виникає ефект "втрати" 100 грн.

СУБД повинна гарантувати, що при одночасному доступі до бази даних багатьох користувачів подібних конфліктів не відбудеться.

5. Сервіси відновлення

СУБД надає засоби відновлення бази даних на випадок будь-якого її пошкодження або руйнування.

При обговоренні підтримки транзакцій згадувалося, що *при збої транзакції база даних повинна бути повернута в несуперечливий стан*. Подібний збій може відбутися в результаті виходу з ладу системи або запам'ятовуючого пристрою, помилки апаратного або програмного забезпечення, які можуть призвести до не функціонування СУБД. Крім того, користувач може виявити помилку під час виконання транзакції і зажадати її скасування. У всіх цих випадках СУБД повинна надати механізм відновлення бази даних і повернення її до несуперечливого стану.

6. Сервіси контролю доступу до даних

СУБД має механізм, що гарантує можливість доступу до бази даних тільки *санкціонованих користувачів*.

Викладачам можна було б надати всю інформацію, пов'язану з екзаменаційними білетами, але бажано приховати її від студентів. Крім того, базу даних слід було б захистити від будь-якого несанкціонованого доступу. Термін безпеки відноситься до захисту бази даних від навмисного або випадкового несанкціонованого доступу. Передбачається, що СУБД забезпечує механізми такого захисту даних.

7. Підтримка обміну даними

СУБД має здатність до інтеграції з комунікаційним програмним забезпеченням.

Сучасна СУБД працює, як правило, в мережевому режимі, *отримуючи запити у вигляді повідомлень обміну даними (communications messages) і аналогічним чином відповідаючи на них*. Такі спроби передачі даних управляються менеджером обміну даними. Хоча цей менеджер не є частиною власне СУБД, тим не менш, щоб бути комерційно життєздатною,

будь-яка

СУБД повинна мати здатність інтеграції з різноманітними існуючими менеджерами обміну

даними.

Замість кількох розрізнених баз даних для кожного окремого користувача при роботі в мережі можна встановити одну централізовану базу даних і використовувати її як загальний ресурс для всіх існуючих користувачів. При цьому передбачається, що не база даних повинна бути розподілена в мережі, а віддалені користувачі повинні мати можливість доступу до централізованої бази даних. Така топологія називається *розподіленою обробкою*.

Розподілена обробка досягається за рахунок *реалізації механізму підтримки уявлень чи підсхем*.

Приклад розподіленої обробки представлений на рис. 6.7. Сервер зберігає свою певну інформацію, а користувачу видна інформація, що міститься в Єдиній центральній БД. Він не вникає в структуру зберігання.

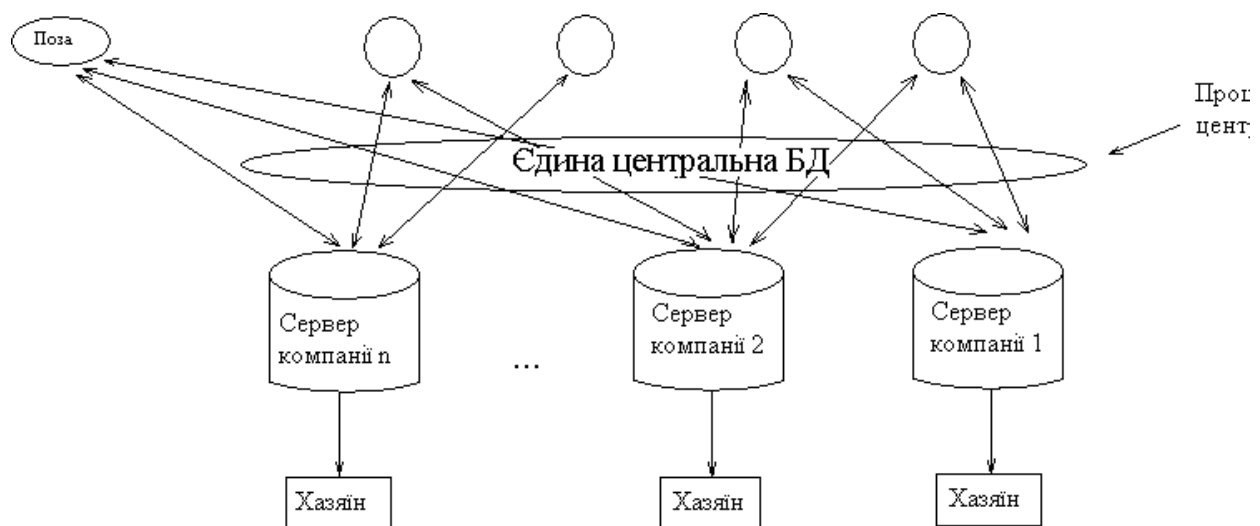


Рис. 6.7. Приклад розподіленої обробки.

Фізична незалежність від даних досягається досить просто, так як зазвичай є кілька типів допустимих змін фізичних характеристик бази даних, які ніяк не впливають на представлення.

Важче домогтися повної логічної незалежності від даних.

Як правило, система легко адаптується до додавання нового об'єкта, атрибута або зв'язку, але не до їх видалення. У деяких системах взагалі забороняється вносити будь-які зміни у вже існуючі компоненти логічної схеми.

8. Допоміжні служби

СУБД надає певний набір різних допоміжних служб.

Допоміжні утиліти зазвичай призначені для надання допомоги АБД в ефективному адмініструванні бази даних.

Одні утиліти працюють на зовнішньому рівні, а тому вони, в принципі, можуть бути створені самим АБД, тоді як інші функціонують на

внутрішньому рівні системи і тому повинні бути надані самим розробником СУБД.

Мінімальний перелік утиліт:

- *Утиліти імпортування*, призначені для завантаження бази даних з плоских файлів, а також *утиліти експортування*, які служать для вивантаження бази даних в плоскі файли.
- *Засоби моніторингу*, призначені для відстеження характеристик функціонування та використання бази даних.
- *Програми статистичного аналізу*, що дозволяють оцінити продуктивність або ступінь використання бази даних.
- *Інструменти реорганізації індексів*, призначені для перебудови індексів і обробки випадків їх переповнення.
- *Інструменти збірки сміття та перерозподілу пам'яті* для фізичного усунення видалених записів з запам'ятовуючих пристроїв, об'єднання звільненого простору і перерозподілу пам'яті у разі потреби.

Мови баз даних

Внутрішній мова СУБД для роботи з даними складається з двох частин: *мови визначення даних (Data Definition Language-DDL)* і *мови управління даними (Data Manipulation Language-DML)*. Мова DDL використовується для визначення схеми бази даних, а мова DML - для читання і поновлення даних, що зберігаються в базі.

Мови DDL і DML називаються **підмовою даних**, оскільки в них відсутні конструкції для виконання всіх обчислювальних операцій, зазвичай використовуються в мовах програмування високого рівня.

У багатьох СУБД передбачено механізм впровадження операторів підмови даних у програми, написані на таких мовах програмування високого рівня, як COBOL, Fortran, Pascal, Ada або C.

У цьому випадку мову високого рівня прийнято називати базовою мовою (host language). Перед компіляцією файлу програми на базовій мові, яка містить впроваджені оператори підмови даних, буде потрібно попередньо видалити ці оператори, замінивши їх викликами відповідних функцій СУБД. Потім цей попередньо оброблений файл звичайним чином компілюється з внесенням результатів в об'єктний модуль, який компонується з бібліотекою, яка містить функції, що викликаються в програмі СУБД. Після цього отриманий програмний текст буде готовий до виконання. Крім механізму впровадження, для більшості підмов даних також надаються засоби інтерактивного виконання їх

операторів, що вводяться користувачем безпосередньо зі свого терміналу.

Мова визначення даних DDL – мова опису, яка дозволяє АБД або користувачеві описати і назвати сутності, необхідні для роботи деякої прикладної програми, а також зв'язків, наявних між різними сутностями

Схема бази даних складається з набору визначень, виражених на спеціальній мові визначення даних - DDL. Мова DDL використовується як для визначення нової схеми, так і для модифікації вже існуючої. Цю мову не можна використовувати для управління даними. Результатом компіляції DDL-операторів є набір таблиць, що зберігається в особливих файлах, названі системним каталогом. У системному каталозі інтегровані метадані - тобто дані, які описують об'єкти бази даних, а також дозволяють спростити спосіб доступу до них і управління ними. Метадані включають визначення записів, елементів даних, а також інші об'єкти, що представляють інтерес для користувачів або необхідні для роботи СУБД. Перед доступом до реальних даних СУБД зазвичай звертається до системного каталогу. Для позначення системного каталогу також використовуються терміни словник даних і каталог даних, хоча перший з них (словник даних) зазвичай відноситься до програмного забезпечення більш загального типу, ніж просто каталог СУБД.

Теоретично для кожної схеми в трирівневій архітектурі можна було б виділити кілька різних мов DDL, а саме мова DDL зовнішніх схем, мова DDL концептуальної схеми і мова DDL внутрішньої схеми. Однак на практиці існує одна спільна мова DDL, яка дозволяє задавати специфікації, як мінімум, для зовнішньої та концептуальної схем.

Мова керування даними (DML) - мова, що містить набір операторів для підтримки основних операцій маніпулювання, що містяться в базі даними.

До операцій маніпулювання (управління) даними відносяться:

- вставка в базу даних нових відомостей;
- модифікація відомостей, що зберігаються в базі даних;
- витяг відомостей, що містяться в базі даних;
- видалення відомостей з бази даних.

Поняття маніпулювання даними застосовується як до зовнішнього і концептуального а також до внутрішнього рівнів.

Однак на внутрішньому рівні для цього необхідно визначити дуже складні процедури низького рівня, що дозволяють виконувати доступ до даних дуже ефективно. На більш високих рівнях, навпаки, акцент переноситься в бік більшої простоти використання і основні зусилля направляються на забезпечення ефективної взаємодії користувача системи. Мови DML відрізняються базовими конструкціями вилучення даних.

Розрізняють два типи мов DML: *процедурна і непроцедурна*. Основна відмінність між ними полягає в тому, що процедурні мови вказують те, як можна отримати результат оператора мови DML, тоді як непроцедурні мови описують те, який результат буде отриманий. Як правило, в процедурних

мовах записи розглядаються окремо, тоді як непроцедурні мови оперують з цілими наборами записів.

Процедурні мови DML - мови, які дозволяють повідомити системі про те, які дані необхідні, і точно вказати, як їх можна витягти.

За допомогою процедурної мови DML користувач, а точніше - програміст, вказує на те, які дані йому необхідні і як їх можна отримати. Це означає, що користувач повинен визначити всі операції доступу до даних (здійснювані за допомогою виклику відповідних процедур), які повинні бути виконані для отримання необхідної інформації. Зазвичай така процедурна мова DML дозволяє витягти запис, обробити його і, залежно від отриманих результатів, витягти інший запис, який повинний бути підданий аналогічній обробці, тощо. Подібний процес вилучення даних продовжується до тих пір, поки не будуть отримані всі дані запиту. Мови DML мережних і ієрархічних СУБД зазвичай є процедурним

Непроцедурні мови DML - мови, які дозволяє вказати лише те, які дані потрібні, але, не те, як їх слід витягати.

Непроцедурні мови DML дозволяють визначити весь набір необхідних даних за допомогою одного оператора вилучення або оновлення. За допомогою непроцедурних мов DML користувач вказує, які дані йому потрібні, без визначення способу їх отримання. СУБД транслює вираз мовою DML в процедуру, (або набір процедур), яка забезпечує маніпулювання потрібним набором записів. Даний підхід звільняє користувача від необхідності знати деталі внутрішньої реалізації структур даних і особливості алгоритмів, що використовуються для отримання і можливого перетворення даних. У результаті робота користувача отримує певну ступінь незалежності від даних.

Непроцедурні мови часто також називають декларативними мовами. Реляційні СУБД в тій чи іншій формі зазвичай включають підтримку непроцедурних мов маніпулювання даними - найчастіше це буває мова *структурованих запитів SQL (structured Query Language)* або мова *запитів за зразком QBE (Query-by-Example)*. [посилання]

Частина непроцедурної мови DML, яка відповідає за вилучення даних, називається мовою запитів. Мова запитів можна визначити як високорівневу вузькоспеціалізовану мову, призначену для задоволення різних вимог по вибірці інформації з бази даних.

Мови 4GL. Аббревіатура "4GL" представляє собою скорочений англійський варіант написання терміна мову четвертого покоління (Fourth-Generation-Language). Не існує чіткого визначення цього поняття, хоча, по суті, мова йде про деякий стенографічний варіант мови програмування. Якщо для організації деякої операції з даними на мові третього покоління (3GL) типу COBOL буде потрібно написати сотні рядків коду, то для реалізації цієї ж операції на мові четвертого покоління буде достатньо 10-20 рядків.

У той час як мови третього покоління є процедурними, мови 4GL виступають як непроцедурні, оскільки користувач визначає, що повинно бути

зроблено, але не говорить, як саме бажаний результат повинен бути досягнутий. Передбачається, що реалізація мов четвертого покоління буде значною мірою заснована на використанні компонентів високого рівня, які часто називають "інструментами четвертого покоління". Користувачеві не потрібно визначати всі етапи виконання програми, необхідні для вирішення поставленого завдання, а достатньо буде лише, визначити потрібні параметри, на підставі яких згадані вище інструменти автоматично здійснять генерацію прикладного програмного забезпечення. Очікується, що мови четвертого покоління дозволять підвищити продуктивність роботи на порядок, але за рахунок обмеження типів завдань, які можна буде вирішувати з їх допомогою.

Виділяють такі типи **мов четвертого покоління**:

- *мови представлення інформації, наприклад мови запитів або генератори звітів;*
- *спеціалізовані мови, наприклад мови електронних таблиць і баз даних;*
- *генератори додатків, які при створенні додатків забезпечують визначення, вставку, оновлення або витяг відомостей з бази даних;*
- *мови дуже високого рівня, призначені для генерацій коду додатків.*

В якості мов четвертого покоління можна вказати згадувані вище мови SQL і QBE. Розглянемо коротко деякі інші типи 4GL-мов.

Генератор форм *являє собою інтерактивний інструмент, призначений для швидкого створення шаблонів введення і відображення даних в екранних формах, дозволяє користувачеві визначити зовнішній вигляд екранної форми, її вміст і місце розташування на екрані.*

З його допомогою можна задавати кольори елементів екрану, а також інші характеристики, наприклад напівжирне, підкреслене мерехтливе або реверсивний креслення тощо. Більш досконалі генератори форм дозволяють створювати обчислювані атрибути з використанням арифметичних або узагальнюючих функцій, а також задавати правила перевірки введених даних.

Генератор звітів *є інструментом створення звітів на основі збереженої в базі даних інформації.*

Він подібний мови запитів в тому сенсі, що користувачеві надаються засоби створення запитів до бази даних і вилучення з неї інформації використовуваної для подання у звіті. Однак генератори звітів, як правило, передбачають набагато більші можливості управління зовнішнім виглядом звіту. Генератор звітів дозволяє або автоматично визначати вид одержуваних результатів, або за допомогою спеціальних команд створювати свій власний варіант зовнішнього вигляду документа, що друкується. Існує два основних типи генераторів звітів: мовний і візуальний.

У першому випадку для визначення потрібних для звіту даних і зовнішнього вигляду документа слід ввести відповідну команду на деякій підмові. У другому випадку для цих цілей використовується візуальний інструмент, подібний генератору форм.

Генератор графічного представлення даних - *цей генератор являє собою інструмент, призначений для вилучення інформації з бази даних і відображення її у вигляді діаграм з графічним представленням існуючих тенденцій і зв'язків. Звичайно за допомогою подібного генератора створюються гістограми, кругові, лінійчаті, точкові діаграми і т. д.*

Генератори додатків - *генератор додатків являє собою інструмент для створення програм, що взаємодіють з базою даних.*

Застосовуючи генератор додатків, можна скоротити час, необхідний для проектування повного обсягу необхідного прикладного програмного забезпечення. Генератори додатків зазвичай складаються з попереднього створення модулів, що містять фундаментальні функції, які потрібні для роботи більшості програм. Ці модулі, зазвичай створюються на мовах високого рівня, утворюють "бібліотеку" доступних функцій. Користувач вказує, які завдання програма повинна виконати, а генератор додатків визначає, як їх слід виконати.

Компоненти СУБД

СУБД складається з *декількох програмних компонентів (модулів)*, кожен з яких призначений для виконання специфічної операції.

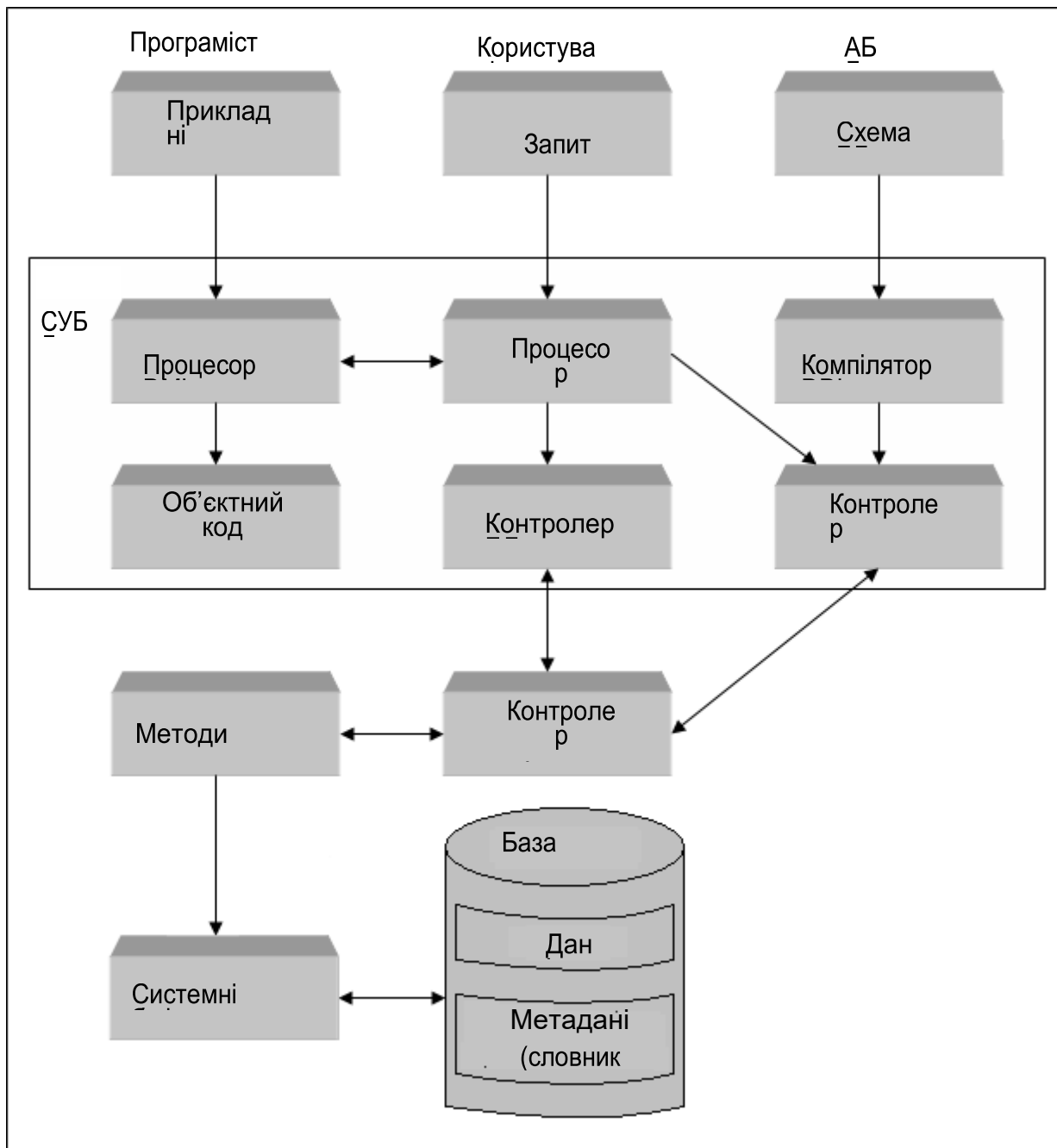
Деякі функції СУБД можуть підтримуватися операційною системою, що використовується. У такому випадку операційна система надає тільки базові служби та СУБД являє собою надбудову над ними.

При проектуванні СУБД слід враховувати *особливості інтерфейсу* між створюваною СУБД і конкретною операційною системою.

Основні програмні компоненти середовища СУБД представлені на рис.

6.8.

На цій схемі також показано, як СУБД взаємодіє з іншими програмними компонентами, наприклад з такими, як користувацькі запити і методи доступу (тобто методи управління файлами, використовувані при збереженні і витяганні записів з даними).



На рис. 6.8 показані найбільш важливі програмні компоненти середовища СУБД, що відповідають за доступ до даних

- **Процесор запитів.** Це основний компонент СУБД, який перетворює запити в послідовність низькорівневих інструкцій для контролера бази даних. СУБД повинна бути здатна обробляти запити користувача на вибірку, зміна або видалення даних, вже існуючих в базі, або на додавання в неї нових даних. Іншими словами, СУБД повинна включати в себе компонент процесора DML або компілятора DML, що забезпечує підтримку мови маніпулювання даними. В цілому, запити DML поділяються на *плановані і неплановані*.

Планований запит - це запит, необхідність виконання якого була передбачена попередньо. Адміністратор бази даних, можливо, повинен буде розробити фізичний проект бази даних таким чином, щоб гарантувати достатню швидкодію виконання подібних запитів.

Непланований запит - це, навпаки, деякий довільний запит на вибірку або (що менш ймовірно) на оновлення, необхідність виконання якого не була передбачена заздалегідь і виникла з якоїсь особливої причини. Фізичний проект бази даних може забезпечувати успішне виконання конкретного довільного запиту, або може виявитися не зовсім відповідним.

Плановані запити більш характерні для операційних, або виробничих програм, а неплановані - для додатків підтримки прийняття рішень. Більше того, плановані запити зазвичай надходять із заздалегідь підготовлених додатків, а неплановані запити за визначенням вводяться інтерактивно, за допомогою процесора мови запитів.

- **Контролер бази даних.** Цей компонент взаємодіє з запущеними користувачами прикладними програмами і запитамі. Контролер бази даних приймає запити і перевіряє зовнішні та концептуальні схеми для визначення тих концептуальних записів, які необхідні для задоволення вимог запиту. Потім контролер бази даних викликає контролер файлів для виконання запиту, що поступив. Компоненти контролера бази даних показані на рис. 6.9.

- **Контролер файлів** маніпулює призначеними для зберігання даних файлами і відповідає за розподіл доступного дискового простору. Він створює і підтримує список структур і індексів, визначених у внутрішньої схемою. Якщо використовуються хешовані файли, то в його обов'язки входить і виклик функцій хешування для генерації адрес записів. Контролер файлів не керує фізичним введенням і виведенням даних безпосередньо, а лише передає запити відповідним методам доступу, які зчитують дані в системні буфери або записують їх звідти на диск.

- **Препроцесор мови DML.** Цей модуль перетворює впроваджені в прикладні програми DML-оператори у виклики стандартних функцій базової мови. Для генерації відповідного коду препроцесор мови DML взаємодіє з процесором запитів.

- **Компілятор мови DDL.** Компілятор мови DDL перетворює DDL команди в набір таблиць, що містять метадані. Потім ці таблиці зберігаються в системному каталозі, а керуюча інформація - у заголовках файлів з даними.

- **Контролер словника.** Контролер словника управляє доступом до системного каталогу і забезпечує роботу з ним. Системний каталог доступний більшості компонентів СУБД.

Словник містить «дані про дані» (*метадані або дескриптори*), тобто в ньому знаходяться визначення інших об'єктів системи, а не просто звичайні дані. Зокрема, у словнику даних повинні бути записані вихідні та об'єктні форми всіх існуючих схем (зовнішніх, концептуальної тощо.) і відображень, а також встановлені обмеження захисту і цілісності даних. Розширений словник може також включати великий обсяг додаткової інформації, наприклад, про те, які з програм використовують ту чи іншу частину бази даних, які звіти потрібні кожному з користувачів і т.д. Словник може бути (а фактично просто повинен бути) інтегрований в визначену ним базу даних, а значить, повинен містити опис самого себе.

Основні програмні компоненти, що входять до складу контролера бази даних:

- **Контроль прав доступу.** Цей модуль перевіряє наявність у даного користувача повноважень для виконання запитованої операції

- **Процесор команд.** Після перевірки повноважень користувача для виконання затребуваної операції управління передається процесору команд.

- **Засоби контролю цілісності.** У разі операцій, які змінюють вміст бази даних, засоби контролю цілісності виконують перевірку того, чи задовольняє Незатребувана операція всім встановленим обмеженням підтримки цілісності даних (наприклад, вимогам, встановленим для ключів).

- **Оптимізатор запитів.** Цей модуль визначає оптимальну стратегію виконання запиту як планованого, так і непланованого. Потім оптимізовані запити виконуються під управлінням диспетчера етапу прогону (*run-time manager*). На практиці диспетчер етапу прогону для отримання доступу до збережених даних, можливо, буде використовувати щось подібне диспетчеру доступу до файлів (*контролер файлів*).

- **Контролер транзакцій.** Цей модуль здійснює необхідну обробку операцій, що надходять у процесі виконання транзакцій.

- **Планувальник.** Цей модуль відповідає за безконфліктне виконання паралельних операцій з базою даних. Він керує відносним порядком виконання операцій, що вимагаються в окремих транзакціях.

- **Контролер відновлень.** Цей модуль гарантує відновлення бази даних до несуперечливого стану при виникненні збоїв, в тому числі відповідає за фіксацію і скасування результатів виконання транзакцій

- **Контролер буферів.** Цей модуль відповідає за перенесення даних між оперативною пам'яттю і вторинним запам'ятовуючим пристроєм - наприклад, жорстким диском або

магнітною стрічкою. Контролер відновлення і контролер буферів іноді (у сукупності) називають **контролером даних**.

Для втілення бази даних на фізичному рівні крім перерахованих вище модулів потрібні деякі інші структури даних, до яких відносяться:

- Файли даних та індексів,
- Системний каталог.

Групою DAFTG (Database Architecture Framework Task Group) була зроблена спроба стандартизації СУБД, і в 1986 році нею була запропонована деяка еталонна модель. Призначення еталонної моделі полягає у визначенні концептуальних рамок для розділення спроб, що вживаються для стандартизації на більш керовані частини і вказання взаємозв'язків між ними на дуже широкому рівні.

Переваги та недоліки СУБД

СУБД володіють як багатообіцяючими потенційними перевагами, так і недоліками.

Переваги:

- Контроль за надмірністю даних.
- Несуперечність даних.
- Більше корисної інформації при тому ж обсязі збережених даних.
- Спільне використання даних
- Підтримка цілісності даних.
- Підвищена безпека.
- Застосування стандартів.
- Підвищення ефективності із зростанням масштабів системи.
- Можливість знаходження компромісу при суперечливих вимогах.
- Підвищення доступності даних та їх готовності до роботи.
- Покращення показників продуктивності.
- Спрощення супроводу системи за рахунок незалежності від даних.
- Покращене керування паралельністю.
- Розвинені служби резервного копіювання і відновлення.

Недоліки:

- Складність.
- Розмір.
- Вартість СУБД.
- Додаткові витрати на апаратне забезпечення.
- Витрати на перетворення.
- Продуктивність.
- Більш серйозні наслідки при виході системи з ладу.

Коротко розглянемо процес *передачі даних*. Запит кінцевого користувача до бази даних насправді передається від робочої станції користувача (яка може бути фізично віддалена від самої системи баз даних) до деякого інтерактивного програмного компоненту (вбудованого чи ні), а від нього - до СУБД у формі *комунікаційних повідомлень*. Більше того, відповіді користувачеві також передаються у формі подібних повідомлень (від СУБД або оперативного додатки до станції користувача). *Передача таких повідомлень відбувається під управлінням ще одного програмного компонента - диспетчера передачі даних*.

Диспетчер передачі даних не є частиною СУБД; він являє собою автономну систему зі своїми правами. Однак, оскільки очевидно, що від диспетчера передачі даних і СУБД вимагається злагоджена спільна робота, вони іноді розглядаються як рівноправні компоненти програмного продукту більш високого рівня, який називається **системою бази даних і передачі даних** (*DataBase / Data-Communication - DB / DC*). У такій системі СУБД *відповідає за роботу з базою даних, а диспетчер передачі даних обробляє всі повідомлення, що надходять в СУБД і з неї* (точніше, що надходять у той програмний компонент, в якому використовується СУБД, і з нього).

Архітектура багатокористувацьких СУБД

В даний час існують такі *основні типові архітектурні рішення*, які використовуються при реалізації *багатокористувацьких СУБД*, а саме:

- *зі схемами звичайної телеобробки*
- *файловим сервером,*
- *технологією "клієнт / сервер".*

Телеобробка

Традиційною архітектурою багатокористувацьких систем раніше вважалася схема, що отримала назву "**телеобробки**", при якій *один комп'ютер з єдиним процесором був з'єднаний з декількома терміналами* так, як показано на рис. 6.10. При цьому вся обробка виконувалася в рамках єдиного комп'ютера, а приєднані до нього термінали користувачів були *типовими "неінтелектуальними" пристроями*, не здатними функціонувати самостійно.

З центральним процесором термінали були пов'язані з допомогою кабелів, по яких вони посилали повідомлення прикладним програмам користувачів (через підсистему управління обміном даними операційної системи). У свою чергу, прикладні програми користувачів зверталися до

необхідних службам СУБД. Таким же чином повідомлення поверталися назад на термінала користувача.

При такій архітектурі основна і надзвичайно велике навантаження покладалася на центральний комп'ютер, який мав виконувати не тільки дії прикладних програм і СУБД, а й значну роботу з обслуговування терміналів (наприклад, форматування даних, що виводяться на екрани терміналів).

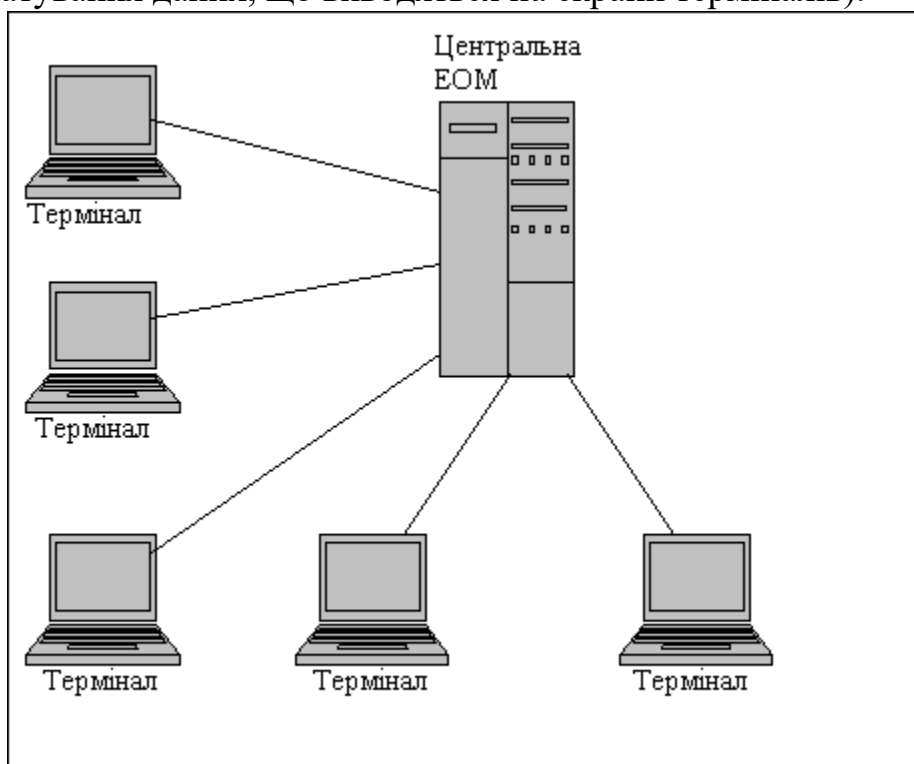


Рис. 5.10. Топологія архітектури телеобробки

В останні роки був досягнутий істотний прогрес у розробці високопродуктивних персональних комп'ютерів і складених з них мереж. При цьому у всій індустрії спостерігається помітна тенденція до децентралізації (downsizing), тобто заміні дорогих мейнфреймів більш ефективними, з точки зору експлуатаційних витрат, мережами персональних комп'ютерів, що дозволяють отримати такі ж результати, якщо не краще. Ця тенденція привела до появи наступних двох типів архітектури СУБД: технології файлового сервера і технології "клієнт / сервер".

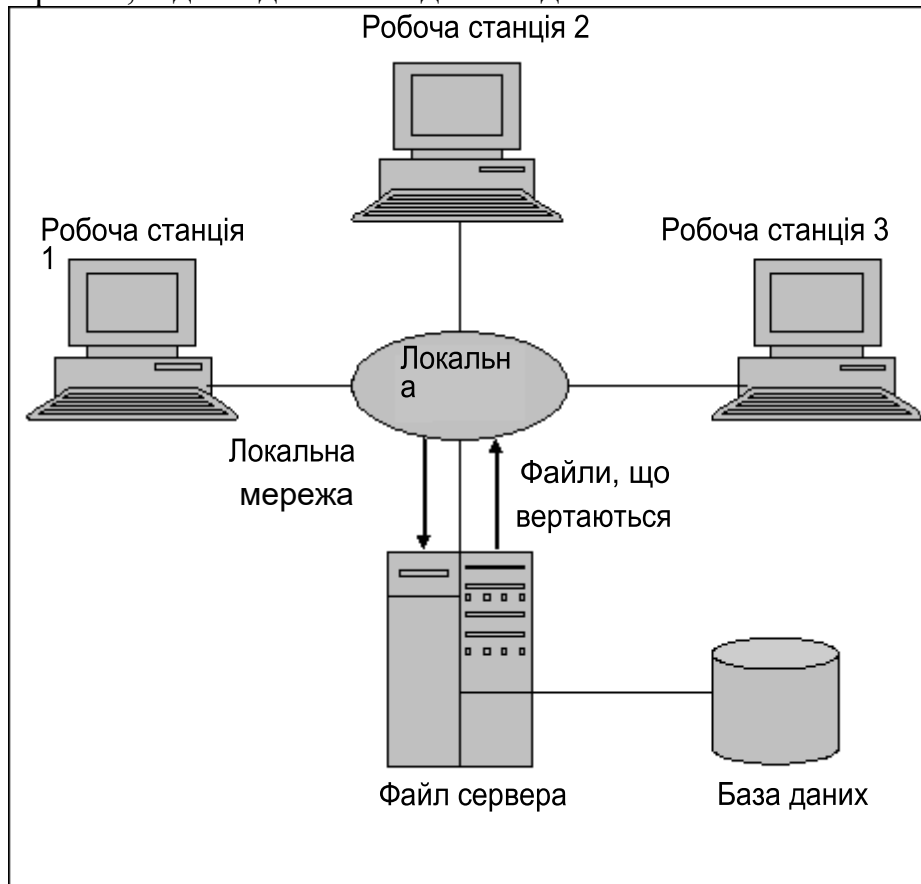
Файловий сервер

У середовищі файлового сервера обробка даних розподілена в мережі, зазвичай представляє собою локальну обчислювальну мережу (ЛОМ). Файловий сервер містить файли, необхідні для роботи прикладних програм і самої СУБД.

Прикладні програми користувачів і сама СУБД розміщені і функціонують на окремих робочих станціях, і звертаються до файлового

серверу тільки в міру необхідності отримання доступу до потрібних їм файлів - як показано на рис. 6.11.

Файловий сервер функціонує просто як спільного використання жорсткий диск. СУБД на кожній робочій станції посилає запити файлового серверу за всіма необхідними їй даними, які зберігаються на диску файлового сервера. Такий підхід характеризується *значним мережним трафіком*, що може призвести до *зниження продуктивності всієї системи в цілому*, так як файловий сервер не розуміє команд на мові SQL, тому СУБД запитує у файлового сервера файли, відповідні необхідним відношенням.



Архітектура з використанням *файлового сервера* володіє наступними

основними недоліками:

- великий обсяг *мережевого трафіку*
- на кожній робочій станції повинна знаходитися повна копія СУБД,
- управління *паралельністю, відновленням і цілісністю ускладнюється*, оскільки доступ до одних і тих же файлів можуть здійснювати відразу декілька екземплярів СУБД.

Технологія "клієнт / сервер"

Технологія "клієнт / сервер" була розроблена з метою усунення недоліків, наявних в перших двох підходах.

"Клієнт / сервер" означає такий спосіб взаємодії програмних компонентів, при якому вони утворюють єдину систему.

Існує якийсь клієнтський процес, що вимагає певних ресурсів, а також серверний процес, який ці ресурси надає.

При цьому зовсім необов'язково, щоб вони знаходилися на одному і тому ж комп'ютері. На практиці прийнято розміщувати сервер на одному вузлі локальної мережі, а клієнти - на інших вузлах. На рис. 6.12 показана архітектура типу "клієнт / сервер", а на рис. 6.13 - деякі можливі варіанти топології "клієнт / сервер".

У контексті бази даних клієнт керує інтерфейсом користувача і логікою програми, діючи як складна робоча станція, на якій виконуються прикладні програми баз даних.

Клієнт приймає від користувача запит, перевіряє синтаксис і генерує запит до бази даних на мові SQL або іншій мові бази даних, який відповідає логіці програми. Потім він передає повідомлення серверу, очікує надходження відповіді і форматує отримані дані для подання їх користувачеві.

Сервер приймає і обробляє запити до бази даних, а потім передає отримані результати назад клієнтові. Така обробка включає перевірку повноважень клієнта, забезпечення вимог цілісності, підтримку системного каталогу, а також виконання запиту і оновлення даних, при цьому підтримується управління паралельністю і відновленням.

Виконувані клієнтом і сервером операції наведено в табл. 6.2.

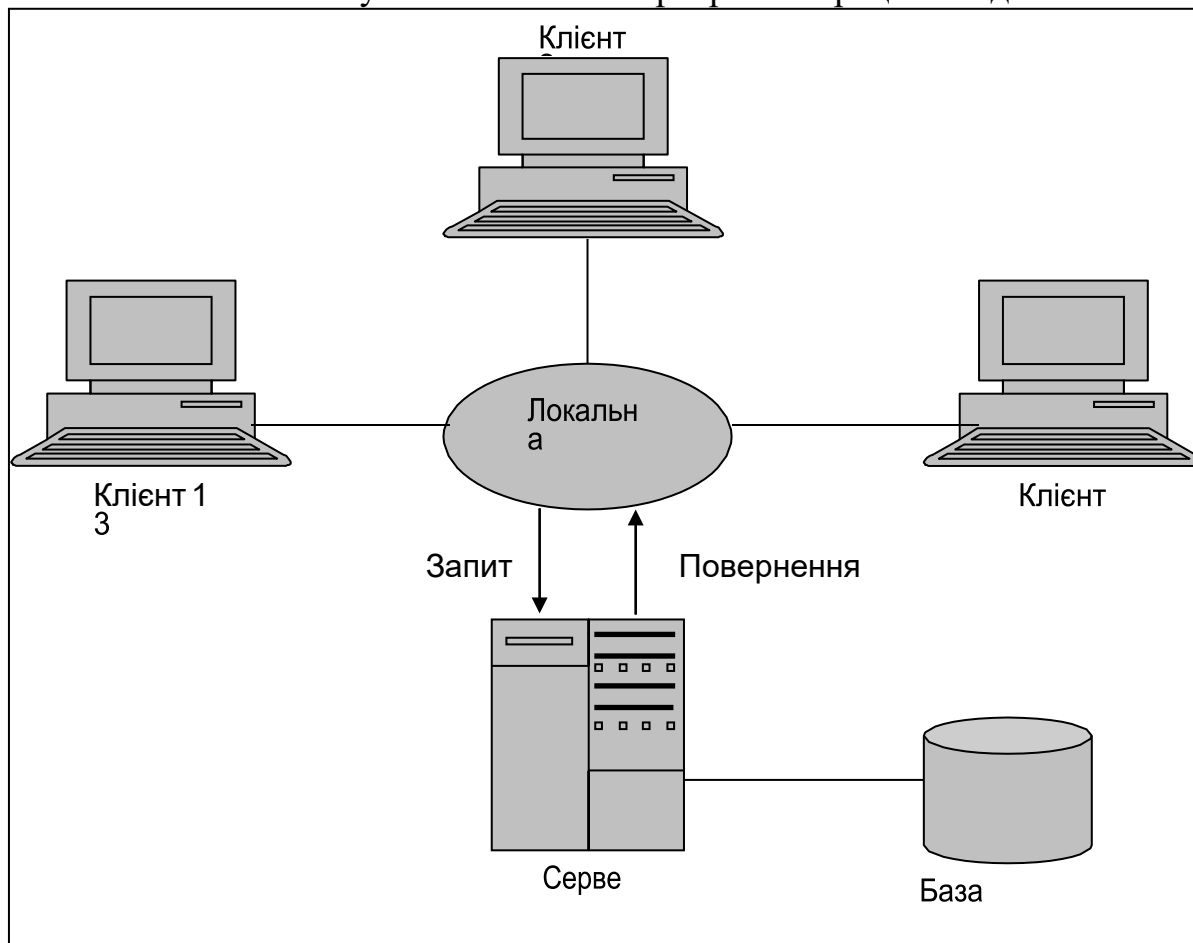


Рис 5.12. Загальна схема побудови систем з архітектурою "клієнт / сервер".

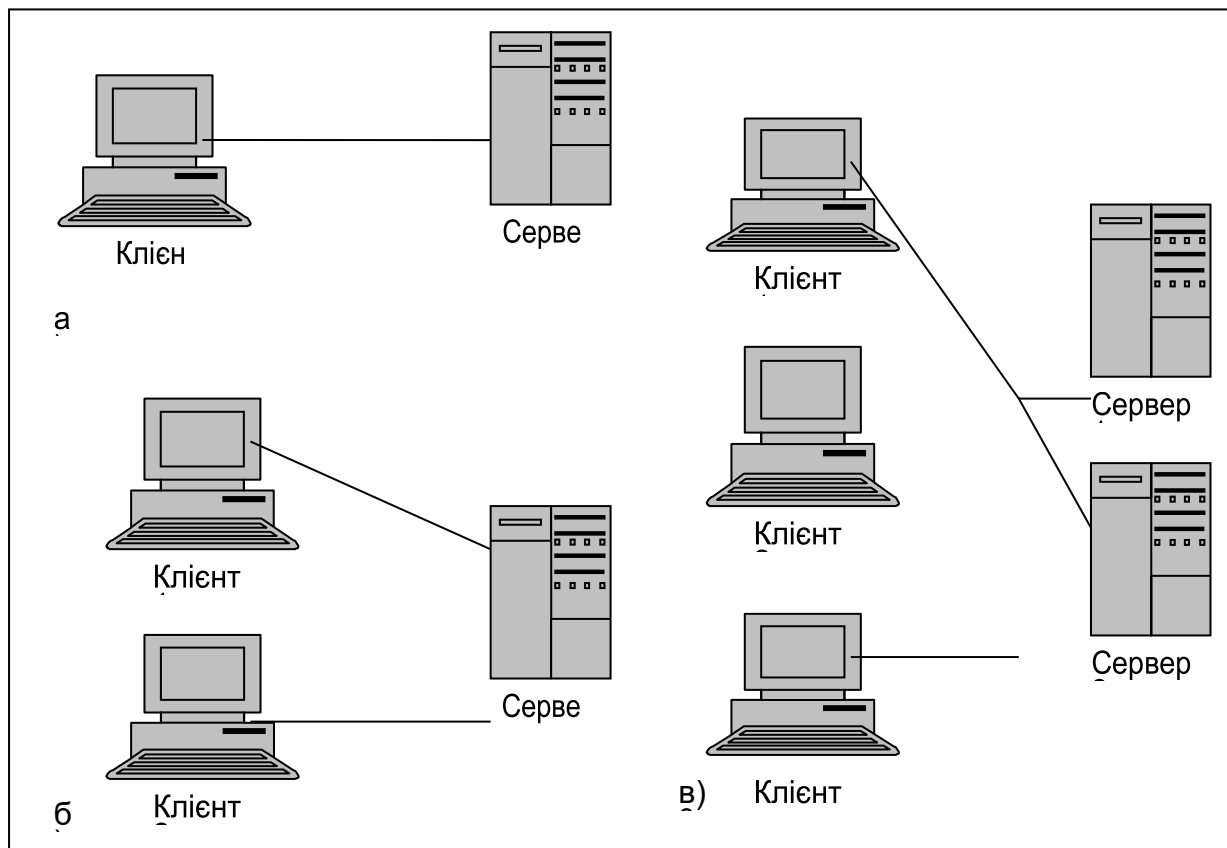


Рис 5.13. Альтернативні топології систем з архітектурою "клієнт / сервер":

а) один клієнт - один сервер, б) кілька клієнтів - один сервер, в) кілька клієнтів - кілька серверів.

Таблиця 6.2. Функції, що їх учасниками виконуються при взаємодії в середовищі "клієнт / сервер"

Клієнт	Сервер
Управляє інтерфейсом користувача	Приймає і перевіряє синтаксис введеного користувачем запиту
Приймає і обробляє запити до бази даних з боку клієнтів	Перевіряє повноваження користувачів
Виконує прикладні програми	Гарантує дотримання обмежень цілісності
Генерує запит до бази даних і передає його серверу	Виконує запити / оновлення і повертає результати клієнту
Відображає отримані дані користувачеві	Підтримує системний каталог Забезпечує паралельний доступ до БД Забезпечує управління відновленням

Цей тип архітектури володіє наведеними нижче перевагами:

- *Забезпечується більш широкий доступ до існуючих баз даних.*
 - *Підвищується загальна продуктивність системи,*
- оскільки клієнти і сервер знаходяться на різних комп'ютерах, їх

процесори здатні виконувати програми паралельно. При цьому налаштування продуктивності комп'ютера з сервером спрощується, якщо на ньому виконується тільки робота з базою даних.

- *Вартість апаратного забезпечення знижується*, досить потужний комп'ютер з великим пристроєм зберігання потрібен тільки серверу - для зберігання і управління базою даних.

- *Скорочуються комунікаційні витрати*, оскільки додатки виконують частину операцій на клієнтських комп'ютерах і посилають через мережу тільки запити до бази даних, що дозволяє істотно скоротити обсяг даних, що пересилаються по мережі.

- *Підвищується рівень несуперечності даних*, оскільки сервер може самостійно управляти перевіркою цілісності даних, при цьому всі обмеження визначаються і перевіряються тільки в одному місці і кожному додатку не доведеться виконувати власну перевірку.

- *Архітектура дуже природно відображається на архітектурі відкритих систем.*

Деякі розробники баз даних використовували цю архітектуру для організації засобів роботи з розподіленими базами даних, тобто з набором кількох баз даних, логічно пов'язаних і розподілених у комп'ютерній мережі. Однак, незважаючи на те, що *архітектура "клієнт / сервер"* цілком може бути використана для організації розподіленої - СУБД, сама по собі вона не утворює розподілену СУБД.

У загальному випадку кожен сервер може обслуговувати багато клієнтів, а кожен клієнт може працювати з багатьма серверами. Якщо система забезпечує повну прозорість доступу (тобто клієнт працює так, як ніби він має справу з одним сервером на єдиному комп'ютері, не враховуючи реального фізичного розміщення даних), то в наявності справжня **розподілена система баз даних**.

Подальше розширення дворівневої архітектури "клієнт / сервер" призвело до створення *трирівневої архітектури "клієнт / сервер"*, при якому функціональна частина колишнього, товстого (інтелектуального) клієнта розподіляється на дві частини.

У трирівневої архітектурі "клієнт / сервер" *тонкий (неінтелектуальний) клієнт на робочій станції управляє тільки інтерфейсом для користувача, тоді як середній рівень обробки даних управляє рештою логікою програми.*

Третім рівнем тут є сервер бази даних. Ця трирівнева архітектура виявилася більш підходящою для деяких середовищ - наприклад, для мереж Internet і Intranet, де в якості клієнта може використовуватися звичайний Web-браузер.

Системні каталоги

СУБД повинна мати доступний кінцевому користувачеві системний каталог чи словник даних.

Системний каталог - це сховище даних, які описують інформацію, що зберігається в базі даних, тобто метадані, або "дані про дані".

Системний каталог СУБД є одним з фундаментальних компонентів системи. Багато перерахованих раніше програмних компонентів будуються на використанні даних, що зберігаються в системному каталозі.

Наприклад, модуль контролю прав доступу використовує системний каталог для перевірки наявності у користувача повноважень, необхідних для виконання необхідних йому операцій.

Системний каталог для контролю прав доступу включає такі обов'язкові компоненти:

- імена користувачів, для яких дозволений доступ до бази даних;
- імена елементів даних в базі даних;
- елементи даних, до яких кожен користувач має право доступу, і дозволені типи доступу до них - для вставки, оновлення, видалення або читання.

Іншим прикладом можуть служити засоби перевірки цілісності даних, які використовують системний каталог для перевірки того, чи задовольняє необхідна операція всім встановленим обмеженням підтримки цілісності даних. Для виконання цієї перевірки в системному каталозі повинні зберігатися такі відомості:

- імена елементів даних з бази даних;
- типи і розміри елементів даних;
- обмеження, встановлені для кожного з елементів даних.

Як вже згадувалося раніше, термін "словник даних" часто використовується для програмного забезпечення більш загального типу, ніж просто каталог СУБД.

Система словника даних може бути або *пасивним*, або *активним*. Активна система завжди узгоджується зі *структурою бази даних*, оскільки вона автоматично підтримується цією системою.

Пасивна система може суперечити стану бази даних через ініційовані користувачами зміни.

Якщо словник даних є частиною бази даних, то він називається *інтегрованим словником даних*.

Ізольований словник даних має свою власну спеціалізовану СУБД. Його переважно використовують на початкових етапах проектування бази даних для деякої організації, коли потрібно відкласти на якийсь час прив'язку до конкретної СУБД. Однак *недолік цього підходу* полягає в тому, що після вибору СУБД і втілення бази даних *ізольований словник даних значно важче підтримувати в узгодженості з станом бази даних*.

Проблему узгодженості словника даних і бази даних можна звести до мінімуму, якщо перетворити словник, що використовувався при проектуванні даних, безпосередньо у каталог СУБД.

До недавнього часу ніякого вибору не було взагалі, але в міру розвитку стандартів словників даних ця ідея стає все більш реальною (можливо використовувати CASE-засоби, наприклад Erwin).

Служба IRDS, як стандарт словників даних

У багатьох системах словник даних є внутрішнім компонентом СУБД, в якому зберігається тільки інформація, безпосередньо пов'язана з цією базою даних. Однак збережені за допомогою СУБД дані зазвичай забезпечують тільки частину всіх інформаційних потреб.

Як правило, є деяка додаткова інформація, яка зберігається за допомогою інших інструментів - наприклад, таких як CASE - інструменти, інструменти ведення документації, інструменти налаштування і управління проектами. Кожне із згаданих вище засобів зазвичай має свій внутрішній словник даних, доступний і іншим зовнішнім інструментам.

Не існує ніякого універсального способу спільного використання різних наборів інформації різними групами користувачів або прикладними програмними компонентами.

Нещодавно була зроблена спроба стандартизувати інтерфейс словників даних для досягнення більшої доступності та спрощення їх спільного використання. Результатом стала розробка служби словника інформаційних ресурсів (*Information Resources Dictionary System - IRDS*).

Служба IRDS являє собою програмний інструмент, призначений для управління інформаційними ресурсами організації, а також для їх документування.

Вона включає:

- визначення таблиць, що містять словник даних,
- та операцій, які можуть бути використані для доступу до цих таблиць.

Згадані операції забезпечують несуперечливий метод доступу до словника даних і спосіб перетворення визначень даних зі словника одного типу в визначення іншого типу.

Наприклад, за допомогою служби IRDS інформація, що зберігається в IRDS - сумісному словнику даних СУБД DB2, може бути передана в IRDS-сумісний словник даних СУБД Oracle або спрямована деякому прикладному програмному компоненту системи DB2.

Одним з достоїнств служби IRDS є здатність розширення словника даних.

Так, якщо користувач захоче зберегти визначення нового типу інформації в якомусь інструменті (наприклад, визначення звітів про управління проектом - у якійсь СУБД), то система IRDS в даній СУБД може бути розширена для включення цієї інформації.

Визначення IRDS були прийняті в якості стандарту Міжнародною організацією стандартизації ISO (International Organization for Standardization) (ISO, 1990; 1993).

Стандарти IRDS визначають набір правил зберігання інформації в словнику даних і доступу до неї, переслідуючи при цьому три наступні цілі:

- розширюваність даних;
- цілісність даних;
- контрольований доступ до даних.

Служба IRDS побудована на використанні інтерфейсу сервісів, що складається з набору функцій, доступного для виклику з метою отримання доступу до словника даних.

Інтерфейс сервісів може викликатися з боку таких типів користувацьких інтерфейсів, як:

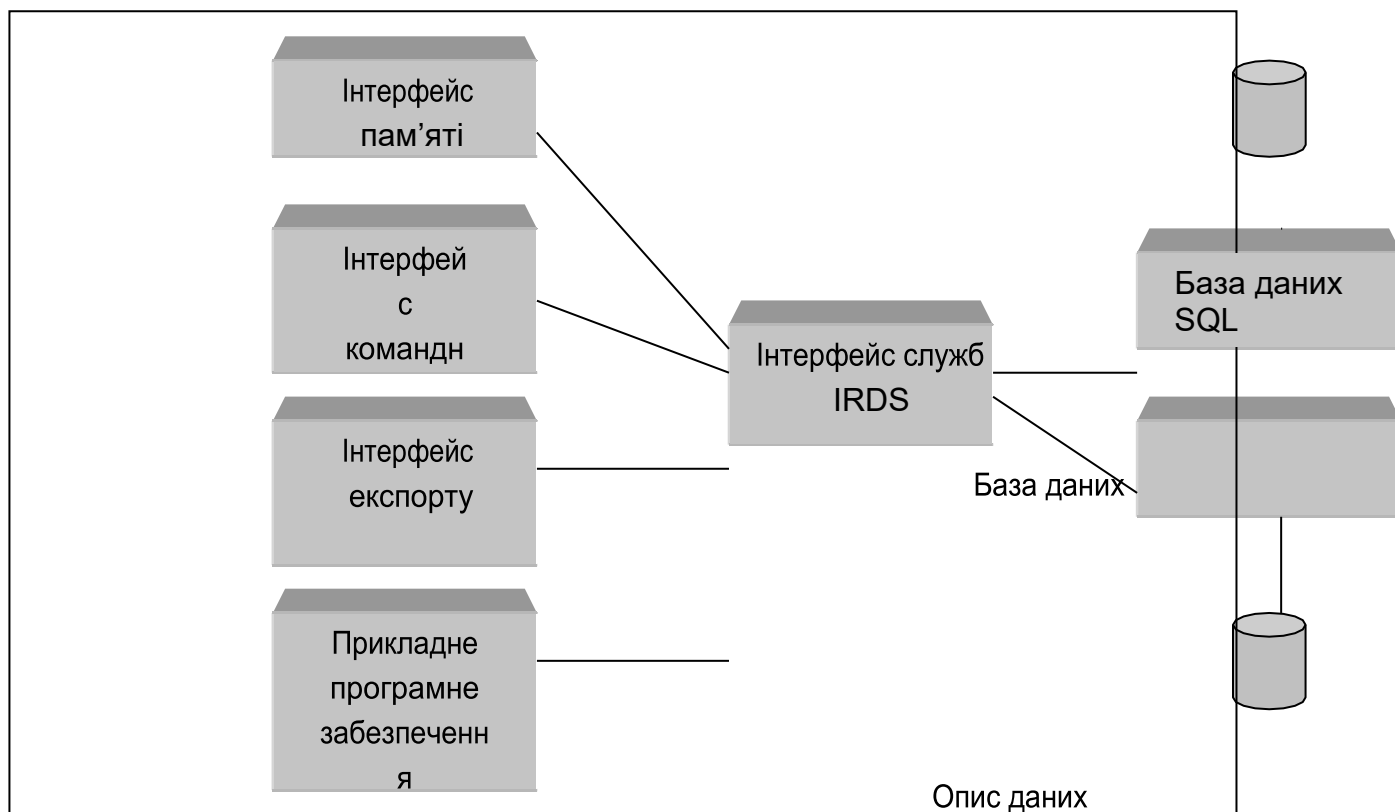
- панельний інтерфейс;
- командний мова;
- файли експорту / імпорту;
- прикладні програми.

Панельний інтерфейс складається з набору панелей або екранів, кожен з яких надає доступ до заздалегідь описаному набору сервісів, нагадує мову QBE і дозволяє користувачам переглядати і змінювати словник даних.

Інтерфейс командної мови складається з набору команд або операторів, які дозволяють виконувати маніпуляції зі словником даних. Інтерфейс командної мови може викликатися інтерактивно (за допомогою терміналу) або за допомогою впровадження операторів у програми на мовах програмування високого рівня.

Інтерфейс експорту / імпорту генерує файл, який можна передавати між різними IRDS-сумісними системами.

Стандарт IRDS визначає універсальний формат обміну інформацією, не вимагає, щоб база даних, що лежить в основі словника даних, відповідала якоїсь однієї моделі даних. Саме тому інтерфейс IRDS-сервісів здатний об'єднувати гетерогенні СУБД, як показано на рис. 6.14.



Утиліти

Утиліти - це програми, розроблені для адміністратора баз даних і використовуються ним при вирішенні різних адміністративних завдань.

Деякі утиліти виконуються на зовнішньому рівні системи і тому являють собою не що інше, як додаток спеціального призначення. Одні з них можуть бути створені навіть не постачальником даної СУБД, а якимись сторонніми розробниками. Однак інші утиліти виконуються на внутрішньому рівні і тому повинні надаватися постачальниками СУБД

Нижче наводиться кілька прикладів утиліт різних типів, які часто застосовуються на практиці.

- Інструменти **завантаження**, застосовувані для створення вихідної версії бази даних з одного або кількох файлів операційної системи.

- Інструменти **вивантаження-перезавантаження**, застосовуються для вивантаження бази даних або її частини, створення резервних копій збережених даних і відновлення бази даних з цих копій. (Безумовно, інструменти перезавантаження, по суті, ідентичні згаданим вище інструментам завантаження.)

- Інструменти **реорганізації**, застосовуються для перерозподілу даних у збереженій базі даних (зазвичай в цілях підвищення продуктивності). Зокрема, це може бути процедура групування даних на диску деяким спеціальним чином або звільнення простору, до того зайнятого даними, які більше не

використовуються.

- **Статистичні** інструменти, застосовуються для обчислення різних статистичних показників і показників продуктивності, таких як відомості про розміри файлів або значення даних, про кількість операцій введення-виведення тощо.

- Інструменти **аналізу**, застосовуються для аналізу згаданої вище статистичної інформації.

Цей список охоплює лише невелику частину функціональних можливостей, що зазвичай надаються доступними утилітами. Крім перерахованих, існує безліч інших функцій.

Запитання:

- 6.1. Поясніть мету і причини виникнення трирівневої архітектури ANSI-SPARC.
- 6.2. Дайте порівняльну характеристику цих рівнів.
- 6.3. Яка відмінність між логічної і фізичної незалежності даних?
- 6.4. Що визначають відображення «концептуальний-внутрішній» і «Зовнішній-концептуальний»?
- 6.5. Опишіть три типи схем бази даних.
- 6.6. Дайте визначення терміну СУБД. Визначте компоненти системи бази даних.
- 6.7. Опишіть п'ять компонентів середовища СУБД і поясніть, як вони пов'язані один з одним.
- 6.8. Поясніть ролі наступних груп користувачів бази даних:
 - а) адміністратор даних;
 - б) адміністратор бази даних;
 - в) проєктувальник логічної частини бази даних;
 - г) проєктувальник фізичної частини бази даних;
 - д) прикладний програміст;
 - е) звичайні користувачі.
- 6.9. Перелічіть і поясніть основні функції СУБД.
- 6.10. Поясніть призначення мови визначення даних (Data Definition Language-DDL) і мови управління даними (Data Manipulation Language-DML).
- 6.11. Назвіть основні компоненти СУБД.
- 6.12. Які компоненти входять до складу контролера бази даних?
- 6.13. Перерахуйте основні переваги і недоліки СУБД.
- 6.14. Що ви знаєте про систему управління передачею даних?
- 6.15. Назвіть основні типові архітектурні рішення, використовувані при реалізації багатокористувацьких СУБД.
- 6.16. Що означає термін «архітектура» клієнт / сервер "? Опишіть переваги, якими володіє ця схема. Порівняйте цієї архітектуру з двома іншими типовими архітектурами побудови СУБД.
- 6.17. Опишіть функції і загальне значення системного каталогу.

Вправи:

- 6.1. Проаналізуйте особливості тих СУБД, якими ви зараз користуєтеся. Визначте ступінь відповідності кожної з них тому набору функцій, які повинна надавати типова СУБД. Які типи мов підтримуються в кожній з них? Який тип архітектури використовується в кожній з цих СУБД? Перевірте ступінь доступності та розширюваності системного каталогу. Чи можна експортувати системний каталог в іншу систему?
- 6.2. Створіть програму, призначену для збереження в базі даних імен та номерів телефонів. Потім напишіть іншу програму, призначену для збереження в базі даних імен і адрес. Змініть ці програми з метою використання зовнішньої, концептуальної і внутрішньої схем. Які переваги і недоліки характерні для кожної з цих модифікацій?
- 6.3. Створіть програму, призначену для збереження в базі даних імен і дат народження. Розширте цю програму так, щоб вона зберігала в базі даних формат використовуваних даних. Інакше кажучи, створіть системний каталог. Розробіть інтерфейс, який зробить цей каталог доступним зовнішнім користувачам.
- 6.4. Як можна було б змінити попередню програму для приведення її у відповідність архітектурі типу "клієнт / сервер"?

ТЕМА 7

Мова SQL

7.1 Мова SQL: Маніпулювання даними

Введення в мову SQL

Призначення мови SQL

Будь-яка мова роботи з базами даних повинна надавати користувачеві наступні можливості:

- створювати бази даних і таблиці з повним описом їх структури;
- виконувати основні операції маніпулювання даними, такі як вставка, модифікація і видалення даних з таблиць;
- виконувати прості і складні запити.

Крім того, мова роботи з базами даних повинна вирішувати всі зазначені вище завдання при мінімальних зусиллях з боку користувача, а структура і синтаксис її команд повинні бути досить прості і доступні для вивчення. І, нарешті, вона повинна бути універсальною, тобто відповідати деякому визнаному стандарту, що дозволить використовувати один і той же синтаксис і структуру команд при переході від однієї СУБД до іншої. Мова SQL задовольняє практично всім цим вимогам. SQL є прикладом мови перетворення даних.

Мова перетворення даних — мова, призначена для роботи з таблицями з метою перетворення вхідних даних до необхідного вихідного виду.

Мова SQL, яка визначена стандартом ISO, має два основних компоненти:

- мова DDL (Data Definition Language), призначена для визначення структур бази даних і управління доступом до даних;
- мова DML (Data Manipulation Language), призначена для вибірки і поновлення даних.

До появи стандарту SQL3 мова SQL включала тільки команди визначення і маніпулювання даними; в ній були відсутні будь-які команди управління ходом обчислень. Іншими словами, в цій мові не було команд IF ... THEN ..., ELSE, GO TO, DO ... WHILE і будь-яких інших, призначених для управління ходом обчислювального процесу. Подібні завдання повинні були вирішуватися програмним шляхом (за допомогою мов програмування або управління завданнями) або інтерактивно (в результаті дій, які виконуються самим користувачем). Унаслідок подібної незавершеності (з точки зору організації обчислювального процесу) мова SQL могла використовуватися двома способами. Перший передбачав інтерактивну роботу, яка полягає у введенні користувачем з терміналу окремих операторів SQL. Другий полягав у впровадженні операторів SQL в програми на процедурних мовах. Формальне визначення мови SQL3 прийнято в 1999 році.

Мова SQL відносно проста у вивченні.

- Це непроцедурна мова, тому в ній необхідно вказувати, яка інформація повинна бути отримана, а не як її можна отримати. Інакше кажучи, мова SQL не вимагає вказівки методів доступу до даних.
- Як і більшість сучасних мов, SQL підтримує вільний формат запису операторів. Це означає, що при введенні окремі елементи операторів не пов'язані з фіксованими позиціями на екрані.
- Структура команд задається набором ключових слів, які представляють собою звичайні слова англійської мови, такі як CREATE TABLE (Створити таблицю), INSERT (Вставити), SELECT (Вибрати).
- Мова SQL може використовуватися широким колом користувачів, включаючи адміністраторів баз даних (АБД), керівний персонал компанії, прикладних програмістів і безліч інших кінцевих користувачів різних категорій.

В даний час для мови SQL існують міжнародні стандарти, які формально визначають її як стандартну мову створення і маніпулювання реляційними базами даних, якою вона фактично і є.

Історія мови SQL

Історія реляційної моделі даних (і побічно мови SQL) почалася в 1970 році з публікації основоположної статті Е. Ф. Кодда (в той час він працював у дослідницькій лабораторії корпорації IBM в Сан-Хосе). У 1974 році Д. Чемберлен, який працював в тій же лабораторії, публікує визначення мови, що отримав назву "Structured English Query Language", або SEQUEL. У 1976 році була випущена перероблена версія цієї мови, SEQUEL / 2; згодом її назву довелося змінити на SQL з юридичних міркувань - абревіатура SEQUEL вже використовувалася кимось раніше. Але до теперішнього часу багато як і раніше вимовляють абревіатуру SQL як "сіквел", хоча офіційно її рекомендується читати як "ес-кю-ел".

У 1976 році на базі мови SEQUEL / 2 корпорація IBM випустила прототип СУБД, що мав назву "System R". Призначення цієї пробної версії складалося в перевірці здійсненності реляційної моделі. Крім інших позитивних аспектів, найважливішим з результатів виконання цього проекту можна вважати розробку власне мови SQL, Однак коріння цієї мови йдуть в мову SQUARE (Specifying Queries as Rational Expressions), який був попередником проекту System R. Мова SQUARE була розроблена як дослідницький інструмент для реалізації реляційної алгебри за допомогою фраз, складених на англійській мові.

В кінці 1970-х років, компанією, яка нині перетворилася в корпорацію Oracle, була випущена СУБД Oracle. Мабуть, це найперша з комерційних реалізацій реляційної СУБД, побудованої на використанні мови SQL. Трохи пізніше з'явилася СУБД INGRES, що використала мову запитів QUEL. Ця мова була більш структурованою, ніж SQL, але її семантика менш близька до звичайної англійської мови. Пізніше, коли SQL була прийнята як стандартна мова реляційних баз даних, СУБД INGRES була повністю переведена на його використання. У 1981 році корпорація IBM випустила свою першу комерційну реляційну СУБД під назвою SQL / DS (для середовища DOS / VSE).

У 1982 році вийшла в світ версія цієї системи для середовища VM / CMS, а в 1983 році - для середовища MVS, але вже під назвою DB2. У 1982 році Національний

інститут стандартизації США (ANSI) почав роботу над мовою RDL (Relation Database Language), керуючись концептуальними документами, отриманими від корпорації IBM. У 1983 році до цієї роботи підключилася Міжнародна організація по стандартизації (ISO). Спільні зусилля обох організацій увінчалися випуском стандарту мови SQL. (Від назви RDL в 1984 році відмовилися, а чорновий проект мови був перероблений з метою наближення до вже існуючих реалізацій мови SQL.).

Початковий варіант стандарту, який був випущений ISO в 1987 році, викликав хвилю критичних зауважень. Зокрема, Дейт, відомий дослідник в цій області, вказував, що в стандарті опущені найважливіші функції, включаючи засоби забезпечення посилювальної цілісності і деякі реляційні оператори. Крім того, він зазначив надмірну надмірність мови - один і той же запит можна було записати в декількох різних варіантах. Велика частина критичних зауважень була визнана справедливою, і необхідні корективи були внесені в стандарт ще до його публікації. Однак було вирішено, що важливіше випустити стандарт якомога швидше (щоб він зміг виконувати роль загальної основи, на якій і сама мова, і його реалізації могли б розвиватися далі), ніж чекати, поки будуть визначені й узгоджені всі функції, які різні фахівці вважають обов'язковими для подібної мови.

У 1989 році ISO опублікувала додаток до стандарту, в якому визначалися функції підтримки цілісності даних. У 1992 році була випущена перша, істотно переглянута версія стандарту ISO, яку іноді називають SQL2 (або SQL-92). Хоча деякі з функцій були визначені в цьому стандарті вперше, багатьох них вже були повністю або частково реалізовані в одній або декількох комерційних реалізаціях мови SQL. А наступна версія стандарту, яку прийнято називати SQL3, була випущена тільки в 1999 році. Ця версія містить додаткові засоби підтримки об'єктно-орієнтованих функцій управління даними.

Функції, які додаються до стандарту мови розробниками комерційних реалізацій, прийнято називати розширеннями. Наприклад, в стандарті мови SQL визначено шість різних типів даних, які можуть зберігатися в базах даних. У багатьох реалізаціях цей список доповнюється різноманітними розширеннями. Кожна з реалізацій мови називається діалектом. Не існує двох абсолютно ідентичних діалектів, як в даний час не існує і жодного діалекту, який повністю відповідає стандарту ISO. Більш того, оскільки розробники баз даних вводять в системи все нові функціональні засоби, вони постійно розширюють свої діалекти мови SQL, в результаті чого окремі діалекти все більше і більше відрізняються один від одного. Однак основне ядро мови SQL залишається більш-менш стандартизованим у всіх реалізаціях.

Хоча базові концепції мови SQL були розроблені корпорацією IBM, її важливість дуже скоро підштовхнула і інших розробників до створення власних реалізацій. В даний час на ринку доступні буквально сотні продуктів, побудованих на використанні мови SQL, причому постійно доводиться чути про випуск все нових і нових версій.

Особлива роль мови SQL

Мова SQL є першою і поки єдиною стандартною мовою роботи з базами даних, яка отримала досить широке поширення. Є ще одна стандартна мова роботи з базами даних, NDL (Network Database Language), яка побудована на використанні мережевої

моделі CODASYL, але вона застосовується лише в небагатьох розробках. Практично всі найбільші розробники СУБД в даний час створюють свої продукти з використанням мови SQL або інтерфейсу SQL, і більшість таких компаній беруть участь в роботі, щонайменше, однієї організації, яка займається розробкою стандартів цієї мови. У SQL зроблені величезні інвестиції, як з боку розробників, так і з боку користувачів. Вона стала частиною архітектури додатків (наприклад, такий як System Application Architecture (SAA) корпорації IBM), а також є стратегічним вибором багатьох великих і впливових організацій (наприклад, консорціуму X / Open, зайнятого розробкою стандартів для середовища UNIX). Мова SQL також прийнята в якості федерального стандарту обробки інформації (Federal Information Processing Standard - FIPS), який повинен дотримуватися в СУБД для отримання дозволу продавати її на території США. Консорціум розробників SQL Access Group докладає зусиль зі створення розширень мови SQL, які дозволять забезпечити взаємодію різнорідних систем.

Мова SQL використовується в інших стандартах і навіть впливає на розробку багатьох стандартів як інструмент їх визначення. Як приклад можна привести стандарти ISO "Information Resource Dictionary System" (IRDS) і "Remote Data Access" (RDA).

Нагадуємо. Служба IRDS являє собою програмний інструмент, призначений для управління інформаційними ресурсами організації, а також для їх документування. Вона включає визначення таблиць, що містять словник даних, і операцій, які можуть бути використані для доступу до цих таблиць.

Розробка мови викликала певну зацікавленість наукових кіл, що виразилася як у виробленні необхідних теоретичних основ, так і в підготовці успішно реалізованих технічних рішень. Це особливо справедливо по відношенню до оптимізації запитів, методів розподілу даних і реалізації засобів захисту. Почали з'являтися спеціалізовані реалізації мови SQL, призначені для нових ринків, такі як OnLine Analytical Processing (OLAP).

Використовувана термінологія

Стандарт ISO SQL не підтримує таких формальних термінів, як *відношення*, *атрибут* і *кортеж*, замість них застосовуються терміни *таблиця*, *стовпець* і *рядок*. Під час обговорення мови SQL в основному спираються на термінологію ISO.

Нагадуємо.

Відношення — плоска таблиця, що складається з стовпців і рядків.

Атрибут — іменований стовпець відношення.

Домен — набір допустимих значень одного або декількох атрибутів.

Кортеж — рядок відношення.

Ступінь — ступінь відношення визначається кількістю атрибутів, яке воно містить,

Кардинальність — кількість кортежів, яке міститься в відношенні.

Реляційна база даних — набір нормалізованих відношень, які розрізняються за своїми назвами.

Крім того, слід зазначити, що стандарт SQL не передбачає строгої підтримки визначень реляційної моделі даних. Наприклад, в мові SQL допускається, що створена в результаті виконання операції SELECT таблиця може містити повторювані

рядки, встановлюється певна послідовність стовпців, а користувачеві дозволяється сортувати рядки в таблиці.

Запис операторів SQL

Оператор SQL складається із зарезервованих слів, а також зі слів, які визначаються користувачем. Зарезервовані слова є постійною частиною мови SQL і мають певне значення. Їх слід записувати саме так, як вказано в стандарті, і не можна розбивати на частини для переносу з одного рядка в іншу. Слова, які визначаються користувачем, задаються самим користувачем (відповідно до визначених синтаксичними правилами) і являють собою імена різних об'єктів бази даних, таблиць, стовпців, уявлень, індексів і т.д. Слова в операторі розміщуються відповідно до встановлених синтаксичними правилами. Хоча в стандарті це не вказано, багато діалектів мови SQL вимагають завдання в кінці оператора деякого символу, що позначає закінчення його тексту; як правило, з цією метою додати крапку з комою (;).

Більшість компонентів операторів SQL не чутливі до регістру. Це означає, що можуть використовуватися будь-які літери — як малі, так і великі. Одним важливим винятком з цього правила є символічні літерали.

Символьні літерали – дані, які повинні вводитися точно так же, як були введені відповідні їм значення, що зберігаються в базі даних.

Наприклад, якщо в базі даних зберігається значення прізвища 'ІВАНОВ', а в умові пошуку вказано символічний літерал 'Іванов', то цей запис не буде знайдено.

Оскільки мова SQL має вільний формат, окремі оператори SQL та їх послідовності матимуть більш зручний для читання вигляд при використанні відступів і вирівнювання. Рекомендується дотримуватися наступних правил.

- Кожна конструкція в операторі повинна починатися з нового рядка.
- Початок кожної конструкції має бути позначено таким же відступом, що і початок інших конструкцій оператора.
- Якщо конструкція складається з декількох частин, кожна з них повинна починатися з нового рядка з деяким відступом щодо початку конструкції, що буде вказувати на їх підпорядкованість.

Будемо застосовувати наступну розширену форму системи позначень BNF (Backus Naur Form - форма Бекуса-Наура):

- Прописні букви будуть використовуватися для запису зарезервованих слів і повинні вказуватися в операторах точно так само, як це буде показано.
- Рядкові літери будуть використовуватися для запису слів, які визначаються користувачем.
- Вертикальна межа (|) вказує на необхідність вибору одного з декількох наведених значень, наприклад b | c.
- Фігурні дужки визначають обов'язковий елемент, наприклад {a}.
- Квадратні дужки визначають необов'язковий елемент, наприклад [a].
- Три крапки (...) використовують для зазначення необов'язкової можливості повторення конструкції від нуля до декількох раз, наприклад {a b} [, c ...]. Цей запис означає, що після a чи b може слідувати від нуля до декількох повторень c, розділених комами.

На практиці для визначення структури бази даних (в основному її таблиць) використовуються оператори DDL, а для заповнення цих таблиць даними та вибірки з них інформації за допомогою запитів - оператори DML. У цьому розділі спочатку познайомимося з операторами DML і лише потім звернемося до операторів мови DDL. Подібний підхід свідчить про велику важливість операторів DML з точки зору пересічного користувача. Основні оператори DDL розглядаються в темі 7.2.

Маніпулювання даними

В базі «Учебний процес» є наступні таблиці:

СТУДЕНТ (НГ, НС, ПІБ, ДАТАНАР, АДРЕСА, ПБАЛ)

ГРУПА (НГ, КОЛ, ПБАЛ)

КАФЕДРА (ККАФ, НКАФ, ТЕЛ, ЗАВ, ФОТО)

ВИКЛАДАЧ (ТАБН, ПІБ, СТ, ЗВ, ККАФ)

ПРЕДМЕТ (КП, НП, ГОДИНИ, ЛЕК, ПР, ЧС, ПРОГР)

ВИВЧЕННЯ (НГ, КП, ТАБН, ВИДЗ, ГОДИНИ, СБАЛГР)

УСПІШНІСТЬ (НГ, НС, КП, ТАБН, ВИДЗ, ОЦІНКА),

які заповнені відповідними даними (див. додаток)

Літерали

Літерали являють собою константи, які використовуються в операторах *SQL*.

Існують різні форми літералів для кожного типу даних, які підтримуються *SQL*. Зазначимо відмінності між літералами, які слід поміщати в одинарні лапки, і тими, які не слід. Всі *нечислові* значення даних завжди повинні поміщатись в одинарні лапки, а всі *числові* дані не повинні поміщатись в одинарні лапки. Нижче наведено приклад використання літералів для вставки даних в таблицю.

```
INSERT INTO СТУДЕНТ (НГ, НС, ПІБ, ДАТАНАР, АДРЕСА, ПБАЛ)
```

```
VALUES { 'ТМ-31', 26, 'Чортів', 1986, NULL, 4,3};
```

Значення стовпців НС і ДАТАНАР є літералом цілочисленного типу, а значення стовпця ПБАЛ — це десятковий числовий літерал. Жоден з них не повинен поміщатись в одинарні лапки. Значення всіх інших стовпців представляють собою символічні рядки і обов'язково повинні бути взяті в одинарні лапки.

Прості запити

Призначення оператора *SELECT* полягає у вибірці і відображенні даних однієї або більше таблиць бази даних. Це виключно потужний оператор, здатний виконувати дії, еквівалентні операторам реляційної алгебри вибірки, проекції і з'єднання, причому в межах єдиної виконуваної команди.

Нагадуємо. Операція вибірки застосовується до одного відношення *R* і визначає результуюче відношення, яке містить тільки ті кортежі (рядки) з відношення *R*, які задовольняють заданій умові (предикату).

Операція проекції застосовується до одного відношення *R* і визначає нове відношення, що містить вертикальну підмножину відношення *R*, створювану за допомогою видалення значень зазначених атрибутів і виключення з результату рядків-дублікатів.

Об'єднання двох відношень R і S визначає нове відношення, яке включає всі кортежі, що містяться тільки в R, тільки в S, одночасно в R і S, причому всі дублікати кортежів виключені. При цьому відношення R і S повинні бути сумісними за об'єднанням.

Оператор SELECT є найчастіше використовуваною командою мови SQL. Загальний формат оператора SELECT має наступний вигляд:

```
SELECT [DISTINCT | ALL] { * | [columnExpression [AS newName]] [ , ... ] }  
FROM TableName [alias] [ , ... ]  
[WHERE condition]  
[GROUP BY columnList] [HAVING condition]  
[ORDER BY columnList]
```

Тут параметр columnExpression являє собою ім'я стовпця або вираз з декількох імен. Параметр TableName є ім'ям існуючої в базі даних таблиці (або представлення), до якої відбувається підключення. Необов'язковий параметр alias - це скорочення, яке встановлюється для імені таблиці TableName. Обробка елементів оператора SELECT виконується в наступній послідовності.

- **FROM.** *Визначаються імена використовуваної таблиці або декількох таблиць.*
- **WHERE.** *Виконується фільтрація рядків об'єкта відповідно до заданих умов.*
- **GROUP BY.** *Утворюються групи рядків, що мають одне і те ж значення в зазначеному стовпці.*
- **HAVING.** *Фільтруються групи рядків об'єкта відповідно до зазначеної умови.*
- **SELECT.** *Встановлюється, які стовпці повинні бути присутніми в вихідних даних.*
- **ORDER BY.** *Визначається впорядкованість результатів виконання оператора.*

Порядок конструкцій в операторі SELECT не може бути змінений. Тільки дві конструкції оператора - SELECT і FROM - є обов'язковими, всі інші конструкції можуть бути опущені. Операція вибірки за допомогою оператора SELECT є замкнутою, в тому сенсі, що результат запиту до таблиці також є таблицею. Існує безліч варіантів використання даного оператора, що ілюструється наведеними нижче прикладами.

Приклад 7.1.1. Вибірка всіх стовпців і всіх рядків

Складіть список докладних відомостей про кожного зі студентів.

Оскільки в наведеному вище запиті не вказано жодних обмежень, в оператор не потрібно поміщати конструкцію WHERE. Крім того, необхідно вибрати всі існуючі в таблиці стовпці. Тому даний запит записується в такий спосіб:

```
SELECT ИГ, ИС, ПИБ, ДАТАНАР, АДРЕСА, ПБАЛ  
FROM СТУДЕНТ;
```

Оскільки вибірка всіх наявних в таблиці стовпців виконується досить часто, в мові SQL визначено спрощений варіант запису значення "всі стовпці" - замість імен стовпців вказується символ зірочки (*). Наведений нижче оператор повністю еквівалентний першому і являє собою спрощений варіант запису того ж самого запиту:

```
SELECT *  
FROM СТУДЕНТ;
```

Таблиця 7.1.1. Результат виконання запиту із прикладу 7.1.1

Група	№ студента	ПІБ	РІК НАРОДЖЕННЯ	АДРЕСА	Прохідний бал
001	01	Васильєв М.В.	1985		5
001	02	Носков А.К.	1976		4,1
101	01	Аристов Р.П.	1979		4,5
101	02	Бондаренко С.А.	1978		4,5
101	03	Борисова А.І.	1979		4,25
101	04	Макова Н.В.	1977		4,75
102	01	Боярська Н.П.	1977		4,5
102	02	Федоров Д.К.	1977		4,25
102	03	Сидоров І.Р.	1977		4,5
103	01	Андрєєв Г.М.	1978		4,25
103	02	Петров О.К.	1979		4,75
104	01	Іванов К.К.	1977		4,5
104	07	Григор'єва Е.А.	1979		4,75
104	14	Степанов І.В.	1978		3
105	01	Серов І.Г.	1980		4,5
105	12	Антоненко К. П.	1978		4,5
201	12	Карпов Р.А.	1984		3,8
201	23	Сомов О.В.	1983		4
403	01	Кромова В.Ю.	1987		4
403	09	Пимонов Г.Р.	1988		4
ТМ-31	07	Глушко Р.В.	1986		4,5
ТМ-31	12	Костюшко Т.А.	1986		4,25
ТМ-31	19	Самойленко Ю.А.	1986		4,75
ТМ-31	20	Святненко Ю.В.	1986		4,75
ТМ-31	23	Терпило Д.А.	1986		4,75
ТМ-31	24	Черняк А.Ю.	1986		4,25

Приклад 7.1.2. Вибірка конкретних стовпців і всіх рядків

Створіть звіт про прохідний бал всіх студентів із зазначенням тільки номера студента (НС + НС), його імені та прізвища, а також відомостей про бал.

```
SELECT НГ, НС, ПІБ, Пбал
FROM СТУДЕНТ
```

У цьому прикладі на основі таблиці СТУДЕНТ створюється нова таблиця, яка включає тільки зазначені в запиті стовпці НГ, НС, ПІБ, ПБАЛ, причому саме в цьому порядку. Результат виконання даного запиту наведено в табл. 7.1.2. Зверніть

увагу, що рядки в результуючій таблиці можуть виявитися несортованими, якщо не дано спеціальну вказівку. З іншого боку, в деяких СУБД сортування результуючої таблиці виконується за замовчуванням з урахуванням значень одного або декількох стовпців (наприклад, в базі даних Microsoft Access сортування цієї результуючої таблиці була б виконана за значенням первинного ключа НГ + НС). Способи сортування рядків в результуючій таблиці розглядаються в наступному розділі.

Таблиця 7.1.2. Результат виконання запиту з прикладу 7.1.2

Група	№ студента	ПІБ	Прохідний бал
001	01	Васильєв М.В.	5
001	02	Носков А.К.	4,1
101	01	Аристов Р.П.	4,5
101	02	Бондаренко С.А.	4,5
101	03	Борисова А.І.	4,25
101	04	Макова Н.В.	4,75
102	01	Боярська Н.П.	4,5
102	02	Федоров Д.К.	4,25
102	03	Сидоров І.Р.	4,5
103	01	Андрєєв Г.М.	4,25
103	02	Петров О.К.	4,75
104	01	Іванов К.К.	4,5
104	07	Григор'єва Е.А.	4,75
104	14	Степанов І.В.	3
105	01	Серов І.Г.	4,5
105	12	Антоненко К. П.	4,5
201	12	Карпов Р.А.	3,8
201	23	Сомов О.В.	4
403	01	Кромова В.Ю.	4
403	09	Пимонов Г.Р.	4
ТМ-31	07	Глушко Р.В.	4,5
ТМ-31	12	Костюшко Т.А.	4,25
ТМ-31	19	Самойленко Ю.А.	4,75
ТМ-31	20	Святненко Ю.В.	4,75
ТМ-31	23	Терпило Д.А.	4,75
ТМ-31	24	Черняк А.Ю.	4,25

Приклад 7.1.3. Використання ключового слова DISTINCT

Складіть список номерів усіх номерів груп в яких вивчаються якісь предмети
 SELECT НГ
 FROM ВИВЧЕННЯ;

Результат виконання цього запиту представлений в табл. 7.1.3. Зверніть увагу, що результат виконання запиту містить повторювані значення, оскільки, на відміну від операції проєкції реляційної алгебри, оператор SELECT не виключає повторюваних значень при виконанні проєкції за значеннями одного або декількох

стовпців. Для видалення з результуючої таблиці повторюваних рядків використовується ключове слово DISTINCT. Скоригований запит виглядає наступним чином:

```
SELECT DISTINCT НГ
FROM ВИВЧЕННЯ;
```

Результати виконання другого варіанту запиту представлені в табл. 7.1.4.

Таблиця 7.1.3. Результат виконання запиту зі збереженням значень, що повторюються з прикладу 7.1.3

Ном. групи
001
001
101
101
101
101
101
101
101
101
101
101
101
101
102
102
102
102
105
201
201
202
203
204
403
ТМ-31

Таблиця 7.1.4. Результат виконання запиту з виключенням повторюваних значень з прикладу 7.1.3

Ном. групи
001
101
102
105
201
202

203
204
403
ТМ-31

Приклад 7.1.4. обчислювані поля

Створіть звіт про кількість годин вивчення кожного предмета в кожному семестрі.

```
SELECT КП, НП, ГОДИНИ / НС
FROM ПРЕДМЕТ;
```

Результати виконання запиту представлені в табл. 7.1.5.

Таблиця 7.1.5. Результат виконання запиту з прикладу 7.1.4

Код предмета	Назва предмета	Col3
01	Інформатика	50
02	Математика	50
03	Історія	46,6
04	Іноземна мова	50
05	Філософія	50
06	Фізкультура	50
07	Програмування	70
08	Право	100
09	Фізика	72,5
10	Електродинаміка	290
11	Психологія	90

Це приклад використання в запиті обчислюваного поля (інколи ці поля називають *розрахунковими*, або *похідними*). У загальному випадку для створення обчислюваного поля в списку SELECT слід вказати деякий вираз мови SQL. У цих виразах можуть застосовуватися операції додавання, віднімання, множення і ділення. При побудові складних виразів можуть використовуватися круглі дужки. Для отримання значення обчислюваного поля можуть використовуватися значення з кількох стовпців таблиці, однак тип даних тих стовпців, які входять в арифметичні вирази, обов'язково повинен бути цифровим.

У таблиці, отриманої в результаті виконання запиту, четвертий стовпець називається col3. Зазвичай стовпцями результуючої таблиці присвоюються імена відповідних їм стовпців вихідних таблиць бази даних. Однак в даному випадку це правило не застосовується, оскільки в стандарті SQL не визначені правила іменування похідних стовпців. В одних діалектах мови SQL імена таким стовпцям привласнюють відповідно до порядку їх розташування в таблиці (наприклад, col3), в інших діалектах у подібного стовпця ім'я може бути зовсім відсутнім або замість нього може використовуватися вираз, що

знаходиться в списку SELECT. Стандарт ISO дозволяє явно задавати інші імена стовпців результуючої таблиці, для чого застосовується конструкція AS. При використанні цієї конструкції наведений вище оператор SELECT може бути переписаний наступним чином:

```
SELECT КП, НП, ГОДИНИ / НС AS К-сть_годин / семестр
```

```
FROM ПРЕДМЕТ;
```

В цьому випадку третій стовпець результуючої таблиці буде називатися К-сть_годин / семестр, а не col3.

Вибірка рядків (конструкція WHERE)

У наведених вище прикладах в результаті виконання операторів SELECT вибиралися всі рядки зазначеної таблиці. Однак дуже часто потрібно тим або іншим чином обмежити набір рядків, які розміщені в результуючій таблиці запиту. Це досягається за допомогою вказівки в запиті конструкції WHERE. Вона складається з ключового слова WHERE, за яким слідує перелік умов пошуку, що визначають ті рядки, які повинні бути обрані при виконанні запиту. Існує п'ять основних типів умов пошуку (або *предикатів*, якщо користуватися термінологією ISO). Сравнивание. Сравниваются результаты вычисления одного выражения с результатами вычисления другого выражения.

- Діапазон. Перевіряється, чи потрапляє результат обчислення виразу в заданий діапазон значень.
- Належність до безлічі. Перевіряється, чи належить результат обчислення виразу до заданої множини значень.
- Відповідність шаблону. Перевіряється, чи відповідає деяке строкове значення заданому шаблону.
- Значення NULL. Перевіряється, чи містить даний стовпець NULL (невизначене значення).

Конструкція WHERE еквівалентна операції вибірки реляційної алгебри. Розглянемо приклади використання всіх зазначених типів умов пошуку.

Приклад 7.1.5. Умова пошуку шляхом порівняння

Перерахуйте всіх студентів з прохідним балом > 4.75.

```
SELECT НГ, НС, ПІБ, ПБАЛ
```

```
FROM СТУДЕНТ
```

```
WHERE ПБАЛ > 4.75;
```

У цьому запиті використовуються таблиця СТУДЕНТ і предикат ПБАЛ > 4.75. При виконанні запиту буде створена нова таблиця, яка містить тільки ті рядки таблиці СТУДЕНТ, в яких значення стовпця ПБАЛ більше 4.75. Результати виконання запиту представлені в табл. 7.1.6.

Таблиця 7.1.6. Результат виконання запиту з прикладу 7.1.5

Група	№ студента	ПІБ	Прохідний бал
001	01	Васильєв	5

У мові SQL можна використовувати прості операції порівняння, перераховані в табл 7.1.7.

Таблиця 7.1.7. операції порівняння

Знак операції	Призначення
=	дорівнює
<>	НЕ дорівнює (стандарт ISO)
!-	НЕ дорівнює (використовують в деяких діалектах)
<	менше
>	більше
<=	Менше або дорівнює
>=	Більше або дорівнює

Більш складні предикати можуть бути побудовані за допомогою логічних операцій AND, OR та NOT, а також за допомогою дужок, що використовуються для визначення порядку обчислення виразу (якщо це необхідно або бажано). Обчислення виразів в умовах виконується за такими правилами.

- Вираз обчислюється зліва направо.
- Першими обчислюються підвирази в дужках.
- Операції NOT виконуються перед операціями AND і OR.
- Операції AND виконуються перед операціями OR.

Для усунення будь-якої можливої неоднозначності рекомендується використовувати круглі дужки.

Приклад 7.1.6. Складні умови пошуку

Перерахуйте всіх викладачів, які є професорами або асистентами.

SELECT *

FROM ВИКЛАДАЧ

WHERE ЗВ = 'професор' OR ЗВ = 'асистент';

У цьому прикладі для вибірки відомостей про вчені звання викладачів, в конструкції WHERE використовується логічний оператор OR. Результати виконання запиту представлені в табл. 7.1.8.

Таблиця 7.1.8. Результат виконання запиту з прикладу 7.1.6

Таб.номер	ПІБ викл.	Наук.ступінь	Наук. звання	Код кафедри
101	Андрєєв А.П.	д-р техн. наук	професор	01
201	Блюмкіна І.П.	д-р ф-мат. наук	професор	02
202	Львова П.Р.		асистент	02
203	Шапошніков С.І.	д-р техн. наук	професор	02
204	Новіков П.Н.		асистент	02
405	Кудряшова Г.М.		асистент	04
503	Єрмолін Е.Н.		асистент	05
801	Якорнов Є.А.	д-р техн. наук	професор	08
604	Соловйов С.І.		асистент	06

Приклад 7.1.7. Використання діапазонів (BETWEEN / NOT BETWEEN) в умовах пошуку

Перерахуйте всіх студентів з прохідним балом від 4,5 до 4,75

```
SELECT НГ, НС, ПІБ, ПБАЛЛ
```

```
FROM СТУДЕНТ
```

```
WHERE ПБАЛЛ BETWEEN 4,5 AND 4,75;
```

Наявність ключового слова BETWEEN вимагає завдання кордонів діапазону значень. В даному випадку результати перевірки будуть позитивними для всіх студентів з прохідним балом від 4,5 до 4,75 включно. Результати виконання запиту представлені в табл. 7.1.9.

Таблиця 7.1.9. Результати виконань запиту з прикладу 7.1.7

Група	№ студента	ПІБ	Прохідний бал
101	01	Аристов Р.П.	4,5
101	02	Бондаренко С.А.	4,5
101	04	Макова Н.В.	4,75
102	01	Боярська Н.П.	4,5
102	03	Сидоров І.Р.	4,5
103	02	Петров О.К.	4,75
104	01	Іванов К.К.	4,5
104	07	Григор'єва О.А.	4,75
105	01	Серов І.Г.	4,5
105	12	Антоненко К. П.	4,5
ТМ-31	07	Глушко І.В.	4,5
ТМ-31	19	Самойленко Ю.А.	4,75
ТМ-31	20	Святненко Ю.В.	4,75
ТМ-31	23	Терпило Д.А.	4,75

Існує також версія перевірки діапазону значень, яка має протилежний зміст (NOT BETWEEN). В цьому випадку потрібно, щоб значення, яке перевіряється, лежало поза межами заданого діапазону. Наявність ключового слова BETWEEN і відповідної перевірки лише незначно підвищує виразну потужність мови SQL, оскільки ті ж результати можуть бути досягнуті за допомогою виконання двох звичайних перевірок. Наведений вище запит можна представити таким чином:

```
SELECT НГ, НС, ПІБ, ПБАЛЛ
```

```
FROM СТУДЕНТ
```

```
WHERE ПБАЛЛ BETWEEN >= 4,5 AND <= 4,75
```

Однак багато хто вважає, що перевірка приналежності до діапазону за допомогою ключового слова BETWEEN є більш простим способом записи умов вибірки, ніж звичайні перевірки.

Приклад 7.1.8. Умови пошуку з перевіркою приналежності до безлічі (IN / NOT IN)

Складіть список всіх викладачів, які є професорами або асистентами.

```
SELECT ТАБН, ПІБ, СТ, ЗВ, ККАФ
FROM ВИКЛАДАЧ
WHERE ЗВ IN ( 'професор', 'асистент');
```

Перевірка приналежності до безлічі забезпечується за допомогою ключового слова IN. При цьому перевіряється, чи відповідає результат обчислення виразу одному зі значень в наданому списку - в нашому випадку це рядки 'професор' і 'асистент'. Результати виконання запиту представлені в табл. 7.1.10.

Таблиця 7.1.10. Результат виконання запиту з прикладу 7.1.8

Таб. номер	ПІБ викл.	Наук. ступінь	Наук. звання	Код кафедри
101	Андрєєв А.П.	д-р техн. наук	професор	01
201	Блюмкіна І.П.	д-р ф-мат. наук	професор	02
202	Львова П.Р.		асистент	02
203	Шапошніков С.І.	д-р техн. наук	професор	02
204	Новіков П.Н.		асистент	02
405	Кудряшова Г.М.		асистент	04
503	Єрмолін Е.Н.		асистент	05
801	Якорнов Є.А.	д-р техн. наук	професор	08
604	Соловійов С.І.		асистент	06

Існує також версія такої перевірки, що має протилежний зміст (NOT IN), яка використовується для відбору будь-яких значень, крім тих, які вказані в представленому списку. Як і оператор BETWEEN, оператор IN незначно підвищує виразну потужність мови SQL - той же самий запит може бути представлений таким чином:

```
SELECT ТАБН, ПІБ, СТ, ЗВ, ККАФ
FROM ВИКЛАДАЧ
WHERE ЗВ = 'професор' OR ЗВ = 'асистент');
```

Однак використання оператора IN є більш ефективний спосіб запису умов пошуку, особливо якщо набір допустимих значень досить великий.

Приклад 7.1.9. Умови пошуку із зазначенням шаблонів (LIKE / NOT LIKE)

Знайдіть всіх зав. кафедрою, в прізвищах яких міститься рядок 'Ж'

При виконанні даного запиту необхідно організувати пошук рядка 'Ж', яка може перебувати в будь-якому місці значень стовпця ПІБ_зав.кафедрою таблиці КАФЕДРА.

```
SELECT ККАФ, НКАФ, ТЕЛ, ЗАВ
FROM КАФЕДРА
WHERE ЗАВ LIKE '% Ж%';
```

Результати виконання цього запиту представлені в табл. 7.1.11.

Таблиця 7.1.11. Результат виконання запиту з прикладу 7.1.9

Код	Назва	ТЕЛ	ПІБ зав. каф.
04	Іноземної мови	310-47-17	Жданова А.Є.

У мові SQL існують два спеціальних символи шаблону, використовуваних при перевірці символічних значень.

- %. Символ відсотка являє будь-яку послідовність з нуля або більше символів (тому часто іменується також як знак підстановки).
- _. Символ підкреслення представляє будь-який окремий символ.
- Всі інші символи в шаблоні представляють самі себе.
- • ЗАВ LIKE 'Ж%'. Цей шаблон означає, що перший символ значення обов'язково повинен бути символом Н, а всі інші символи не представляють інтересу і не перевіряються.
- • ЗАВ LIKE 'Ж ____'. Цей шаблон означає, що значення має мати довжину, рівну строго чотирьом символам, причому першим символом обов'язково повинен бути символ 'Ж'.
- • ЗАВ LIKE '% а'. Цей шаблон визначає будь-яку послідовність символів довжиною не менше одного символу, причому останнім символом обов'язково повинен бути символ а.
- ЗАВ LIKE 'Ж'. Цей шаблон означає, що нас цікавить будь-яка послідовність символів, що включає підрядок 'Ж';
- ЗАВ NOT LIKE 'K%'. Цей шаблон вказує, що потрібні будь-які рядки, які не починаються з символу К.

Якщо потрібний рядок повинен включати також службовий символ, звичайно застосовуваний в якості символу підстановки, то слід визначити за допомогою конструкції ESCAPE "Маскуючий" символ, який вказує, що наступний за ним символ більше не має спеціального значення, і помістити його перед символом підстановки. Наприклад, для перевірки значень на відповідність літерального рядку '15%' можна скористатися таким предикатом: LIKE '15 #%' ESCAPE ' # '

Приклад 7.1.10. Використання значення NULL в умовах пошуку (IS NULL / IS NOT NULL)

Складіть список всіх викладачів с кодом кафедри 02, для яких немає даних про їх наукового ступеня.

Переглядаючи таблицю ВИКЛАДАЧ, можна прийти до висновку, що дані записи можна вибрати за допомогою наступних умов пошуку:

(ККАФ = '02 'AND СТ = ' ')

Однак це рішення помилково. Відсутній коментар (значення NULL) розглядається як невизначене значення, тому не можна визначити його рівність чи нерівність з іншого рядком. Якщо спробувати виконати запит до наведеного вище умов пошуку, то результуюча таблиця виявиться порожньою. Правильне рішення полягає в явній перевірці на наявність порожнього значення, для чого використовується спеціальне ключове слово IS NULL:

```
SELECT ПІБ, ТАБН  
FROM ВИКЛАДАЧ  
WHERE ККАФ = '02 'AND СТ = IS NULL;
```

Результати виконання запиту представлені в табл. 7.1.12.

Таблиця 7.1.12. Результат виконання запиту з прикладу 7.1.10

Таб. номер	ПІБ викл.
202	Львова П.Р.

Для перевірки присутності в стовпці значень, відмінних від NULL, може використовуватися версія цієї умови пошуку, яка має протилежний СЕНС (IS NOT NULL).

Сортування результатів (конструкція ORDER BY)

У загальному випадку рядки в результуючій таблиці запиту SQL не впорядковані яким-небудь певним чином (хоча в деяких СУБД може бути передбачено застосування за замовчуванням певного способу впорядкування, наприклад по первинному ключу). Однак їх можна впорядкувати належним чином, для чого в оператор SELECT поміщається конструкція ORDER BY. Конструкція ORDER BY включає список розділених комами ідентифікаторів стовпців, по яких потрібно впорядкувати результуючу таблицю запиту. Ідентифікатор стовпчика може бути або його ім'я, або номер 1, який позначає елемент списку SELECT відповідно до його позиції в цьому списку. Лівий елемент списку має номер 1, наступний - 2 і т.д. Номери стовпців можуть використовуватися в тих випадках, коли стовпчики, за якими слід упорядкувати результат, є обчислюваними, а конструкція AS із зазначенням імені цього стовпця в операторі SELECT відсутня. Конструкція ORDER BY дозволяє впорядкувати вибрані записи в порядку зростання (ASC) чи зменшення (DESC) значень будь-якого стовпця або комбінації стовпців, незалежно від того, присутні ці стовпці в таблиці результатів чи ні. Однак в деяких діалектах SQL потрібно, щоб конструкція ORDER BY обов'язково була присутня в списку вибірки оператора SELECT. У будь-якому випадку конструкція ORDER BY завжди повинна бути останнім елементом в операторі SELECT.

Приклад 7.1.11. Сортування за значеннями одного стовпчика

Складіть звіт про повний загальний середній прохідний бал всіх груп, розташувавши рядки в порядку убутання бала.

```
SELECT НГ, КІЛ, ПБАЛ
FROM ГРУПА
ORDER BY ПБАЛ DESC;
```

Дані слід упорядкувати по спадаючій значень в стовпці ПБАЛ. Це досягається за допомогою приміщення в кінець оператора SELECT конструкції ORDER BY, яка задає сортування результуючої таблиці по спадаючій значень в стовпці ПБАЛ. А оскільки значення повинні розташовуватися в порядку убутання, то вказано ключове слово DESC. Результати виконання запиту представлені в табл. 7.1.13. Зверніть увагу, що конструкція ORDER BY може бути записана і в наступному вигляді: ORDER BY 3 DESC. Тут 3 позначає третій стовпець в списку вибірки оператора SELECT, тобто стовпець ПБАЛ.

Таблиця 7.1.13. Результат виконання запиту з прикладу 7.1.11

№ групи	К-сть. ст. в групі	Прох.бал
101	15	4,80
105	35	4,80
203	28	4,70
001	25	4,60
ТМ-31	6	4,45

103	18	4,25
104	19	4,20
102	20	4,00
202	30	4,00
204	25	4,00
403	2	4,00
201	35	3,90

У конструкції ORDER BY може бути зазначено і більше одного елемента. Старший ключ сортування визначає загальну упорядкованість рядків результуючої таблиці. У попередньому прикладі старшим ключем сортування є стовпець ПБАЛ. Якщо значення старшого ключа сортування у всіх рядках результуючої таблиці є унікальними, немає необхідності використовувати додаткові ключі сортування. Однак, якщо значення старшого ключа не унікальні, в результуючій таблиці будуть присутні кілька рядків з одним і тим же значенням старшого ключа сортування. У цьому випадку може виявитися бажаним впорядкувати рядки з одним і тим же значенням старшого ключа з якого-небудь додаткового ключа сортування. Якщо в конструкції ORDER BY присутні другий і наступні елементи, то такі елементи називають молодшими ключами сортування.

Приклад 7.1.12. Сортування за кількома стовпцями

Складіть звіт про середній прохідний бал всіх груп, розташувавши рядки в порядку убудування бала і кількості студентів в групі.

```
SELECT НГ, КІЛ, ПБАЛ
FROM ГРУПА
ORDER BY ПБАЛ DESC;
```

Після виконання цього запиту буде створена результуюча таблиця (табл. 7.1.14).

Таблиця 7.1.14. Результат виконання запиту з прикладу 12 з сортуванням по одним стовпці

№ групи	К-сть. ст. в групі	Прох.бал
101	15	4,80
105	35	4,80
203	28	4,70
001	25	4,60
ТМ-31	6	4,45
103	18	4,25
104	19	4,20
102	20	4,00
202	30	4,00
204	25	4,00
403	2	4,00
201	35	3,90

В отриманих результатах присутні відомості по двох групах з прохідним балом 4,80 і по чотирьох групах з прохідним балом 4,00. Якщо не вказано молодший ключ сортування, система розташує ці рядки в довільному порядку. Для того щоб упорядкувати їх, наприклад, в порядку зменшення кількості студентів К-сть, слід додатково вказати молодший ключ сортування:

```
SELECT НГ, КІЛ, ПБАЛ
FROM ГРУПА
ORDER BY ПБАЛ, К-сть DESC;
```

Тепер результат виконання запиту буде впорядкований спочатку по прохідному балу (за спаданням), а в межах одного типу об'єкта - в порядку убутання значень кількості студентів в групі. Результати виконання нового варіанту запиту представлені в табл. 7.1.15.

Таблиця 7.1.15. Результат виконання запиту з прикладу 7.1.12 з сортуванням по двох стовпчиках

№ групи	К-сть. ст. в групі	Прох.бал
105	35	4,80
101	15	4,80
203	28	4,70
001	25	4,60
ТМ-31	6	4,45
103	18	4,25
104	19	4,20
202	30	4,00
204	25	4,00
102	20	4,00
403	2	4,00
201	35	3,90

У стандарті ISO зазначено, що значення NULL в стовпці або виразах, для сортування яких застосовується конструкція ORDER BY, повинні розглядатися або як менші, або як великі за величиною, ніж всі непусті значення. Вибір того чи іншого варіанту залишений на розсуд розробників СУБД.

Використання агрегуючих функцій мови SQL

Стандарт ISO містить визначення наступних п'яти *агрегуючих функцій*:

- COUNT — повертає кількість значень у вказаному стовпці;
- SUM – повертає суму значень в зазначеному стовпці;
- AVG – повертає усереднене значення в зазначеному стовпці;
- MIN – повертає мінімальне значення в зазначеному стовпці;
- MAX – повертає максимальне значення у вказаному стовпці.

Всі ці функції оперують зі значеннями в єдиному стовпці таблиці і повертають єдине значення. Функції COUNT, MIN і MAX застосовні як до числових, так і до нечислових полів, тоді як функції SUM і AVG можуть використовуватися тільки в разі числових полів. За винятком COUNT (*), при обчисленні результатів будь-яких функцій спочатку виключаються всі порожні значення, після чого необхідна операція застосовується тільки до решти непустих значень стовпця. Варіант COUNT (*) є особливим випадком використання функції COUNT - його призначення полягає в підрахунку всіх рядків в таблиці, незалежно від того, містяться там порожні, повторювані або будь-які інші значення.

Якщо до застосування агрегує функції необхідно виключити повторювані значення, слід перед ім'ям стовпця у визначенні функції помістити ключове слово DISTINCT. Стандарт ISO допускає використання ключового слова ALL з метою явної вказівки того, що виключення повторюваних значень не потрібно, хоча це ключове слово має на увазі за замовчуванням, якщо ніякі інші визначники не задані. Ключове слово DISTINCT не має сенсу для функцій MIN і MAX. Однак його використання може впливати на результати виконання функцій SUM і AVG, тому слід заздалегідь обмірковувати, чи має воно бути присутнім в кожному конкретному випадку. Крім того, ключове слово DISTINCT в кожному запиті може бути зазначено не більше одного разу.

Слід зазначити, що агрегує функції можуть використовуватися тільки в списку вибірки SELECT і в конструкції HAVING. У всіх інших випадках застосування цих функцій неприпустимо. Якщо список вибірки SELECT містить агрегує функцію, а в тексті запиту відсутня конструкція GROUP BY, що забезпечує об'єднання даних в групи, то жоден з елементів списку вибірки SELECT не може включати будь-яких посилань на стовпці, за винятком випадку, коли цей стовпець використовується як параметр агрегує функції. Наприклад, наступний запит є некоректним:

```
SELECT КП, COUNT (ГОДИНИ)
FROM ВІВЧЕННЯ;
```

Помилка полягає в тому, що в даному запиті відсутня конструкція GROUP BY, а звернення до колонки КП в списку вибірки SELECT виконується без застосування агрегуючої функції.

Приклад 7.1.13. Використання функції COUNT (*)

Визначте, скільки студентів мають прохідний бал більше 4,50.
 SELECT COUNT (*) AS count FROM СТУДЕНТ WHERE ПБАЛ > 4,5;
 Обмеження на підрахунок тільки тих студентів, які мають прохідний бал більше 4,50, реалізується за допомогою використання конструкції WHERE. Загальна кількість студентів, що відповідають вказаній умові, може бути визначено за допомогою агрегуючої функції COUNT. Результати виконання запиту представлені в табл. 7.1.16.

Таблиця 7.1.16. Результат виконання запиту з прикладу 7.1.13

count
7

Приклад 7.1.14. Використання функції COUNT (DISTINCT)

Визначте, скільки різних груп мають 30 і більше (але не більше 160) годин лекцій або практики з якого-небудь предмету

```
SELECT COUNT (DISTINCT НГ) AS count
```

FROM ВІВЧЕННЯ

WHERE ГОДИННИК BETWEEN 30 AND 160;

І в цьому випадку обмеження результатів запиту аналізом тільки тих груп, в яких було від 30 до 160 годин лекційних або практичних занять, досягається за допомогою використання конструкції WHERE. Загальна група, яка відповідає вказаній умові, може бути визначена за допомогою агрегуючої функції COUNT. Однак, оскільки одна і та ж група може мати таку кількість годин по лекціях і практиці одночасно або одночасно з кількох предметів, необхідно у визначенні функції вказати ключове слово DISTINCT - це дозволить виключити з розрахунку повторювані значення. Результати виконання запиту представлені в табл. 7.1.17.

Таблиця 7.1.17. Результат виконання запиту з прикладу 7.1.14

count
7

Приклад 7.1.15. Використання функцій COUNT і SUM

Визначте загальну кількість груп, де викладає Андреев А.П. (Табличний номер - 101) і обчисліть суму годин його роботи в усіх групах.

SELECT COUNT (НГ) AS count, SUM (ГОДИННИК) AS sum

FROM ВІВЧЕННЯ

WHERE ТАБН = '101';

Обмеження на відбір відомостей тільки про даний викладача досягається зазначенням в запиті відповідної конструкції WHERE. Загальна кількість груп і сума годин визначаються шляхом застосування до результуючої таблиці запиту агрегуючих функцій COUNT і SUM. Результати виконання запиту представлені в табл. 7.1.18.

Таблиця 7.1.18. Результат виконання запиту з прикладу 7.1.15

count	sum
4	80

Приклад 7.1.16. Використання функцій MIN, MAX і AVG

Розрахуйте значення мінімального, максимального і середнього прохідного бала.

SELECT MIN (ПБАЛЛ) AS min, MAX (ПБАЛЛ) AS max, AVG (ПБАЛЛ) AS avg
FROM СТУДЕНТ;

У цьому прикладі необхідно обробити відомості про всіх студентів, тому використовувати конструкцію WHERE не потрібно. Необхідні значення можуть бути обчислені за допомогою функцій MIN, MAX і AVG, що застосовуються до колонки ПБАЛ таблиці СТУДЕНТ. Результати виконання запиту представлені в табл. 7.1.19.

Таблиця 7.1.19. Результат виконання запиту з прикладу 7.1.16

min	max	avg
3	5	4.2

Групування результатів (конструкція GROUP BY)

Наведені вище приклади зведених даних подібні підсумковим рядкам, зазвичай розміщуються в кінці звітів. У підсумках все детальні дані звіту стискаються в один узагальнюючий рядок. Однак дуже часто в звітах потрібно формувати і проміжні підсумки. Для цієї мети в операторі SELECT може вказуватися конструкція GROUP BY.

*Запит, в якому присутня конструкція GROUP BY, називається **групуючим запитом**, оскільки в ньому групуються дані, отримані в результаті виконання операції SELECT, після чого для кожної окремої групи створюється єдиний підсумковий рядок. Стовпці, перераховані в конструкції GROUP BY, називаються **згрупованими стовпцями**.*

Стандарт ISO вимагає, щоб конструкції SELECT і GROUP BY були тісно пов'язані між собою. При використанні в операторі SELECT конструкції GROUP BY кожен елемент списку в списку вибірки SELECT повинен мати *єдине значення для всієї групи*. Більш того, конструкція SELECT може включати тільки такі типи елементів:

- Імена стовпців;
- Агрегуючі функції;
- Константи;
- Вирази, що ключають комбінації перерахованих вище елементів;

Всі імена стовпців, наведені в списку вибірки SELECT, повинні бути присутніми і в конструкції GROUP BY, за винятком випадків, коли ім'я стовпця використовується тільки в агрегуючій функції. Протилежне твердження не завжди справедливо - в конструкції GROUP BY можуть бути присутніми імена стовпців, відсутні в списку вибірки SELECT. Якщо спільно з конструкцією GROUP BY використовується конструкція WHERE, то вона обробляється в першу чергу, а групуванню піддаються тільки ті рядки, які задовольняють умові пошуку.

Стандартом ISO визначено, що при проведенні групування всі відсутні значення розглядаються як рівні. Якщо два рядки таблиці в одному і тому ж групуючому стовпці містять значення NULL і ідентичні значення у всіх інших непустих групуючих стовпцях, вони поміщаються в одну і ту ж групу.

Приклад 7.1.17. Використання конструкції GROUP BY

Визначте кількість студентів в кожній групі, а також їх середній прохідний бал.

```
SELECT НГ, COUNT (HC) AS count, AVG (ПБАЛ) AS avg
FROM СТУДЕНТ
GROUP BY НГ
ORDER BY НГ;
```

Немає необхідності включати імена стовпців HC і ПБАЛЛ в список елементів GROUP BY, оскільки вони з'являються тільки в списку вибірки SELECT з агрегуючими функціями. У той же час стовпець НГ в списку конструкції SELECT не пов'язаний з якою-небудь агрегуючою функцією і з цієї причини обов'язково повинен бути зазначений в конструкції GROUP BY. Результати виконання запиту представлені в табл. 7.1.20.

Таблиця 7.1.20. Результат виконання запиту з прикладу 7.1.17

НГ	count	avg
001	2	4,55

101	4	4,5
102	3	4,42
103	2	4,25
104	3	4,08
105	2	4,50
201	2	3,90
403	2	4,00
TM-31	6	4,54

Концептуально при обробці цього запиту виконуються наступні дії:

1. Рядки таблиці СТУДЕНТ розподіляються в групи відповідно до значень в стовпці номера групи. У межах кожної з груп виявляються дані про весь персонал одного з відділень компанії.
2. Для кожної з груп обчислюється загальна кількість рядків, що дорівнює кількості студентів в групі, а також середнє значення в стовпці прохідний бал. Потім генерується єдиний підсумковий рядок для всієї групи вихідних рядків.
3. Отримані рядки результуючої таблиці сортуються в порядку зростання номера групи, зазначеного в стовпчику НГ.

Стандарт SQL допускає поміщення в список вибірки SELECT вкладених запитів. Тому наведений вище запит можна також представити таким чином:

```
SELECT НГ, (SELECT COUNT (HC) AS count
FROM СТУДЕНТ s
WHERE s.НГ = b.НГ),
(SELECT AVG(salary) AS avg
FROM СТУДЕНТ s
WHERE s.НГ = b.НГ)
FROM СТУДЕНТ b
ORDER BY НГ;
```

Але в цій версії запиту для кожної з груп, описаних в таблиці СТУДЕНТ, створюються два результати обчислення агрегуючих функцій, тому в деяких випадках можлива поява рядків, що містять нульові значення.

Обмеження на виконання групування (конструкція HAVING)

Конструкція HAVING призначена для використання спільно з конструкцією GROUP BY для завдання обмежень, що зазначені з метою відбору тих груп, які будуть поміщені в результуючу таблицю запиту. Хоча конструкції HAVING і WHERE мають подібний синтаксис, їх призначення різне. Конструкція WHERE призначена для відбору окремих рядків, призначених для заповнення результуючої таблиці запиту, а конструкція HAVING використовується для відбору груп, які розміщені в результуючу таблицю запиту. Стандарт ISO вимагає, щоб імена стовпців, що застосовуються в конструкції HAVING, обов'язково були присутні в списку елементів GROUP BY або застосовувалися в агрегуючій функції. На практиці умови пошуку в конструкції HAVING завжди включають, щонайменше, одну агрегуючу функцію; в іншому випадку ці умови пошуку повинні бути

поміщені в конструкцію WHERE і застосовані для відбору окремих рядків. (Пам'ятаєте, що агрегуючі функції не можуть використовуватися в конструкції WHERE.)

Конструкція HAVING не є необхідною частиною мови SQL - будь-який запит, написаний з використанням конструкції HAVING, може бути представлений в іншому вигляді, без її застосування.

Приклад 7.1.18. Використання конструкції HAVING

Для кожної групи з чисельністю студентів понад трьох осіб визначте кількість студентів і їх середній бал.

```
SELECT HG, COUNT (HC) AS count, AVG (salary) AS avg  
FROM СТУДЕНТ  
GROUP BY HG  
HAVING COUNT (HC)> 3  
ORDER BY HG;
```

Цей приклад аналогічний попередньому, але тут використовуються додаткові обмеження, які вказують на те, що нас цікавлять відомості тільки про тих групах, в яких значиться більше трьох осіб. Подібна вимога накладається на групи, тому в запиті слід використовувати конструкцію HAVING. Результати виконання запиту представлені в табл. 7.1.21.

Таблиця 7.1.21. Результат виконання запиту з прикладу 7.1.18

НГ	count	avg
101	4	4,5
ТМ-31	6	4,54

Підзапити

У цьому розділі ми обговоримо використання закінчених операторів SELECT, впроваджених в тіло іншого оператора SELECT. Зовнішній (другий) оператор SELECT використовує результат виконання внутрішнього (першого) оператора для визначення змісту остаточного результату всієї операції. Внутрішні запити можуть перебувати в конструкціях WHERE і HAVING зовнішнього оператора SELECT - в цьому випадку вони отримують назву підзапитів, або вкладених запитів. Крім того, внутрішні оператори SELECT можуть використовуватися в операторах INSERT, UPDATE і DELETE. Існують три типи підзапитів.

Скалярний підзапит повертає значення, вибираємо з перетину одного стовпця з одним рядком, тобто єдине значення. В принципі скалярний підзапит може використовуватися скрізь, де потрібно вказати єдине значення. Варіанти скалярних підзапитів наведені в прикладах 7.1.13 і 7.1.14.

Строковий підзапит повертає значення декількох стовпців таблиці, але у вигляді єдиного рядка. Строковий підзапит може використовуватися скрізь, де застосовується конструктор строкових значень, - зазвичай це предикати. Варіант строкового підзапиту наведено в прикладі 7.1.15

Табличний підзапит повертає значення одного або декількох стовпців таблиці, розміщені в більш ніж одному рядку. Табличний підзапит може використовуватися скрізь, де допускається вказувати таблицю, наприклад як операнд предиката IN.

Приклад 7.1.19. Використання підзапиту з перевіркою на рівність

Складіть список викладачів, які працюють на кафедрі інформатики.

```
SELECT ТАБН, ПІБ, СТ, ЗВ, ККАФ
FROM ВИКЛАДАЧ
WHERE ККАФ = (SELECT ККАФ
               FROM КАФЕДРА
               WHERE НККАФ = 'Інформатики');
```

Внутрішній оператор SELECT (SELECT ККАФ FROM КАФЕДРА ...)

призначений для визначення коду кафедри, що носить назву кафедри 'Інформатики' (Існує тільки одна така назва, тому даний приклад є прикладом скалярного підзапиту.). Після отримання номера необхідного відділення виконується зовнішній підзапит, призначений для вибірки докладних відомостей про викладачів цієї кафедри. Інакше кажучи, внутрішній оператор SELECT повертає таблицю, що складається з одного значення '01'. Він являє собою номер тієї кафедри, яка носить назву кафедри 'Інформатики'. В результаті зовнішній оператор SELECT набуває такого вигляду:

```
SELECT ТАБН, ПІБ, СТ, ЗВ, ККАФ
FROM ВИКЛАДАЧ
WHERE ККАФ = '01';
```

Результати виконання цього запиту представлені в табл. 7.1.22.

Таблиця 7.1.22. Результат виконання запиту з прикладу 7.1.19

Таб. номер	ПІБ викл.	Наук. ступінь	Наук.звання	Код кафедри
101	Андрєєв А.П.	д-р техн. наук	професор	01
102	Апуктін І.С.	канд. техн.наук	доцент	01
103	Глухов І.Л.	канд. техн.наук	доцент	01
104	Сеченов Ю.Б.	канд. техн.наук	доцент	01
105	Чернов Л.К.	канд. техн.наук	доцент	01

Підзапит являє собою інструмент створення тимчасової таблиці, вміст якої витягується і обробляється зовнішнім оператором. Підзапит, можна вказувати безпосередньо після операторів порівняння (тобто операторів =, <, >, <=, >=, <>) в конструкції WHERE або HAVING. Текст під запиту має бути укладений в круглі дужки.

Приклад 7.1.20. Використання підзапитів з агрегуються функціями

Складіть список всіх студентів, які мають бал вище середнього, вказавши, наскільки їх бал перевищує середній бал по інституту.

```
SELECT НГ, НС, ПІБ, РІКНАР
ПБАЛЛ - (SELECT AVG (ПБАЛ) FROM СТУДЕНТ) AS сер_бал
FROM СТУДЕНТ
WHERE ПБАЛ > (SELECT AVG (ПБАЛ) FROM СТУДЕНТ);
```

Необхідно відзначити, що не можна безпосередньо включити в запит вираз 'WHERE ПБАЛ> AVG (ПБАЛ)', оскільки застосовувати агрегуючу функцію в конструкції WHERE заборонено. Для досягнення бажаного результату слід створити підзапит, що обчислює середнє значення середнього прохідного бала, а потім використовувати його в зовнішньому операторі SELECT, призначеному для вибірки відомостей про тих студентів, чий бал це середнє значення. Інакше кажучи, підзапит повертає значення середнього бала в інституті, рівне 4,2. Результат виконання цього скалярного підзапиту використовується в зовнішньому операторі SELECT як для обчислення відхилення бала від середнього рівня, так і для відбору відомостей про працівників. Тому зовнішній оператор SELECT набуває такого вигляду:

```
SELECT НГ, НС, ПІБ, РІКНАР, ПБАЛ - 4,2 AS сер_бал
FROM СТУДЕНТ
WHERE ПБАЛ> 4,2
```

Результати виконання запиту представлені в табл. 7.1.23

Таблиця 7.1.23. Результат виконання запиту з прикладу 7.1.20

Група	№ студента	ПІБ	Рік народження	Прохідний бал
001	01	Васильєв М.В.	1985	5
101	01	Аристов Р.П.	1979	4,5
101	02	Бондаренко С.А.	1978	4,5
101	03	Борисова Є.І.	1979	4,25
101	04	Макова Н.В.	1977	4,75
102	01	Боярська Н.П.	1977	4,5
102	02	Федоров Д.К.	1977	4,25
102	03	Сидоров І.Р.	1977	4,5
103	01	Андрєєв Г.М.	1978	4,25
103	02	Петров О.К.	1979	4,75
104	01	Іванов К.К.	1977	4,5
104	07	Григор'єва О.О.	1979	4,75
105	01	Серов І.Г.	1980	4,5
105	12	Антоненко К. П.	1978	4,5
ТМ-31	07	Глушко І.В.	1986	4,5
ТМ-31	12	Костюшка Т.А.	1986	4,25
ТМ-31	19	Самойленко Ю.А.	1986	4,75
ТМ-31	20	Святненко Ю.В.	1986	4,75
ТМ-31	23	Терпило Д.А.	1986	4,75
ТМ-31	24	Черняк О.Ю.	1986	4,25

До підзапитів застосовуються такі правила і обмеження:

1. У підзапитах не повинна використовуватися конструкція ORDER BY, хоча вона може бути присутньою в зовнішньому операторі SELECT.

2. Список вибірки SELECT підзапиту повинен складатися з імен окремих стовпців або складених з них виразів, за винятком випадку, коли в підзапиті використовується ключове слово EXISTS.

3. За замовчуванням імена стовпців в підзапиті відносяться до таблиці, ім'я якої зазначено в конструкції FROM підзапиту. Однак дозволяється посилатися і на стовпці таблиці, зазначеному в конструкції FROM зовнішнього запиту, для чого використовуються уточнені імена стовпців (як описано нижче).

4. Якщо підзапит є одним з двох операндів, що беруть участь в операції порівняння, то підзапит повинен вказуватися в правій частині цієї операції

Приклад 7.1.21. Вкладені підзапити і використання предиката IN

Складіть перелік груп, які вивчають іноземну мову

SELECT НГ, КІЛ, ПБАЛ

FROM ГРУПА

WHERE НГ IN (SELECT НГ

FROM ВИВЧЕННЯ

WHERE КП = (SELECT КП

FROM ПРЕДМЕТ

WHERE НП = 'Іноземна мова'));

Перший, самий внутрішній, запит призначений для визначення коду предмета 'Іноземна мова'. Другий, проміжний, запит здійснює вибірку відомостей у групах, які вивчають цей предмет. В даному випадку вибирається більше одного рядка даних і тому в зовнішньому запиті можна використовувати оператор порівняння =. Замість нього необхідно використовувати ключове слово IN. Зовнішній запит здійснює вибірку відомостей про групи, які вивчають предмет, код якого був отриманий в результаті виконання проміжного запиту. Результати виконання запиту представлені в табл. 7.1.24

Таблиця 7.1.24. Результат виконання запиту з прикладу 7.1.21

№ групи	К-сть. ст. в групі	Прох.бал
101	15	4,80
102	20	4,00
202	30	4,00

Ключові слова ANY и ALL

Ключові слова ANY і ALL можуть використовуватися з підзапитах, які повертають один стовпець чисел. Якщо підзапиту передуватиме ключове слово ALL, умова порівняння вважається виконаною тільки в тому випадку, якщо вона виконується для всіх значень в результуючому стовпці підзапиту. Якщо тексту підзапиту передує ключове слово ANY, то умова порівняння буде вважатися виконаною, якщо вона задовольняється хоча б для будь-якого (одного або декількох) значення в результуючому стовпці підзапиту. Якщо в результаті виконання підзапиту буде отримано порожнє значення, то для ключового слова ALL умова порівняння

буде вважатися виконаною, а для ключового слова ANY - невиконаною. Відповідно до стандарту ISO додатково можна використовувати ключове слово SOME, що є синонімом ключового слова ANY.

Приклад 7.1.22. Використання ключових слів ANY і SOME

Знайдіть всіх студентів, чий бал перевищує бал хоча б одного студента групи 001

```
SELECT НГ, НС, ПІБ, РІКНАР, АДРЕСА, ПБАЛ
FROM СТУДЕНТ
WHERE ПБАЛ > SOME (SELECT ПБАЛ
                     FROM СТУДЕНТ
                     WHERE НГ = 001);
```

Хоча цей запит може бути записаний з використанням підзапиту, що визначає мінімальний бал студентів групи ТМ-31, після чого зовнішній підзапит зможе вибрати відомості про всіх студентів інституту, чий бал перевершує це значення, можливий і інший підхід, що полягає у використанні ключових слів SOME / ANY . У цьому випадку внутрішній підзапит створює безліч значень {5; 4,1}, а зовнішній запит вибирає відомості про тих працівників, чий бал більше будь-якого з значень в цій множині (фактично більше мінімального значення - 4,1). Подібний альтернативний метод можна вважати більш природним, ніж визначення в підзапиті мінімального бала. Але і в тому і в іншому випадку виробляються однакові результати виконання запиту, які представлені в табл. 7.1.25.

Таблиця 7.1.25. Результат виконання запиту з прикладу 7.1.22

Група	№ студента	ПІБ	Рік народження	АДРЕСА	Прохідний бал
001	01	Васильєв М.В.	1985		5
101	01	Аристов Р.П.	1979		4,5
101	02	Бондаренко С.А.	1978		4,5
101	03	Борисова Є.І.	1979		4,25
101	04	Макова Н.В.	1977		4,75
102	01	Боярська Н.П.	1977		4,5
102	02	Федоров Д.К.	1977		4,25
102	03	Сидоров І.Р.	1977		4,5
103	01	Андрєєв Г.М.	1978		4,25
103	02	Петров О.К.	1979		4,75
104	01	Іванов К.К.	1977		4,5
104	07	Григор'єва О.О.	1979		4,75
105	01	Сєров І.Г.	1980		4,5
105	12	Антоненко К. П.	1978		4,5
ТМ-31	07	Глушков І.В.	1986		4,5
ТМ-31	12	Костюшка Т.А.	1986		4,25

ТМ-31	19	Самойленко Ю.А.	1986		4,75
ТМ-31	20	Святненко Ю.В.	1986		4,75
ТМ-31	23	Терпило Д.А.	1986		4,75
ТМ-31	24	Черняк О.Ю.	1986		4,25

Приклад 7.1.23. Використання ключового слова ALL

Знайдіть всіх студентів, чий бал перевищує бал будь-якого студентата групи

403

```
SELECT НГ, НС, ПІБ, РІКНАР, АДРЕСА, ПБАЛ
FROM СТУДЕНТ
WHERE ПБАЛ > SELECT ПБАЛ
                FROM СТУДЕНТ
                WHERE НГ = 403);
```

В цілому цей запит подібний до попереднього. І в даному випадку можна було б використовувати підзапит, який визначає максимальне значення бала студентів групи під номером 403, після чого за допомогою зовнішнього запиту вибрати відомості про всіх студентів інституту, бал яких перевищує це значення. Однак в даному прикладі вибрано підхід з використанням ключового слова ALL. Результати виконання запиту представлені в табл. 7.1.26.

Таблиця 7.1.26. Результат виконання запиту з прикладу 7.1.23

Група	№ студента	ПІБ	Рік народження	АДРЕСА	Прохідний бал
001	01	Васильєв М.В.	1985		5
001	02	Носков А.К.	1976		4,1
101	01	Аристов Р.П.	1979		4,5
101	02	Бондаренко С.А.	1978		4,5
101	03	Борисова Є.І.	1979		4,25
101	04	Макова Н.В.	1977		4,75
102	01	Боярська Н.П.	1977		4,5
102	02	Федоров Д.К.	1977		4,25
102	03	Сидоров І.Р.	1977		4,5
103	01	Андрєєв Г.М.	1978		4,25
103	02	Петров О.К.	1979		4,75
104	01	Іванов К.К.	1977		4,5
104	07	Григор'єва О.О.	1979		4,75
105	01	Серов І.Г.	1980		4,5
105	12	Антоненко К. П.	1978		4,5
ТМ-31	07	Глушко І.В.	1986		4,5
ТМ-31	12	Костюшка Т.А.	1986		4,25

ТМ-31	19	Самойленко Ю.А.	1986		4,75
ТМ-31	20	Святненко Ю.В.	1986		4,75
ТМ-31	23	Терпило Д.А.	1986		4,75
ТМ-31	24	Черняк О.Ю.	1986		4,25

Багатотабличні запити

Всі розглянуті вище приклади мають одне і те ж важливе обмеження: поміщені в результуючу таблицю стовпці завжди вибираються з єдиної таблиці. Однак у багатьох випадках цього виявляється недостатньо. Для того щоб об'єднати в результуючій таблиці стовпці з декількох вихідних таблиць, необхідно виконати операцію з'єднання. У мові SQL операція з'єднання використовується для об'єднання інформації з двох таблиць за допомогою створення пар пов'язаних рядків, обраних з кожної таблиці. Розміщені в об'єднану таблицю пари рядків складаються по рівності вхідних до них значень зазначених стовпців. Якщо необхідно отримати інформацію з декількох таблиць, то можна або застосувати підзапит, або виконати з'єднання таблиць. Якщо результуюча таблиця запиту повинна містити стовпці з різних вихідних таблиць, то доцільно використовувати механізм з'єднання таблиць. Для виконання з'єднання досить в конструкції FROM вказати імена двох і більше таблиць, розділивши їх комами, після чого включити в запит конструкцію WHERE з визначенням стовпців, які використовуються для з'єднання зазначених таблиць. Крім цього, замість імен таблиць можна використовувати псевдоніми, призначені їм в конструкції FROM. В цьому випадку імена таблиць і призначені їм псевдоніми повинні розділятися пробілами. Псевдоніми можуть використовуватися з метою уточнення імен стовпців у всіх тих випадках, коли можлива неоднозначність щодо того, до якої таблиці відноситься той чи інший стовпець. Крім того, псевдоніми можуть використовуватися для скороченого позначення імен таблиць. Якщо для таблиці визначено псевдонім, він може застосовуватися в будь-якому місці, де виникне потреба у вказівках імені цієї таблиці.

Приклад 7.1.24. просте з'єднання

Складіть список викладачів, які вже поставили хоча б якісь оцінки студентам.

```
SELECT р.ТАБН, ПБ, СТ, ЗВ, ККАФ
FROM ВИКЛАДАЧ р, УСПІШНІСТЬ u
WHERE u.ТАБН = р.ТАБН;
```

У цьому звіті потрібно подати відомості як з таблиці ВИКЛАДАЧ, так і з таблиці УСПІШНІСТЬ, тому при побудові запиту скористаємося механізмом з'єднання таблиць. У конструкції SELECT перераховуються всі стовпці, які повинні бути поміщені в результуючу таблицю запиту. Зверніть увагу, що для стовпця з номером викладача (ТАБН) треба дещо уточнити, оскільки такий стовпець може бути присутнім і в іншій таблиці, яка бере участь в з'єднанні. Тому необхідно явно вказати, значення якої таблиці нас цікавлять. (В даному прикладі з тим же успіхом можна було вибрати значення стовпця ТАБН з таблиці УСПІШНІСТЬ). Уточнення імені здійснюється шляхом вказівки в якості префікса перед ім'ям стовпця імені відповідної

таблиці (або її псевдоніма). В даному прикладі використовується значення 'р', заданий як псевдонім таблиці ВИКЛАДАЧ.

Для формування результуючих рядків використовуються ті рядки вихідних таблиць, які мають ідентичне значення в стовпці ТАБН. Ця умова визначається за допомогою завдання Услов пошуку $u.TABH = p.TABH$. Подібні стовпці вихідних таблиць називають поєднуваними стовпцями. Описана операція еквівалентна операції з'єднання з рівності реляційної алгебри.

Нагадуємо. Операція тета-з'єднання визначає відношення, яке містить, кортежі з декартового добутку відношень R і S , що задовольняють предикату F . Якщо предикат F містить тільки операцію порівняння з рівності ($=$), то з'єднання називається з'єднанням по еквівалентності.

Результати виконання запиту представлені в табл. 7.1.27.

Таблиця 7.1.27. Результат виконання запиту з прикладу 7.1.24

Таб.номер	ПІБ викл.			Наук.ступінь	Наук.звання	Код кафедри
101	Андрєєв А.П.	д-р техн. наук	професор	01		
202	Львова П.Р.		асистент	02		
302	Пустинцев А.П.	канд. ист. наук	доцент	03		
501	Жігарева П.Р.	канд. пед. наук	доцент	05		

Найчастіше багато табличні запити виконуються для двох таблиць, з'єднаних зв'язком типу "один до багатьох" ($1:*$), або батьківсько-дочірнім зв'язком.

Нагадуємо. Зв'язком типу "один до багатьох" називається зв'язок, при якому кожному запису в першій таблиці може відповідати кілька записів у другій, але кожному запису в другій таблиці повинен відповідати тільки один запис в першій. Перша таблиця називається головною або батьківською, а друга - підлеглою або дочірньою.

У наведеному вище прикладі, що включає звернення до таблиць ВИКЛАДАЧ і УСПІШНІСТЬ, останні з'єднані саме таким зв'язком. Кожен рядок таблиці УСПІШНІСТЬ (дочірньої) пов'язана лише з одним рядком таблиці ВИКЛАДАЧ (батьківської), тоді як одна і та ж рядок таблиці (батьківської) може бути пов'язана з багатьма рядками таблиці УСПІШНІСТЬ (дочірньої). Пари рядків, які генеруються при виконанні запиту, являють собою результат всіх допустимих комбінацій рядків дочірньої і батьківської таблиць. Таблиця, що містить зовнішній ключ, зазвичай є дочірньою, тоді як таблиця, що містить первинний ключ, завжди буде батьківською. Для використання батьківсько-дочірнього зв'язку в запиті SQL необхідно вказати умову пошуку, в якому будуть порівнюватися зовнішній і первинний ключі. У прикладі 7.1.24 первинний ключ таблиці ВИКЛАДАЧ (р.ТАБН) порівнюється із зовнішнім ключем таблиці УСПІШНІСТЬ (u.ТАБН).

Стандарт SQL додатково надає наступні способи визначення даного з'єднання:
 FROM ВИКЛАДАЧ р JOIN УСПІШНІСТЬ u ON u.ТАБН = р.ТАБН
 FROM ВИКЛАДАЧ JOIN УСПІШНІСТЬ USING ТАБН
 FROM ВИКЛАДАЧ NATURAL JOIN УСПІШНІСТЬ

У кожному разі конструкція FROM заміщає вихідні конструкції FROM і WHERE. Однак в першому варіанті створюється таблиця з двома ідентичними стовпцями ТАБН, тоді як в інших двох випадках результуюча таблиця буде містити тільки один стовпець ТАБН.

Приклад 7.1.25. Сорткування результатів з'єднання

Для кожної групи перерахуйте номери і імена студентів, які отримали будь-які оцінки, а також вкажіть коди предметів, за якими вони отримали оцінки.

```
SELECT s.НГ, s. НС, ПІБ, КП
FROM СТУДЕНТ s, УСПІШНІСТЬ y
WHERE s. НГ = p. НГ AND s. НС = p. НС
ORDER BY s. НГ, s. НС, КП;
```

Для того щоб результати стали більш зручними для читання, отриманий висновок відсортований з використанням номера групи в якості старшого ключа сорткування, а номери студента і код предмета - як молодших ключів. Результати виконання запиту представлені в табл. 7.1.28.

Таблиця 7.1.28. Результат виконання запиту з прикладу 7.1.25

Номер групИ	Ном. студента	ПІБ	Код предм.
101	01	Аристов Р.П.	01
101	01	Аристов Р.П.	03
101	02	Бондаренко С.А	01
101	02	Бондаренко С.А	03
101	02	Бондаренко С.А	05
101	03	Борисова Є.І.	01
101	03	Борисова Є.І.	03
101	04	Макова Н.В.	01
101	04	Макова Н.В.	03
102	01	Боярська Н.П.	02

Приклад 7.1.26. З'єднання трьох таблиць

Для кожної групи перерахуйте номери і імена студентів, які отримали будь-які оцінки, а також вкажіть код і назви предметів, за якими вони отримали оцінки.

```
SELECT s.НГ, s. НС, ПІБ, y.КП, НП
FROM СТУДЕНТ s, УСПІШНІСТЬ y, ПРЕДМЕТ p
WHERE s. НГ = p. НГ AND s. НС = p. НС AND y.КП = p.КП
ORDER BY s. НГ, s. НС, p.КП;
```

У результуючу таблицю необхідно помістити стовпці з трьох вихідних таблиць - СТУДЕНТ, УСПІШНІСТЬ і ПРЕДМЕТ, тому в запиті слід виконати з'єднання цих таблиць. Таблиці УСПІШНІСТЬ і ПРЕДМЕТ можуть бути з'єднані за допомогою умови y.КП = p.КП, в результаті чого коди отриманих предметів будуть

пов'язані з їх назвами. Таблиці СТУДЕНТ і УСПІШНІСТЬ можуть бути з'єднані за допомогою умови $s. НГ = p. НГ \text{ AND } s. НС = p. НС$ В результаті кожен студент буде пов'язаний з тими предметами, оцінки за якими він отримав. Результати виконання запиту представлені в табл. 7.1.29.

Таблиця 7.1.29. Результати виконання запиту з прикладу 7.1.26

Номер групи	Ном. студента	ПІБ	Код предм.	Назва предмета
101	01	Аристов Р.П.	01	інформатика
101	01	Аристов Р.П.	03	історія
101	02	Бондаренко С.А	01	інформатика
101	02	Бондаренко С.А	03	історія
101	02	Бондаренко С.А	05	філософія
101	03	Борисова Є.І.	01	інформатика
101	03	Борисова Є.І.	03	історія
101	04	Макова Н.В.	01	інформатика
101	04	Макова Н.В.	03	історія
102	01	Боярська Н.П.	02	математика

Приклад 7.1.27. Групування за кількома стовпцями

Визначте кількість оцінок, отриманих кожним із студентів,

```
SELECT s.НГ, s.НС, COUNT (*) AS count
```

```
FROM СТУДЕНТ s, УСПІШНІСТЬ y
```

```
WHERE s.НС = y.НС AND s.НГ = y.Нг
```

```
GROUP BY s.НГ, s.НС
```

```
ORDER BY s.НГ, s.НС;
```

Щоб скласти необхідний звіт, перш за все необхідно з'ясувати, хто зі студентів інституту отримував оцінки. Це завдання можна вирішити за допомогою з'єднання таблиць СТУДЕНТ і УСПІШНІСТЬ за стовпцями НГ і НС в конструкціях FROM / WHERE. Потім необхідно сформулювати групи, що складаються з номера групи і номерів її студентів, для чого слід застосувати конструкцію GROUP BY. Нарешті, результуюча таблиця повинна бути відсортована за допомогою завдання конструкції ORDER BY. Результати виконання запиту представлені в табл. 7.1.30.

Таблиця 7.1.30. Результат виконання запиту з прикладу 7.1.27

Номер групи	Ном. студента	count
101	01	2
101	02	3
101	03	2
101	04	2

102	01	1

Виконання з'єднань

З'єднання є підмножиною більш загальної комбінації даних двох таблиць, званої декартових твором

Нагадуємо. Декартово твір двох таблиць являє собою іншу таблицю, що складається з усіх можливих пар рядків, що входять до складу обох таблиць. Набір стовпців результуючої таблиці являє собою всі стовпці першої таблиці, за якими слідує всі стовпці другої таблиці.

Якщо ввести запит до двох таблиць без завдання конструкції WHERE, результат виконання запиту в середовищі SQL буде являти собою декартовий добуток цих таблиць. Крім того, стандарт ISO передбачає спеціальний формат оператора SELECT, що дозволяє обчислити декартовий добуток двох таблиць:

```
SELECT [DISTINCT | ALL] {*} | columnList
FROM tableName1 CROSS JOIN tableName 2
```

Процедура генерації таблиці, яка містить результати з'єднання двох таблиць за допомогою оператора SELECT, полягає в наступному:

1. Формується декартовий добуток таблиць, зазначених в конструкції FROM.

2. Якщо в запиті присутня конструкція WHERE, застосування умов пошуку до кожного рядка таблиці декартового добутку і збереження в таблиці тільки тих рядків, які задовольняють заданим умовам. У термінах реляційної алгебри ця операція називається обмеженням декартового добутку.

3. Для кожного рядка, що заохився, визначається значення кожного елемента, зазначеного в списку вибірки SELECT, в результаті чого формується окремий рядок результуючої таблиці.

4. Якщо у вихідному запиті присутня конструкція SELECT DISTINCT, з результуючої таблиці видаляються всі рядки-дублікати. В реляційній алгебрі дії, що виконуються на 3 та 4 етапах, еквівалентні операції проєкції за стовпцями, заданим в списку вибірки SELECT.

5. Якщо виконуваний запит містить конструкцію ORDER BY, здійснюється переупорядкування рядків результуючої таблиці.

Зовнішні з'єднання

При виконанні операції з'єднання дані з двох таблиць комбінуються з утворенням пар пов'язаних рядків, в яких значення зіставляваних стовпців є однаковими. Якщо одне із значень в зіставл.ваному стовпці однієї таблиці не збігається ні з одним зі значень в зіставл.ваному стовпці іншої таблиці, то відповідний рядок видаляється з результуючої таблиці. Саме це правило застосовувалося у всіх розглянутих вище прикладах з'єднання таблиць. Стандартом ISO передбачено й інший набір операторів сполук, які називаються зовнішніми з'єднаннями.

Нагадуємо. Іноді може знадобитися, щоб рядок з одного відношення був представлений в результатах з'єднання, навіть якщо в іншому відношенні немає значення, що збігається. Ця мета може бути досягнута за допомогою зовнішнього з'єднання.

У зовнішньому з'єднанні в результуючу таблицю поміщаються також рядки, що не задовольняють умові з'єднання. Щоб зрозуміти особливості виконання операцій зовнішнього з'єднання, скористаємося спрощеними таблицями Кафедра1 і Предмет1, вміст яких представлено в табл. 7.1.31 і 7.1.32.

Таблиця 7.1.31. Таблиця Кафедра 1

Код кафедри (ККАФ)	Назв. кафедри (НКАФ)
01	Інформатика
02	Математика
03	Історія
07	Захист інформації

Таблиця 7.1.32. Таблиця Предмет 1

Код предмета (КП)	Назв. предмета (НП)
01	Інформатика
02	Математика
03	Історія
08	Право

Звичайне (внутрішнє) поєднання цих таблиць виконується за допомогою наступного оператора SQL:

```
SELECT b.*, p.*
FROM Кафедра1 b, Предмет1 p
WHERE b.НКАФ = p.НП
```

Результати виконання цього запиту представлені в табл. 7.1.33.

Таблиця 7.1.33. Результат у внутрішньому сполученні спрощених таблиць

код кафедри	Назв. кафедри (НКАФ)	Код предмета (КП)	Назв. Предмета (НП)
(ККАФ)	Інформатика	01	Інформатика
01	Математика	02	Математика
02	Історія	03	Історія

Як можна бачити, в результуючій таблиці запиту є тільки три рядки, що містять однакові назви предметів, вибрані з обох таблиць. Зверніть увагу, що у вихідних даних немає відповідності для кафедри Захисту інформації та для предмета Право. Якщо в результуючу таблицю потрібно включити і ці не мають відповідності рядка, то слід використовувати зовнішнє з'єднання. Існують три типи зовнішнього з'єднання: ліве, праве і повне. Розглянемо особливості кожного з них на наведених нижче прикладах.

Приклад 7.1.28. Ліве зовнішнє з'єднання

Перерахуйте всі кафедри для яких є відповідні предмети, а також кафедри, що не задовольняють умові запиту.

Використовуємо ліве зовнішнє з'єднання цих двох таблиць, яке виглядає наступним чином: SELECT b.*, p.*

FROM Кафедра1 b LEFT JOIN Предмет1 p ON b.НКАФ = p.НП;

Результати виконання цього запиту представлені в табл. 7.1.34. У цьому прикладі за рахунок застосування лівого зовнішнього з'єднання в результуючу таблицю потрапили не тільки три рядки, в яких є відповідність між назвами предметів, але також той рядок першої з з'єднуються таблиць (лівої), яка не знайшла собі відповідності в другій таблиці (правої). У цьому рядку всі поля другої таблиці заповнені значеннями NULL.

Таблиця 7.1.34. Результат виконання запиту з прикладу 7.1.28

Код предмета (КП)	Назв. предмета (НП)	Код кафедри (ККАФ)	Назв. кафедри(НКАФ)
01	Інформатика	01	Інформатика
02	Математика	02	Математика
03	Історія	03	Історія
07	Захист інформації	NULL	NULL

Приклад 7.1.29. Праве зовнішнє з'єднання

Перерахуйте всі предмети для яких є відповідні кафедри, а також всі інші предмети, що не задовольняють умові запиту.

Використовуємо праве зовнішнє з'єднання цих двох таблиць, яке виглядає наступним чином:

SELECT b. *, p. *

FROM Кафедра1 b RIGHT JOIN Предмет1 p ON b.НКАФ = p.НП;

Результати виконання цього запиту представлені в табл. 7.1.35. У цьому прикладі при виконанні правого зовнішнього з'єднання в результуючу таблицю були включені не тільки ті три рядки, які мають однакові значення в зіставляються шпальтах з кафедри, але також і ті рядки з другої (правої) таблиці, які не знайшли відповідності з рядками в першій (лівої) таблиці. У цьому рядку всі поля з першої таблиці отримали значення NULL.

Таблиця 7.1.35. Результат виконання запиту з прикладу 7.1.29

Код предмета (КП)	Назв. предмета (НП)	Код кафедри (ККАФ)	Назв. кафедри (НКАФ)
01	Інформатика	01	Інформатика
02	Математика	02	Математика
03	Історія	03	Історія
NULL	NULL	08	Право

Приклад 7.1.30. Повне зовнішнє з'єднання

Перерахуйте всі предмети для яких є відповідні кафедри, а також всі інші предмети і кафедри, що не задовольняють умові запиту.

Використовуємо повне зовнішнє з'єднання цих таблиць, яке виглядає наступним чином:

```
SELECT b. *, p. *
```

```
FROM Кафедра1 b FULL JOIN Предмет1 p ON b.НКАФ = p.НП;
```

Результати виконання цього запиту представлені в табл. 7.1.36. У разі повного зовнішнього з'єднання в результуючу таблицю поміщаються не тільки ті два рядки, які мають однакові значення в зіставляються шпальтах з назвою міста, а й всі інші рядки вихідних таблиць, що не знайшли собі відповідності. У цих рядках всі стовпці тій таблиці, в якій не було знайдено відповідності, заповнюються значеннями NULL.

Таблиця 7.1.36. Результат виконання запиту з прикладу 7.1.30

Код предмета (КП)	Назв. предмета (НП)	Код кафедри (ККАФ)	Назв. Кафедри (НКАФ)
01	Інформатика	01	Інформатика
02	Математика	02	Математика
03	Історія	03	Історія
NULL	NULL	08	Право
07	Захист інформації	NULL	NULL

Ключові слова EXISTS і NOT EXISTS

Ключові слова EXISTS і NOT EXISTS призначені для використання тільки спільно з підзапитами. Результат їх обробки є логічне значення TRUE або FALSE. Для ключового слова EXISTS результат дорівнює TRUE в тому і тільки в тому випадку, якщо повернутій підзапитом результуючій таблиці присутній хоча б один рядок. Якщо результуюча таблиця підзапиту порожня, результатом обробки ключового слова EXISTS буде значення FALSE. Для ключового слова NOT EXISTS використовуються правила обробки, зворотні по відношенню до ключового слова EXISTS. Оскільки за ключовими словами EXISTS і NOT EXISTS перевіряється лише наявність рядків в результуючій таблиці підзапиту, то ця таблиця може містити довільну кількість стовпців. Як правило, з метою спрощення у всіх наступних за обговорюваними ключовими словами підзапитах застосовується така форма запису:

```
(SELECT * FROM ...)
```

Приклад 7.1.31. Запит з використанням ключового слова EXISTS

Перерахуйте всіх студентів групи, де вивчають електродинаміку (код 10).

```
SELECT ИС, ПІБ, РІКНАР
```

```
FROM СТУДЕНТ s
```

```
WHERE EXISTS (SELECT *
```

```
FROM ВІВЧЕННЯ i
```

```
WHERE s. НГ = i.НГ AND КП = 10);
```

Цей запит можна перефразувати наступним чином: "Вибрати відомості про всіх студентів, для яких у таблиці ВІВЧЕННЯ існує запис, що містить той же номер групи НГ, який вказаний для даного студента, і в якій значення стовпця КП дорівнює 10".

Перевірка необхідності включення відомостей про студента в результуючу таблицю полягає в аналізі існування подібного запису. Якщо вона існує, результат обробки ключового слова EXISTS буде дорівнює TRUE. Результати виконання запиту представлені в табл. 7.1.37.

Таблиця 7.1.37. Результат виконання запиту з прикладу 7.1.31

№ студента	ПІБ	Рік народження
07	Глушко І.В.	1986
12	Костюшка Т.А.	1986
19	Самойленко Ю.А.	1986
20	Святненко Ю.В.	1986
23	Терпило Д.А.	1986
24	Черняк О.Ю.	1986

Комбінування результуючих таблиць (операції UNION, INTERSECT і EXCEPT)

У мові SQL можна використовувати звичайні операції над множинами - об'єднання (union), перетин (intersection) і різницю (difference), - що дозволяють комбінувати результати виконання двох і більше запитів в єдину результуючу таблицю.

Об'єднанням двох таблиць *A* і *B* називається таблиця, яка містить всі рядки, які є в першій таблиці (*A*), у другій таблиці (*B*) або в обох цих таблицях одночасно.

Перетином двох таблиць називається таблиця, яка містить всі рядки, присутні в обох вихідних таблицях одночасно.

Різницею двох таблиць *A* і *B* називається таблиця, яка містить всі рядки, які присутні в таблиці *A*, але відсутні в таблиці *B*.

Всі ці операції над множинами графічно представлені на рис. 7.1.2. На таблиці, які можуть комбінуватися за допомогою операцій над множинами, накладаються певні обмеження. Найважливіше з них полягає в тому, що таблиці повинні бути сумісні, по з'єднанню - тобто вони повинні мати одну і ту ж структуру. Це означає, що таблиці повинні мати однакову кількість стовпців, причому у стовпцях повинні розміщуватися дані одного і того ж типу і довжини. Обов'язок переконатися в тому, що значення даних відповідних стовпців належать одному і тому ж домену, покладається на користувача. Наприклад, мало сенсу в тому, щоб об'єднувати стовпець з даними про вік працівників з інформацією про кількість кімнат, які здаються в оренду, хоча обидва стовпці матимуть один і той же тип даних – SMALLINT.

Три операції над множинами, передбачені стандартом ISO, зветься UNION, INTERSECT і EXCEPT. У кожному разі формат конструкції з операцією над множинами повинен бути наступним:

operator [ALL] {CORRESPONDING | BY {column1 [, . . .]}}

При вказівці конструкції CORRESPONDING BY операція над множинами виконується для зазначених стовпців. Якщо задано тільки ключове слово CORRESPONDING, а конструкція 3Y відсутня, операція над множинами виконується

для стовпців, які є загальними для обох таблиць. Якщо вказано ключове слово ALL, результуюча таблиця може містити повторювані рядки.

Одні діалекти мови SQL не підтримують операцій INTERSECT і EXCEPT, а в інших замість ключового слова EXCEPT використовується ключове слово MINUS.

Приклад 7.1.32. Використання операції UNION

Створіть список всіх предметів і кафедр

```
(SELECT НП  
FROM ПРЕДМЕТ  
WHERE НП IS NOT NULL)  
UNION  
(SELECT НКАФ  
FROM КАФЕДРА  
WHERE НКАФ IS NOT NULL);
```

Цей запит виконується за допомогою підготовки результуючої таблиці першого запиту і результуючої таблиці другого запиту з подальшим злиттям обох таблиць в єдину результуючу таблицю, яка включає всі рядки з обох проміжних таблиць, але з видаленням повторюваних рядків. Остаточний результат виконання запиту представлений в табл. 7.1.38.

Таблиця 7.1.38. Результат виконання запиту з прикладу 7.1.32

Інформатика
Математика
Історія
Ін. мова
Філософія
Фізкультура
програмування
право
фізика
електродинаміка
Психологія
Захист інформації
ТСМ

Приклад 7.1.33. Використання операції INTERSECT

Створіть список всіх груп, в яких вивчаються предмети і є оцінки

```
(SELECT НГ  
FROM ВИВЧЕННЯ)  
INTERSECT  
(SELECT НГ  
FROM УСПІШНІСТЬ);
```

Цей запит виконується за допомогою підготовки результуючої таблиці першого запиту і результуючої таблиці другого запиту з подальшим створенням єдиної результуючої таблиці, що включає тільки ті рядки, які є загальними для обох

проміжних таблиць. Остаточний результат виконання запиту представлений в табл. 7.1.39.

Таблиця 7.1.39. Результат виконання запиту з прикладу 7.1.33

НГ
101
102

Цей запит можна записати і без використання операції INTERSECT.

Одним з найістотніших недоліків мови SQL є те, що вона дозволяє створювати запити в декількох еквівалентних формах.

Приклад 7.1.34. Використання операції EXCEPT

Створіть список всіх груп, де вивчаються предмети, але немає оцінок

(SELECT НГ
FROM ВИВЧЕННЯ)

EXCEPT

(SELECT НГ
FROM УСПІШНІСТЬ);

Цей запит виконується за допомогою підготовки результуючої таблиці першого запиту і результуючої таблиці другого запиту з подальшим створенням єдиної результуючої таблиці, що включає тільки ті рядки, які є в першій проміжній таблиці, але відсутні в другій. Остаточний результат виконання запиту представлений в табл. 7.1.40.

Таблиця 7.1.40. Результат виконання запиту з прикладу 7.1.34

НГ
105
201
202
203
204
403
ТМ-31

Цей запит можна записати і без використання операції EXCEPT.

Зміна вмісту бази даних

SQL є повнофункціональною мовою маніпулювання даними, яка може використовуватися не тільки для вибірки даних з бази, але і для модифікації її вмісту. Оператори модифікації інформації в базі даних не настільки складні, як

оператор SELECT. У цьому розділі розглядаються три оператора мови SQL, призначених для модифікації вмісту бази даних.

- **INSERT** - призначений для додавання даних в таблицю,
- **UPDATE** - призначений для модифікації вже поміщених в таблицю даних.
- **DELETE** - дозволяє видаляти з таблиці рядки даних.

Додавання нових даних в таблицю (оператор INSERT)

Існують дві форми оператора INSERT. Перша призначена для вставки єдиного рядка в зазначену таблицю. Ця форма оператора INSERT має наступний формат:

```
INSERT INTO TableName [(ColumnList)]  
VALUES (dataValueList)
```

Тут параметр *TableName* (Ім'я таблиці) може становити або ім'я таблиці бази даних, або ім'я оновлюваного представлення. Параметр *columnList* (Список стовпців) являє собою список, що складається з імен одного або більше стовпців, розділених комами. Параметр *columnList* є необов'язковим. Якщо він опущений, то передбачається використання списку з імен всіх стовпців таблиці, зазначених в тому порядку, в якому вони були описані в операторі CREATE TABLE. Якщо в операторі INSERT вказується конкретний список імен стовпців, то будь-які опущені в ньому стовпці повинні бути оголошені при створенні таблиці як допускають значення NULL - за винятком випадків, коли при описі стовпчика використовувався параметр DEFAULT. Параметр *dataValueList* (Список значень даних) повинен таким чином відповідати параметру *columnList*:

- кількість елементів в обох списках має бути однаковою;
- має існувати пряма відповідність між позицією одного і того ж елемента в обох списках, тому перший елемент списку *dataValueList* вважається таким, що до першого елементу списку *columnList*, другий елемент списку *dataValueList* - до другого елементу списку *columnList* і т.д.;
- типи даних елементів списку *dataValueList* повинні бути сумісні з типом даних відповідних стовпців таблиці.

Приклад 7.1.35. Використання конструкції INSERT ... VALUES

Помістіть в таблицю СТУДЕНТ новий запис, що містить дані у всіх стовпцях.

```
INSERT INTO СТУДЕНТ  
VALUES (104, 08, 'Петровський', 1987, 'Київ', 4,8);
```

Якщо розміщувати вставляються в кожен стовпець таблиці дані в порядку їх розташування в таблиці, то вказувати список стовпців необов'язково. Пам'ятайте, що символні значення повинні бути укладені в одинарні лапки.

Приклад 7.1.36. Вставка нового запису з використанням значень, прийнятих за замовчуванням

Помістіть в таблицю СТУДЕНТ новий запис, що містить дані у всіх шпальтах НГ, НС, ПІБ, РІКНАР, АДРЕСА, ПБАЛ

```
INSERT INTO СТУДЕНТ (НГ, НС, ПІБ, РІКНАР, АДРЕСА, ПБАЛ)  
VALUES { 'ТМ-31', 26, 'Чортів', 1986, 'Саратов', 4};
```

Оскільки дані вставляються тільки в певні стовпці таблиці, необхідно вказати імена стовпців, в які ці дані будуть поміщатися. Порядок проходження імен стовпців

несуттєвий, однак більш природно вказати їх в тому порядку, в якому вони слідуєть у таблиці. Крім того, оператор INSERT можна було б записати і таким чином:

```
INSERT INTO СТУДЕНТ (НГ, НС, ПІБ, РІКНАР, АДРЕСА, ПБАЛ)  
VALUES {'TM-31', 26, Чортів ', 1986, NULL, 4} ;
```

В цьому випадку ми явно вказали, що в стовпець адреса має бути вміщено значення NULL.

Друга форма оператора INSERT дозволяє скопіювати безліч рядків однієї таблиці в іншу. Цей оператор має наступний формат:

```
INSERT INTO TableName [(ColumnList)]
```

```
SELECT ...
```

Тут параметри TableName і columnList мають той же формат і зміст, що і при вставці в таблицю одного рядка. Конструкція SELECT може являти собою будь-який допустимий оператор SELECT. Рядки, що вставляються в зазначену таблицю, в точності відповідають рядкам результуючої таблиці, створеної при виконанні вкладеного запиту. Всі обмеження, зазначені вище для першої форми оператора INSERT, застосовні і в цьому випадку.

Відзначимо, що в деяких діалектах мови SQL не допускається використовувати оператор UNION в підзапитах оператора INSERT.

Модифікація даних в базі (оператор UPDATE)

Оператор UPDATE дозволяє змінювати вміст уже існуючих рядків зазначеної таблиці. Цей оператор має наступний формат:

```
UPDATE TableName  
SET columnName1 = dataValue [columnName2 = dataValue2 ...]  
[WHERE searchCondition]
```

Тут параметр TableName представляє або ім'я таблиці бази даних, або ім'я оновлюваного представлення. У конструкції SET вказуються імена одного або більше стовпців, дані в яких необхідно змінити. Конструкція WHERE є необов'язковою. Якщо вона опущена, значення зазначених стовпців будуть змінені у всіх рядках таблиці. Якщо конструкція WHERE присутня, то оновлені будуть тільки ті рядки, які задовольняють умові пошуку, заданій в параметрі searchCondition. Параметри dataValue1, dataValue2 ... представляють нові значення відповідних стовпців і повинні бути сумісні з ними за типом даних.

Приклад 7.1.37. Оновлення всіх рядків таблиці за допомогою оператора UPDATE

Збільшити вивчення всіх предметів на 10%

```
UPDATE ПРЕДМЕТ  
SET ГОДИНИ = ГОДИНИ* 1.1;
```

Оскільки зміни стосуються всіх рядків таблиці ПРЕДМЕТ, конструкцію WHERE вказувати не потрібно.

Приклад 7.1.38. Оновлення деяких рядків таблиці за допомогою оператора UPDATE

Збільшити вивчення всіх предметів, вивчаються 4 семестри на 5%

UPDATE ПРЕДМЕТ

SET ГОДИННИК = ГОДИННИК * 1.05;

WHERE НС = 4;

Тут конструкція WHERE застосовується для поновлення тільки тих рядків таблиці, які містять відомості про предмети, вивчаються 4 семестри. Саме в цих рядках в стовпець ГОДИНИ буде поміщено нове значення, що обчислюється як $ГОДИНИ = ГОДИНИ * 1.05$

Приклад 7.1.39. Оновлення декількох стовпців за допомогою оператора UPDATE

Перекласти Арістова Р.П. (НС = 01, НГ = 101) в групу 102 і підвищити його бал на 0,2

UPDATE СТУДЕНТ

SET НГ = 102, НС=05, ПБАЛ=ПБАЛ+0,2

WHERE НГ=101, НС=01;

Видалення даних з бази (оператор DELETE)

Оператор DELETE дозволяє видаляти рядки даних з зазначеної таблиці. Цей оператор має наступний формат:

DELETE FROM TableName

[WHERE searchCondition]

Як і в випадку операторів INSERT і UPDATE, параметр TableName може являти собою або ім'я таблиці бази даних, або ім'я оновлюваного представлення.

Нагадуємо. Представлення - динамічно сформований результат однієї або декількох реляційних операцій, виконаних над відносинами бази даних з метою отримання нового відносини. Вистава є віртуальним ставленням, яке не завжди реально існує в базі даних, але створюється за запитом певного користувача в ході виконання цього запиту.

Параметр searchCondition є обов'язковим - якщо він опущений, з таблиці будуть видалені всі існуючі в ній рядки. Однак сама по собі таблиця видалена не буде. Якщо необхідно видалити не тільки вміст таблиці, але і її визначення, слід використовувати оператор DROP TABLE. Якщо конструкція WHERE присутній, з таблиці будуть видалені тільки ті рядки, які задовольняють умові відбору, заданому параметром searchCondition.

Приклад 7.1.40. Видалення певних рядків таблиці (оператор DELETE)

Видалити всі записи про оцінки «5».

DELETE FROM УСПІШНІСТЬ

WHERE ОЦІНКА = 5;

Конструкція WHERE дозволяє знайти як призначені для видалення тільки ті рядки таблиці, які відносяться до здає в оренду об'єкту з номером 'PG4', і застосувати до них операцію видалення.

Приклад 7.1.41. Видалення всіх рядків таблиці (оператор DELETE)

Видалити всі рядки з таблиці УСПІШНІСТЬ

DELETE FROM УСПІШНІСТЬ

Оскільки в даному операторі конструкція WHERE не вказана, будуть видалені всі рядки таблиці. В результаті в базі даних збережеться лише опис таблиці УСПІШНІСТЬ, що в подальшому дозволить ввести в неї нову інформацію.

Питання:

- 7.1.1. Назвіть два головні компоненти мови SQL. Які функції вони виконують?
- 7.1.2. Перерахуйте переваги і недоліки мови SQL.
- 7.1.3. Поясніть призначення кожної з конструкцій, які можуть бути присутнім в операторі SELECT. Які обмеження накладаються на ці конструкції?
- 7.1.4. Поясніть принципи роботи конструкцій WHERE, ORDER BY, FROM.
- 7.1.5. Які обмеження накладаються на використання агрегуючих функцій в тілі оператора SELECT? Як агрегуючими функціями обробляються значення NULL?
- 7.1.6. Сформулюйте принципи роботи конструкції GROUP BY. У чому полягає відмінність між конструкціями WHERE і HAVING?
- 7.1.7. Які є типи підзапитів?
- 7.1.8. Визначте відмінності між підзапитом і з'єднанням? при будь обставинах використання підзапитів стає неможливим?
- 7.1.9. Поясніть призначення операторів INSERT, UPDATE, DELETE.
- 7.1.10. Доведіть, що запит, побудований з використанням конструкції HAVING, завжди може бути переписаний в еквівалентному формулюванні без використання конструкції HAVING.
- 7.1.11. Поясніть, чому мова SQL є реляційно-повною?

7.2 Мова SQL: Визначення даних

Типи даних мови SQL, визначені стандартом ISO

Ідентифікатори мови SQL

Ідентифікатори мови SQL призначені для позначення об'єктів в базі даних і є іменами таблиць, представлень і стовпців. Символи, які можуть використовуватися в створюваних користувачем ідентифікаторах мови SQL, повинні бути визначені як набір символів. Стандарт ISO задає набір символів, який повинен використовуватися за замовчуванням; він включає великі та малі літери латинського алфавіту (A-Z, a-z), цифри (0-9) і символ підкреслення (_). Допускається використання і альтернативного набору символів.

На формат ідентифікаторів накладаються наступні обмеження:

- може мати довжину до 128 символів (більшість діалектів передбачає більш жорсткі обмеження);
- повинен починатися з літери;
- Без пробілів.

Скалярні типи даних мови SQL

У табл. 7.2.1 перераховані скалярні типи даних мови SQL, які визначені стандартом ISO. У деяких випадках з метою спрощення маніпулювання і перетворення, а також через схожість основних властивостей дані типів character і bit об'єднуються під назвою "рядкові типи даних", а дані типів exact numeric і approximate numeric - під назвою "числові типи даних".

Таблиця 7.2.1. Типи даних мови SQL, визначені в стандарті ISO

Тип даних	Оголошення		
boolean (Логічний)	BOOLEAN		
character (Символьний)	CHAR	VARCHAR	
bit (Бітовий)	BIT	BIT VARYING	
exact numeric (Точні числа)	NUMERIC SMALLINT	DECIMAL	INTEGER
approximate numeric (Округлені числа)	FLOAT	REAL	DOUBLE PRECISION
datetime (Дата/час)	DATE	TIME	TIMESTAMP
interval (Інтервал)	INTERVAL		

LOB (Великий об'єкт)	CHARACTER LARGE OBJECT	BINARY LARGE OBJECT

Логічні дані (тип **boolean**)

Логічні дані складаються з помітних істинних значень **TRUE** (істинний) і **FALSE** (помилковий). Логічні дані підтримують також істинне значення **UNKNOWN** (невизначений), заданий як значення **NULL**, якщо застосування невизначених значень не заборонено обмеженням **NOT NULL**. Всі значення даних логічного типу та істинності значення SQL можуть спільно застосовуватися в операторах порівняння і присвоювання. Значення **TRUE** в арифметичних операторах порівняння більше значення **FALSE**, а будь-яке порівняння, в якому бере участь значення **NULL** або істинностное значення **UNKNOWN**, повертає результат **UNKNOWN**.

Символьні дані (тип **character**)

Символьні дані складаються з послідовностей символів, що входять в певний творцями СУБД набір символів. Оскільки набори символів є специфічними для різних діалектів мови SQL, перелік символів, які можуть входити до складу значень даних символьного типу, так само залежить від конкретної реалізації. Для визначення даних символьного типу застосовується наступний формат:

- **CHARACTER [VARYING]**
- **CHARACTER** (може бути скорочено до **CHAR**)
- **CHARACTER VARYING** (може бути скорочено до **VARCHAR**)

При визначенні стовпця з символьним типом даних параметр **length** використовується для вказівки максимальної кількості символів, які можуть бути поміщені в даний стовпець (за замовчуванням приймається значення 1). Символьний рядок може бути визначений як той, що має фіксовану або змінну (**VARYING**) довжину. Якщо рядок визначений з фіксованою довжиною, то при введенні в неї меншої кількості символів строкове значення доповнюється до зазначеної довжини пробілами, що додаються справа. Якщо рядок визначений зі змінною довжиною, то при введенні в неї меншої кількості символів в базі даних будуть збережені тільки введені жодного символу, що дозволяє досягти певної економії зовнішньої пам'яті.

Всі приклади будуть ґрунтуватися на базі «Навчальний процес», до якої є наступні таблиці (заповнені таблиці наведені в додатку):

СТУДЕНТ (НГ, НС, ПІБ, РІКНАР, АДРЕСА, ПБАЛ)

ГРУПА (НГ, К-сть, ПБАЛЛ)

КАФЕДРА (ККАФ, НКАФ, ТЕЛ, ЗАВ, ФОТО)

ВИКЛАДАЧ (ТАБН, ПІБ, СТ, ЗВ, ККАФ)

ПРЕДМЕТ (КП, НП, ГОДИНИ, ЛЕК, ПР, НС, ПРОГР)

ВИВЧЕННЯ (НГ, КП, ТАБН, Видз, ГОДИНИ, СБАЛГР)

УСПІШНІСТЬ (НГ, НС, КП, ТАБН, Видз, ОЦІНКА)

Наприклад, стовпець ТАБН таблиці Викладач з позначенням номера викладача має фіксовану довжину чотири символи і може бути оголошений таким чином:

ТАБН CHAR (4)

Бітові дані (тип bit)

Бітовий тип даних використовується для визначення бітових рядків, тобто послідовності двійкових цифр (бітів), кожна з яких може мати значення або 0, або 1. Для визначення даних бітового типу використовується формат, схожий з визначенням символічних даних:

BIT [VARYING] {length}.

Наприклад, для збереження бітового рядка з фіксованою довжиною і значенням 'Вася' може бути оголошений стовпець bitstring:

bitString BIT (4)

Точні числові дані (тип exact numeric)

Тип точних числових даних використовується для визначення чисел, які мають точне представлення в комп'ютері. Числа складаються з цифр і необов'язкових символів (десятькового дробу, знака "плюс" або "мінус"). Дані точного числового типу визначаються значністю (precision) і довжиною дробової частини (scale). Значність задає загальну кількість значущих десятикових цифр числа, до якого входять довжина цілої та дробової частин, але без урахування самого десятикового дробу. Дробова частина вказує кількість дробових десятикових розрядів числа. Наприклад, точне число -12.345 має значність, рівну 5 цифр, і

дробову частину довжиною 3. Особливим різновидом точних чисел є цілі числа. Існує кілька способів визначення даних точного числового типу:

NUMERIC (precision [scale])

DECIMAL (precision [scale]) (може бути скорочено до DEC)

INTEGER (може бути скорочено до INT)

SMALLINT

Типи **NUMERIC** і **DECIMAL** призначені для зберігання чисел в десятковому форматі. За замовчуванням довжина дробової частини дорівнює нулю, а прийнята за замовчуванням значність залежить від реалізації. Тип **INTEGER** використовується для зберігання великих позитивних або негативних цілих чисел. Тип **SMALLINT** використовується для зберігання невеликих позитивних або негативних цілих чисел. При використанні цього типу даних витрата зовнішньої пам'яті істотно скорочується. Наприклад, максимальне абсолютне значення числа, яке може зберігатися в шпальтах з типом даних **SMALLINT**, найчастіше становить 32 767. Для стовпця **HC** таблиці **Предмет**, в якому зберігаються відомості про кількість семестрів, можна вибрати тип **SMALLINT** і оголосити його в такий спосіб:

HC SMALLINT

Округлені числові дані (тип approximate numeric)

Тип округлених числових даних використовується для опису даних, які не можна точно представити комп'ютерними програмами, наприклад, дійсних чисел. Для представлення округлених чисел або чисел з плаваючою точкою використовується Експоненціальна система позначень, в якій число записується за допомогою мантиси, помноженої на певний ступінь десяти (порядок). Існує кілька способів визначення даних з типом округлених числових даних:

FLOAT [precision]

REAL

DOUBLE PRECISION

Параметр **precision** задає значність мантиси. Значність визначень типу **REAL** і **DOUBLE PRECISION** залежить від конкретної реалізації.

Дата і час (тип datetime)

Тип даних "дата / час" використовується для визначення моментів часу з деякою установленою точністю. Прикладами є дати, позначки часу і час доби. Стандарт ISO розділяє тип даних "дата / час" на підтипи **YEAR** (Рік), **MONTH** (Місяць), **DAY** (День), **HOURL** (Година), **MINUTE** (Хвилина), **SECOND** (Секунда)²⁹⁸,

TIMEZONE_HOUR (Зональний годину) і **TIMEZONE_MINUTE** (зональна хвилина). Два останніх типи визначають час і хвилини зсуву зонального часу по відношенню до загального скоординованого часу. Підтримуються три типи полів дати / часу.

DATE

TIME [timePrecision] [WITH TIME ZONE]

TIMESTAMP [timePrecision] [WITH TIME ZONE]

Тип даних **DATE** використовується для зберігання календарних дат, які включають поля **YEAR**, **MONTH** і **DAY**. Тип даних **TIME** використовується для зберігання відміток часу, що включають поля **HOURL**, **MINUTE** і **SECOND**. Тип даних **TIMESTAMP** служить для спільного зберігання дати та часу. Параметр **timePrecision** задає кількість дрібних десяткових знаків, що визначають точність представлення значень у полі **SECOND**. Якщо цей параметр опущений, за замовчуванням його значення для стовпців типу **TIME** приймається рівним нулю (тобто зберігається ціле кількість секунд), тоді як для полів типу **TIMESTAMP** він приймається рівним 6 (тобто позначки часу зберігаються з точністю до мікросекунд) . Наявність ключового слова **WITH TIME ZONE** визначає використання полів **TIMEZONE_HOUR** і **TIMEZONE_MINUTE**.

Інтервальний тип даних interval

Дані з інтервальним типом використовуються для подання періодів часу. Будь-інтервальний тип даних складається з набору полів: **YEAR**, **MONTH**, **DAY**, **HOURL**, **MINUTE** і **SECOND**. Існують два класи даних з інтервальним типом: інтервали рік-місяць і інтервали доба-час доби. У першому випадку дані

включають тільки два поля - YEAR і / або MONTH. Дані другого типу можуть складатися з довільної послідовності полів DAY, HOUR, MINUTE, SECOND.

Дані інтервального типу визначаються наступним чином:

INTERVAL {{startField To endField} singleDatetimeField}

StartField = YEAR MONTH | DAY j HOUR | MINUTE

[(IntervalLeadingFieldPrecision)]

endField = YEAR | MONTH | DAY j HOUR | MINUTE | SECOND

[(FractionalSecondsPrecision)]

singleDatetimeField = startField |; SECONB

[(IntervalLeadingFieldPrecision

[, FractionalSecondsPrecision]])]

Для параметра startField повинна бути завжди вказана розмірність першого поля (intervalLeadingFieldPrecision), яка за замовчуванням приймається рівною двом. наприклад:

INTERVAL YEAR (2) TO MONTH

Це оголошення описує інтервал часу, значення якого може знаходитися між 0 роком, 0 місяцем і 99 роком, 11 місяцем. Ще один приклад:

INTERVAL HOUR TO SECOND (4)

Це оголошення описує інтервал часу, значення якого може змінюватися від 0 годин, 0 хвилин, про секунд до 99 годин, 59 хвилин 59.9999 секунди. (Число дрібних десяткових знаків для секунд встановлено рівним 4.)

Скалярні оператори

Мова SQL включає кілька вбудованих скалярних операторів і функцій, які можуть використовуватися для побудови скалярних виразів, тобто виразів, обчислення яких дає скалярний результат. Крім звичайних арифметичних операторів (+, -, * і /) в мові визначені і інші оператори, представлені в табл. 7.2.2.

Таблиця 7.2.2. Скалярні оператори мови SQL, передбачені стандартом ISO

Оператор	Призначення
BIT_LENGTH	Повертає довжину заданого рядка в бітах. Наприклад, результат обчислення виразу BIT_LENGTH (X'FFFF ') дорівнює 16

OCTET_LENGTH	Повертає довжину заданого рядка в октетах (довжина в бітах, поділена на 8). Наприклад, результат обчислення виразу OCTET_LENGTH (X'FFFF ') дорівнює 2
CHAR_LENGTH	Повертає довжину заданого рядка в символах (або в октетах, якщо рядок є бітовим). Наприклад, результат обчислення виразу CHAR_LENGTH ('Beech') дорівнює 5
CAST	Перетворює значення виразу, побудованого з даних одного типу, в значення даних іншого типу.
CURRENTJJSER ИЛИ USER	Операція конкатенації, Сполучені за допомогою цієї операції два символні або бітові рядки перетворюються в один рядок.
SESSION_USER	Функція повертає символний рядок, що представляє собою поточний ідентифікатор в системі авторизації (або, як прийнято говорити, ім'я облікового запису) поточного користувача
SESSION_USER	Функція повертає символний рядок, що представляє собою ідентифікатор поточного сеансу SQL
SYSTEMJJSER	Функція повертає символний рядок, що представляє собою ідентифікатор користувача, що активізував поточний модуль
LOWER	Функція перетворює в заданому рядку всі великі літери на малі. Наприклад, в результаті обчислення виразу LOWER (SELECT ПІБ FROM Студент WHERE HC = 12) буде отримано значення 'Бабич Ю.'
UPPER	Функція перетворює в заданому рядку всі строкові літери на прописні. Наприклад, в результаті обчислення виразу UPPER (SELECT ПІБ FROM Студент WHERE HC = 12) буде отримано значення 'БАБИЧ Ю.'
TRIM	Функція видаляє зазначені провідні (LEADING), кінцеві (TRAILING) або

	ті та інші (BOTH) символи з заданої рядки. Наприклад, обчислення виразу TRIM (BOTH '*' FROM '*** Hello World ***) дасть результат 'Hello World'
POSITION	Функція повертає позицію одного рядка в іншому рядку.
SUBSTRING	Функція виконує виділення підрядка з заданого рядка.
EXTRACT	Функція повертає значення зазначеного поля з значення типу дати, часу або інтервалу.
CASE	Оператор повертає одне зі значень заданого набору виходячи з результатів перевірки виконання зазначених умов. наприклад: CASE type WHEN 'HC' THEN 1 WHEN 'HG' THEN 2 ELSE 0 END
CURRENT_DATE	Функція повертає поточну дату того часового поясу, в якому знаходиться користувач
CURRENTTIME	Функція повертає поточний час того часового поясу, який в даний час застосовується за умовчанням для поточного сеансу. Наприклад, вираз CURRENTTIME (6) повертає поточний час з точністю до мікросекунд
CURRENT_TIME_STAMP	Функція повертає поточну дату і час того часового поясу, який в даний час застосовується за умовчанням для поточного сеансу. Наприклад, вираз CURRENT_TIMESTAMP (0) поверне тимчасову позначку з точністю до цілих секунд

Засоби підтримки цілісності даних

Підтримка цілісності даних включає засоби завдання обмежень, які вводяться з метою захисту бази від порушення узгодженості збережених в ній даних. Визначено п'ять типів обмежень підтримки цілісності:

- обов'язкові дані;
- обмеження для доменів;
- цілісність сутностей;
- довідкова цілісність;
- вимоги конкретного підприємства.

Ці обмеження можуть бути визначені в операторах CREATE TABLE і ALTER TABLE.

Обов'язкові дані

Для деяких стовпців потрібна наявність в кожному рядку таблиці конкретного і допустимого значення, відмінного від невизначеного значення (або значення NULL). Значення NULL служить для представлення даних, які в даний момент недоступні, відсутні або не визначені. Наприклад, кожен викладач обов'язково має те чи інше вчене звання: професор, доцент і т.п. Для завдання обмежень подібного типу стандарт ISO передбачає використання специфікатор NOT NULL, зазначених вище в операторах CREATE TABLE і ALTER TABLE. Якщо для стовпця заданий специфікатор NOT NULL, система відкидає будь-які спроби вставити в такий стовпець пусте значення. А якщо при визначенні характеристик стовпчика заданий специфікатор NULL, то система допускає розміщення в цьому стовпці значень NULL. Відповідно до стандарту ISO за замовчуванням застосовується специфікатор NULL. Наприклад, для вказівки того, що стовпець СТ (вчене звання) в таблиці Викладач не може містити порожніх значень, слід визначити його, як показано нижче.

СТ VARCHAR (10) NOT NULL

Обмеження для доменів

Кожен стовпець має власний домен, тобто деякий набір допустимих значень. Стандарт ISO передбачає два різних механізми визначення доменів в операторах CREATE TABLE і ALTER TABLE. Перший полягає у використанні конструкції CHECK, що дозволяє задати необхідні обмеження для стовпця або таблиці в цілому. Конструкція CHECK має такий вигляд:

CHECK (searchCandition)

При визначенні обмежень для окремого стовпця в конструкції CHECK можна посилатися лише на визначений стовпець.

Однак стандарт ISO дозволяє визначати і більш складні домени, для чого призначений другий механізм - використання оператора CREATE DOMAIN, що має такий вигляд:

```
CREATE DOMAIN domainName [AS] dataType  
[DEFAULT defaultOption]  
[CHECK (searchCondition)]
```

Кожному створюваному домену присвоюється ім'я, що задається параметром domainName, тип даних, який визначається параметром dataType, необов'язкове значення за замовчуванням, яке встановлюється параметром defaultOption, і необов'язковий набір допустимих значень, що визначається в конструкції CHECK. Слід зазначити, що наведений формат оператора CREATE DOMAIN є неповним, проте його досить для демонстрації основних можливостей.

Значення параметра searchCondition може передбачати звернення до довідкової таблиці. Наприклад, можна створити домен КП (код предмета), який дозволить вводити в відповідні стовпці різних таблиць тільки ті значення, які вже існують в стовпці КП таблиці Предмет. Для цієї мети необхідно використовувати наступний оператор:

```
CREATE DOMAIN КП AS VARCHAR(4)  
CHECK (VALUE IN (SELECT КП FROM Предмет));
```

Видалення доменів з бази даних виконується за допомогою оператора DROP DOMAIN, що має такий вигляд:

```
DROP DOMAIN domainName [RESTRICT | CASCADE]
```

Специфікатор способу видалення домену (RESTRICT або CASCADE) визначає, які дії виконуються в базі даних, якщо домен в даний час використовується. Якщо заданий специфікатор RESTRICT, а домен застосовується в існуючій таблиці, представлення або визначенні перевірки, то операція видалення закінчується невдачею. А якщо заданий специфікатор CASCADE, то в будь-який стовпець таблиці, який заснований на визначенні домену, автоматично вносяться зміни таким чином, щоб в ньому застосовувався базовий тип даних домену, а будь-які обмеження або застосовуються за замовчуванням конструкції операторів для цього домену замінюються в разі необхідності обмеженнями стовпця чи застосовується за умовчанням конструкцією оператора для відповідного стовпчика.

Цілісність сутностей

Первинний ключ таблиці повинен мати унікальне непорожнє значення в кожному її рядку. Наприклад, кожен рядок таблиці Кафедра повинна містити унікальне значення номера кафедри, поміщене в стовпець ККАФ; саме воно буде унікальним чином визначати кафедру, представлену цим рядком таблиці. Стандарт ISO дозволяє задавати подібні вимоги підтримки цілісності даних за допомогою конструкції PRIMARY KEY в операторах CREATE TABLE і ALTER TABLE. Наприклад, для визначення первинного ключа таблиці Кафедра можна використовувати наступну конструкцію:

```
PRIMARY KEY(ККАФ)
```

У разі складеного первинного ключа, наприклад, первинного ключа таблиці Студент, що складається з двох стовпців під іменами НС і НГ, конструкція визначення первинного ключа PRIMARY KEY матиме вигляд PRIMARY KEY (НС, НГ) Конструкція PRIMARY KEY може вказуватися у визначенні таблиці тільки од

раз . Однак існує можливість гарантувати унікальність значень і для будь-яких альтернативних ключів таблиці, для чого призначене ключове слово UNIQUE. Крім того, при визначенні стовпців альтернативних ключів рекомендується використовувати і специфікатор NOT NULL. У кожній таблиці може бути визначено довільну кількість конструкцій UNIQUE. База даних відкидає будь-які спроби виконання операцій INSERT або UPDATE, які тягнуть за собою створення повторюваного значення в будь-якому потенційному ключі (під цим мається на увазі первинний або альтернативний ключ). Наприклад, визначення таблиці Студент можна переписати таким чином:

```
HC VARCHAR(5) NOT NULL,  
HG VARCHAR(5) NOT NULL,  
UNIQUE (HC, HG)
```

Цілісність згідно з посиланням

Зовнішні ключі є стовпці або набори стовпців, призначені для зв'язування кожної з рядків дочірньої таблиці, яка містить цей зовнішній ключ, з рядком батьківської таблиці, яка містить відповідне значення потенційного ключа.

*Поняття **посилальної цілісності** означає, що якщо поле зовнішнього ключа містить деяке значення, то воно обов'язково повинно посилатися на існуючий допустимий рядок в батьківській таблиці.*

Наприклад, значення в стовпці номера кафедри ККАФ таблиці Кафедра завжди має пов'язувати дані про викладача з конкретною рядком таблиці Викладач. Якщо стовець з номером кафедри не порожній, він обов'язково повинен бути допустимим значенням стовпця ККАФ таблиці Викладач. В іншому випадку викладач буде закріплений за неіснуючою кафедрою. Стандарт ISO передбачає механізм визначення зовнішніх ключів за допомогою конструкції FOREIGN KEY операторів CREATE TABLE і ALTER TABLE. Наприклад, для визначення зовнішнього ключа ККАФ в таблиці Кафедра можна використовувати наступну конструкцію:

```
FOREIGN KEY (ККАФ) REFERENCES Викладач
```

Тепер система відхилить виконання будь-яких операторів INSERT або UPDATE, за допомогою яких буде зроблена спроба створити в дочірній таблиці значення зовнішнього ключа, що не відповідає одному з уже існуючих значень потенційного ключа батьківської таблиці. Дії системи, що виконуються при надходженні операторів UPDATE або DELETE, що містять спробу оновити або видалити значення потенційного ключа в батьківській таблиці, якому відповідав би один або кілька рядків дочірньої таблиці, залежать від правил підтримки посилальної цілісності, зазначених в конструкціях ON UPDATE і ON DELETE конструкції FOREIGN KEY . На той випадок, якщо користувач робить спробу видалити з батьківської таблиці рядок, на яку посиляється одна або кілька рядків дочірньої таблиці, в мові SQL передбачені наступні чотири допустимих варіанти дій:

- **CASCADE**. Видалення рядка з батьківської таблиці супроводжується автоматичним видаленням всіх посилальних на неї рядків дочірньої таблиці. Оскільки видалені рядки дочірньої таблиці також можуть містити деякі потенційні ключі, які використовуються в якості зовнішніх ключів в інших таблицях, аналізуються і застосовуються правила обробки зовнішніх ключів цих таблиць, активізується перевірка правил обробки зовнішніх ключів і від

Такий спосіб виконання операції називається каскадним, оскільки він передбачає перехід з одного рівня ієрархії на інший.

- **SET NULL.** Виконується видалення рядка з батьківської таблиці, а в зовнішні ключі всіх що посилаються на неї рядків дочірньої таблиці заносяться значення NULL. Цей варіант можна застосовувати тільки в тому випадку, якщо у визначенні стовпця зовнішнього ключа відсутнє ключове слово NOT NULL.
- **SET DEFAULT.** Виконується видалення рядка з батьківської таблиці, а в зовнішні ключі всіх що посилаються на неї рядків дочірньої таблиці заносяться значення, яке приймається за замовчуванням. Цей варіант можна застосовувати тільки в тому випадку, якщо у визначенні стовпця зовнішнього ключа присутній ключове слово DEFAULT і задано значення, яке використовується за умовчанням.
- **NO ACTION.** . Операція видалення рядка з батьківської таблиці відкидається. Саме це значення використовується за умовчанням в тих випадках, коли в описі зовнішнього ключа конструкція ON DELETE опущена. Ті ж правила застосовуються в мові SQL і тоді, коли значення потенційного ключа батьківської таблиці оновлюється. У разі використання правила CASCADE в стовпці зовнішнього ключа дочірньої таблиці поміщається нове, змінене значення потенційного ключа батьківської таблиці. Аналогічним чином, поновлення каскадно поширюються на інші таблиці, якщо їх зовнішні ключі посилаються на оновлені стовпці дочірньої таблиці. Наприклад, в таблиці Викладач стовпець номера кафедри є зовнішнім ключем, що посилаються на таблицю Кафедра. Для цього зовнішнього ключа можна встановити правило видалення, яке вказує, що в разі видалення запису про кафедру з таблиці Кафедра, відповідне значення в стовпці ККАФ таблиці Викладач має бути замінено значенням NULL:

FOREIGN KEY (ККАФ)

REFERENCES Кафедра ON DELETE SET NULL

Аналогічним чином, стовпець з номером викладача (ТАБН) таблиці Вивчення є зовнішнім ключем, що зв'язує її з таблицею Викладач. Можна встановити правило поновлення, яке вказує, що в разі зміни номера викладача з таблиці Викладач відповідні значення в стовпці ТАБН таблиці Вивчення також повинні бути замінені новим значенням :

FOREIGN KEY (ТАБН) REFERENCES Викладач ON UPDATE CASCADE

Вимоги даного підприємства

Оновлення даних в таблицях можуть бути обмежені існуючими в даній організації вимогами (які прийнято також називати діловим регламентом), встановленими щодо виконання вручну операцій, пов'язаних з внесенням змін до інформації. Стандарт ISO дозволяє реалізувати діловий регламент підприємств або за допомогою конструкцій CHECK і ключового слова UNIQUE в операторах CREATE TABLE і ALTER TABLE, або за допомогою оператора

CREATE ASSERTION. Оператор **CREATE ASSERTION** призначений для введення обмежень цілісності даних, які безпосередньо не пов'язані з визначеннями таблиць. Цей оператор має наступний формат:

CREATE ASSERTION

Якщо вимоги підтримки ділового регламенту пов'язані з використанням даних декількох таблиць, краще застосувати оператор **ASSERTION**, ніж дублювати опис необхідної перевірки в кожній із задіяних таблиць або вносити відомості про обмеження у додаткову таблицю.

Визначення даних

Мова визначення даних SQL DDL (Data Definition Language) дозволяє створювати і знищувати такі об'єкти бази даних, як схеми, домени, таблиці, уявлення і індекси. Стандарт ISO передбачає також можливість створення наборів символів, схем сортування і перетворення.

Нижче перераховані основні оператори мови визначення даних SQL.

CREATE SCHEMA
CREATE DOMAIN
CREATE TABLE
CREATE VIEW

ALTER DOMAIN
ALTER TABLE

DROP SCHEMA
DROP DOMAIN
DROP TABLE
DROP VIEW

Ці оператори використовуються для створення, модифікації та знищення структур, що входять до складу концептуальної схеми. У багатьох СУБД передбачені також наступні два оператора, хоча вони не розглядаються в стандарті SQL:

CREATE INDEX DROP INDEX

Крім того, адміністратор бази даних може скористатися додатковими командами для уточнення параметрів фізичного зберігання даних.

Створення баз даних

У різних СУБД процедура створення баз даних істотно відрізняється. У багатокористувацьких системах право створення баз даних зазвичай закріплюється тільки за адміністратором бази даних (АБД). В однокористувацьких системах передбачена за замовчуванням база даних може

бути створена безпосередньо в процесі установки і настройки параметрів самої СУБД, а інші бази даних створюються самим користувачем у міру необхідності. Стандарт ISO не визначає, як повинні створюватися бази даних, тому в кожному з діалектів мови SQL зазвичай використовується власний підхід. Відповідно до стандарту ISO, таблиці та інші об'єкти бази даних існують в деякому середовищі (environment). Крім усього іншого, кожна середа складається з одного або декількох каталогів (catalog), а кожен каталог - з набору схем (schema). Схема являє собою іменовану колекцію об'єктів бази даних, які певним чином пов'язані один з одним (всі об'єкти в базі даних повинні бути описані в тій чи іншій схемі). Об'єктами схеми можуть бути таблиці, уявлення, домени, затвердження, зіставлення, тлумачення і набори символів. Всі об'єкти схеми мають одного і того ж власника і безліч загальних значень, що застосовуються за замовчуванням. Цей стандарт залишає право вибору конкретного механізму створення і знищення каталогів за розробником СУБД, проте регламентує механізм створення і видалення схем. Оператор визначення схеми має наступний формат (спрощено):

```
CREATE SCHEMA [name | AUTHORIZATION CreatorIdentifier]
```

Таким чином, якщо творцем схеми під ім'ям SqlTests є користувач Smith, то даний оператор буде виглядати наступним чином:

```
CREATE SCHEMA SqlTests AUTHORIZATION Smith;
```

У стандарті ISO також зазначено, що повинна існувати можливість визначити в рамках даного оператора діапазон засобів, доступних користувачам створюваної схеми. Однак конкретні способи визначення подібних привілеїв в різних СУБД відрізняються. Схема видаляється за допомогою оператора DROP SCHEMA, який має такий вигляд:

```
DROP SCHEMA Name [RESTRICT | CASCADE]
```

Якщо вказано ключове слово RESTRICT (саме воно приймається за замовчуванням), схема повинна бути порожньою, інакше виконання операції буде скасовано. Якщо вказано ключове слово CASCADE, при виконанні оператора будуть автоматично видалені всі пов'язані з видаляється схемою об'єкти, причому в порядку, зазначеному вище. Якщо одна з цих операцій видалення буде завершена невдало, виконання всього оператора DROP SCHEMA буде скасовано. Загальний ефект від виконання оператора DROP SCHEMA з параметром CASCADE може торкнутися значну частину бази даних, тому подібні оператори повинні вводитися з винятковою обережністю. В даний час оператори CREATE SCHEMA і DROP SCHEMA реалізовані в дуже небагатьох СУБД.

Створення таблиць (оператор CREATE TABLE)

Після створення загальної структури бази даних можна приступити до створення таблиць, що представляють відношення, що входять до складу проекту

бази даних. Для цієї мети використовується оператор CREATE TABLE, що має такий загальний формат:

```
CREATE TABLED TableName
(columnName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption] CHECK (searchCondition))
[PRIMARY KEY(listOfColumns) ]
{ [UNIQUE] (listOfColumns) ,} [ . . .]
{[FOREIGN KEY (listOfForeignKeyColumns)
REFERENCES Paren TableName[{listOfCandidateKeyColumns} ] ,
MATCH {PARTIAL | FULL}
[ON UPDATE referentialAction]
{[ON DELETE referential Action] [ . . .]}
{[CHECK (searchCondition)] [ . . .] }
```

Ця версія оператора CREATE TABLE включає засоби визначення обмежень посилальної цілісності та інших обмежень. Структура самого оператора і ступінь підтримки тих чи інших обмежень в значній мірі залежать від застосовуваного діалекту мови SQL. Але, як правило, в базі даних слід використовувати всі підтримувані обмеження, оскільки це дозволяє підвищити якість даних, що зберігаються. В результаті виконання цього оператора буде створена таблиця, ім'я якої визначається параметром TableName, що складається з одного або декількох стовпців типу dataType. Для завдання значення, що застосовується за умовчанням при вставці даних в конкретний стовпець, передбачена необов'язкова конструкція DEFAULT. У базі даних це значення застосовується за умовчанням в тих випадках, якщо в операторі INSERT не задано значення для такого стовпчика. Крім інших значень, опція визначення застосовуваного за замовчуванням значення defaultoption може включати літерали. Конструкції NOT NULL, UNIQUE і CHECK. Решта конструкції відомі під назвою обмежень таблиці і можуть бути додатково позначені за допомогою наступної конструкції:

```
CONSTRAINT ConstraintName
```

Ця конструкція дозволяє в подальшому видалити обмеження, вказавши його ім'я в операторі ALTER TABLE. Конструкція PRIMARY KEY визначає один або декілька стовпців, які утворюють первинний ключ таблиці. Якщо ця конструкція передбачена в діалекті SQL, реалізованому в конкретній базі даних, то вона повинна застосовуватися при створенні кожної таблиці. За замовчуванням для всіх стовпців, що представляють первинний ключ, передбачено застосування обмеження NOT NULL. При створенні таблиці дозволено використання тільки однієї конструкції PRIMARY KEY. База даних відкидає всі спроби виконання операцій INSERT або UPDATE, які тягнуть за собою створення рядка з повторюваним значенням в стовпці (стовпцях) PRIMARY KEY. Таким чином, в базі даних гарантується унікальність значень первинного ключа. У конструкції FOREIGN KEY визначається зовнішній ключ (дочірньої) таблиці і її зв'язок з іншою (батьківської) таблицею. Ця конструкція дозволяє реалізувати обмеження посилальної цілісності і складається з наступних частин:

- Список listOfForeignKeyColumns, що містить імена одного або декількох стовпців створюваної таблиці, які утворюють зовнішній ключ.

- Допоміжна конструкція REFERENCES, яка вказує на батьківську таблицю (тобто таблицю, в якій визначено відповідний потенційний ключ). Якщо список UstOfCandidateKsyCalumns опущений, передбачається, що визначення зовнішнього ключа збігається з визначенням первинного ключа батьківської таблиці. В такому випадку батьківська таблиця повинна мати в своєму операторі CREATE TABLE конструкцію PRIMARY KEY.
- Обов'язкове правило оновлення (ON UPDATE) для визначення взаємозв'язку між таблицями, яке вказує, яка дія (referentialAction) має виконуватись при оновленні в батьківській таблиці потенційного ключа, відповідного зовнішньому ключу дочірньої таблиці. Як параметр referentiaJAcCion можна вказати CASCADE, SET NULL, SET DEFAULT АБО NO ACTION. Якщо конструкція ON UPDATE опущена, то за замовчуванням маєтись на увазі, що ніякі дії не виконуються, відповідно до значення NO ACTION.
- Необов'язкове правило видалення (ON DELETE) для визначення взаємозв'язку між таблицями, яке вказує, яка дія (referentialAction) має виконуватись при видаленні рядка з батьківської таблиці, яка містить потенційний ключ, відповідний зовнішньому ключу дочірньої таблиці. Визначення параметра referentzaIAction збігається з визначенням такого ж параметра для правила ON UPDATE.
- За замовчуванням обмеження посилальної цілісності задовольняється, якщо будь-який компонент зовнішнього ключа має значення NULL або в батьківській таблиці є відповідний рядок. Опція MATCH дозволяє ввести додаткові обмеження, що стосуються застосування значень NULL в зовнішньому ключі. Якщо задана опція MATCH FULL, то або всі компоненти зовнішнього ключа повинні бути порожні (NULL), або всі повинні мати непусті значення. А якщо задана опція MATCH PARTIAL, то або всі компоненти зовнішнього ключа повинні бути порожні (NULL), або в батьківській таблиці повинен існувати хоча б один рядок, здатен задовольнити це обмеження, якщо всі інші значення NULL були підставлені правильно. Деякі автори стверджують, що в обмеженнях посилальної цілісності слід застосовувати тільки опцію MATCH FULL.

В операторі створення таблиці може бути задано будь-яку кількість конструкцій FOREIGN KEY. Конструкції CHECK і CONSTRAINT дозволяють визначати додаткові обмеження. Якщо конструкція CHECK використовується в якості обмеження стовпця, то вона може посилатися тільки на визначений стовпець. Обмеження фактично контролюються після застосування кожного оператора SQL до таблиці, на якій вони задані, але така перевірка може бути відкладена до закінчення тієї транзакції, до складу якої входить поточний оператор SQL.

Приклад 7.2.1. Застосування оператора CREATE TABLE

Створіть таблицю Успішність з використанням всіх доступних коштів оператора CREATE TABLE.

```
CREATE DOMAIN HG AS VARCHAR(5)
CHECK (VALUE IN (SELECT НомерГрупи FROM Студент));
CREATE DOMAIN HC AS VARCHAR(2)
```

```

CHECK (VALUE IN (SELECT НомерСтудента FROM Студент ) ) ;
CREATE DOMAIN КП AS VARCHAR(2)
CHECK (VALUE IN (SELECT КодПредмета FROM Предмет));
CREATE DOMAIN ТАБН AS VARCHAR(2 );
CHECK (VALUE IN (SELECT Таб.номерВикладача FROM Викладач));
CREATE DOMAIN ВИДЗ AS VARCHAR(15);
CREATE DOMAIN ОЦІНКА AS VARCHAR(1);
CREATE TABLE Успеваемость(
НомерГруппы          НГ          NOT NULL,
Номер студента       НС          NOT NULL,
Код предмета        КП          NOT NULL,
Таб.номерВикладача  ТАБН       NOT NULL,
ВидЗаняття          ВИДЗ       NOT NULL DEFAULT 'Л',
Оцінка              ОЦІНКА  NOT NULL
PRIMARY KEY (НомерГруппы, Номер студента, Код предмета,
Таб.номерВикладача,
ВидЗаняття),
FOREIGN KEY (НомерСтудента) REFERENCES Студент ON DELETE SET NULL
ON UPDATE CASCADE,
FOREIGN KEY (Таб.номерВикладача) REFERENCES Викладач
ON DELETE NO NULL
ACTION ON UPDATE
CASCADE,
FOREIGN KEY (КодПредмета) REFERENCES Предмет ON DELETE NO NULL
ACTION ON UPDATE
CASCADE);

```

Колонці ВЗ з позначенням виду заняття за замовчуванням присвоюється значення 'Л' (скорочення від 'Лекція'). В якості первинного ключа застосовується номер групи, номер студента, код предмета, таб. Номер викладача, вид заняття. База даних автоматично підтримує унікальність значень в цьому стовпці. Стовпець з даними номерами студентів (НС) визначений як стовпець зовнішнього ключа, який посиляється на таблицю Студент. Визначено правило видалення, згідно з яким при видаленні запису з таблиці Студент відповідних значень стовпця НС таблиці Успішність присвоюється значення NULL. Крім того, визначено правило поновлення, згідно з яким при оновленні НС в таблиці Студент оновлюються також відповідні значення в стовпці НС таблиці Успішність. ТАБН (Таб.номерПреподавателя) визначено як зовнішній ключ, який посиляється на таблицю Викладач. Визначено також правило видалення NO ACTION для запобігання видалення даних з таблиці Викладач, якщо в таблиці Успішність є відповідні значення ТАБН. Задане в цьому операторі правило поновлення CASCADE визначено таким чином, що при ТАБН відповідним значенням в стовпці Таб.номерПреподавателя таблиці Успішність присвоюється це оновлене значення. Оскільки у всіх обмеженнях FOREIGN KEY опущений список ListOfCandidateKeyColumns, передбачається,

що визначення зовнішніх ключів збігаються з визначеннями первинних ключів в відповідних батьківських таблицях.

Модифікація визначення таблиці (оператор ALTER TABLE)

У стандарті ISO передбачено застосування оператора ALTER TABLE для зміни структури таблиці після її створення. Визначення оператора ALTER TABLE складається з шести опцій, що дозволяють виконати наступні дії:

- ввести новий стовпець в таблицю;
- видалити стовпець з таблиці;
- ввести нове обмеження таблиці;
- видалити обмеження таблиці;
- задати для стовпця значення, що застосовується за умовчанням;
- видалити опцію, яка передбачає застосування для стовпця значення, заданого за замовчуванням.

Нижче наведено основний формат цього оператора.

```
ALTER TABLE TableName  
[ADD [COLUMN] columnName dataType [NOT NULL] [UNIQUE]  
[DEFAULT defaultOption] [CHECK searchCondition)]  
[COLUMN] columnName [RESTRICT \ CASCADE]  
[ADD [CONSTRAINT [ConstraintName] ] tableConstraintDefinition]  
[DROP CONSTRAINT ConstraintName [RESTRICT \ CASCADE]]  
[ALTER [COLUMN] SET DEFAULT defaultOption]  
[ALTER [COLUMN] DROP DEFAULT]
```

Майже всі параметри даного оператора збігаються з параметрами оператора CREATE TABLE. Як параметр з визначенням обмеження таблиці tableConstraintDefinition може застосовуватися одна з конструкцій PRIMARY KEY, UNIQUE, FOREIGN KEY або CHECK. Конструкція ADD COLUMN аналогічна конструкції визначення стовпця в операторі CREATE TABLE. У конструкції DROP COLUMN задається ім'я стовпця, що видаляється з визначення таблиці, і є необов'язкова опція, що дозволяє вказати, чи є дія операції DROP каскадною чи ні, як показано нижче.

- RESTRICT. Операція DROP відкидається, якщо на даний стовпець є посилання в іншому об'єкті бази даних (наприклад, у визначенні подання). Це значення опції передбачено за замовчуванням.
- CASCADE. Рятувальна операція DROP триває в будь-якому випадку і посилання на стовпець автоматично видаляються з будь-яких об'єктів бази даних, де вони є. Ця операція виконується каскадно, тому якщо стовпець видаляється з об'єкту, що містить посилання, то в базі даних виконується перевірка того, чи є посилання на цей стовпець в будь-якому іншому об'єкті, такі посилання знищуються і в цьому об'єкті, і т.д.

Приклад 7.2.2. Застосування оператора ALTER TABLE

Змініть визначення таблиці Група, видаливши застосовуване за умовчанням значення 'ТМ-31' для стовпця НГ (номер студента в групі), а для стовпця РІКНАР (рік народження) передбачте за замовчуванням значення '1986'.

```
ALTER TABLE Група  
ALTER ТМ-31 DROP DEFAULT;  
ALTER TABLE Група  
ALTER РОДН SET DEFAULT '1986';
```

Змініть визначення таблиці Група, видаливши обмеження, згідно з яким в групі не повинно бути менше 0 і більше 35.

```
ALTER TABLE Група  
DROP CONSTRAINT К-СТЪ
```

Оператор **ALTER TABLE** доступний не у всіх діалектах мови **SQL**. А в деяких діалектах оператор **ALTER TABLE** не може використовуватися для видалення існуючого стовпця з таблиці. У подібних випадках, якщо стовпець більше не потрібен, він може просто ігноруватися, але залишатися у визначенні таблиці. Але якщо дійсно потрібно видалити стовпець з таблиці, то необхідно виконати наступне:

- вивантажити всі дані з таблиці;
- видалити визначення таблиці з бази даних за допомогою оператора **DROP TABLE**;
- визначити нову таблицю за допомогою оператора **CREATE TABLE**;
- знову завантажити дані в нову таблицю.

Зазначені етапи вивантаження і завантаження, як правило, виконуються за допомогою допоміжних програм спеціального призначення, які входять в поставку СУБД. Але в будь-якому випадку існує також можливість створити тимчасову таблицю, застосувати оператор **INSERT, SELECT** для завантаження даних зі старої таблиці в тимчасову таблицю, а потім з тимчасової таблиці в нову таблицю.

Видалення таблиць (оператор **DROP TABLE**)

З плином часу структура бази даних змінюється: створюються нові таблиці, а колишні стають непотрібними. Непотрібні таблиці видаляються з бази даних за допомогою оператора **DROP TABLE**, що має такий вигляд:

```
DROP TABLE.TableWame [RESTRICT | CASCADE]
```

Наприклад, для видалення таблиці **PropertyForRent** можна використовувати наступний оператор:

```
DROP TABLE PropertyForRent;
```

Однак слід зазначити, що ця команда видалить не тільки зазначену таблицю, але і всі вхідні в неї рядки даних. Якщо потрібно видалити з таблиці лише рядки даних, зберігши в базі опис самої таблиці, то слід використовувати

оператор DELETE. Оператор DROP TABLE додатково дозволяє зазначено, чи це операцію видалення виконувати каскадно.

- **RESTRICT.** Операція DROP відкидається, якщо в базі даних є інші об'єкти, існування яких залежить від того, чи існує в базі даних видаляється таблиця.
- **CASCADE.** Операція DROP триває, і з бази даних автоматично видаляються всі залежні об'єкти (і об'єкти, що залежать від цих об'єктів).

Загальний ефект від виконання оператора DROP TABLE з ключовим словом CASCADE може поширюватися на значну частину бази даних, тому подібні оператори слід використовувати з максимальною обережністю. Найчастіше оператор DROP TABLE використовується для виправлення помилок, допущених при створенні таблиці. Якщо таблиця була створена з неправильною структурою, можна скористатися оператором DROP TABLE для її видалення, після чого створити таблицю заново.

Створення індексу (оператор CREATE INDEX)

Індекс являє собою структуру, що дозволяє виконувати прискорений доступ до рядків таблиці з урахуванням значень одного або декількох її стовпців.

Наявність індексу може істотно підвищити швидкість виконання деяких запитів. Але оскільки індекси повинні оновлюватися системою при кожному внесенні змін до їх базової таблиці, вони створюють додаткове навантаження на систему. Індекси зазвичай створюються з метою задоволення певних критеріїв пошуку, після того як таблиця вже перебувала деякий час в роботі і збільшилася в розмірах. Створення індексів не передбачено стандартом мови SQL. Однак більшість діалектів підтримує як мінімум наступний оператор:

CREATE [UNIQUE] INDEX IndexName

ON TableName (columnName [ASC | DESC], [...])

Зазначені в операторі стовпчики становлять ключ індексу і повинні бути перераховані в порядку зменшення значимості. Індекси можуть створюватися тільки для таблиць бази даних, але не для представлень. Якщо в операторі вказано ключове слово UNIQUE, унікальність значень ключа індексу буде автоматично підтримуватися СУБД. Вимога унікальності значень обов'язково для первинних ключів, а також, можливо, і для інших стовпців таблиці (наприклад, для альтернативних ключів). Хоча створення індексів здійснено в будь-який момент, при побудові індексу для вже заповненої даними таблиці можуть виникнути проблеми, пов'язані з дублюванням даних в різних рядках. Отже, має сенс створювати унікальні індекси (хоча б для первинного ключа)

безпосередньо при створенні таблиці. В результаті система відразу ж візьме на себе контроль над унікальністю значень даних у стовпцях.

Для таблиць Група повинен бути створений, по крайній мере, такий індекс:

CREATE UNIQUE INDEX ГрупаNoInd ON Група (НГ);

Для кожного з ключових стовпців може бути вказаний порядок проходження значень - по зростанню (ASC) чи спадний (DESC), причому значення ASC використовується за умовчанням. Наприклад, для таблиці Викладач можна створити наступний індекс:

CREATE INDEX ПреподавательInd ON Викладач (ПІБ, ТАБН),

При обробці цього оператора буде створений файл під ім'ям ПреподавательInd, що містить дані новоствореного індексу таблиці Викладач. Рядки в цьому файлі будуть розташовані в порядку зростання значень стовпця ПІБ, а всередині них - у порядку зростання значень стовпця ТАБН.

Видалення індексу (оператор DROP INDEX)

Якщо для таблиці бази даних був створений індекс, який згодом виявився непотрібним, то його можна видалити за допомогою оператора DROP INDEX. Цей оператор має наступний формат:

DROP INDEX IndexName

Представлення

Представлення - динамічно сформований результат однієї або декількох реляційних операцій, виконаних над відношеннями бази даних з метою отримання нового відношення.

Представлення є віртуальним відношенням, яке не завжди реально існує в базі даних, але створюється за запитом певного користувача в ході виконання цього запиту. З точки зору користувача бази даних представлення виглядає як реальна таблиця даних, що містить набір поіменованих стовпців і рядків даних. Але на відміну від реальних таблиць представлення не завжди існують в базі як деякий набір збережених значень даних. Насправді доступні через представлення рядки і стовпці даних є результатом виконання запиту, заданого при визначенні подання. СУБД зберігає визначення представлення в базі даних. Виявивши посилання на представлення, СУБД застосовує один з двох наступних підходів для формування представлення. При першому підході СУБД відшукує визначення представлення і перетворюють вихідний запит, що лежить в основі представлення, в еквівалентний запит до таблиць, використаним у визначенні представлення, після чого модифікований запит виконується. Цей процес злиття запитів, називається заміною представлення (під цим мається на увазі заміна представлення оператором SQL, який звертається до базових таблиць).

При другому підході, який називається матеріалізацією представлення, готове представлення зберігається в базі даних у вигляді тимчасової таблиці, а його актуальність постійно підтримується по мірі відновлення всіх таблиць, що лежать в його основі. Тепер познайомимося з тим, як створюються і використовуються представлення.

Створення представлень (оператор CREATE VIEW)

Оператор CREATE VIEW має такий вигляд:

```
CREATE VIEW ViewName [ (newColumnTfame [, . . . ] ) ]
```

```
AS subselect [WITH |CASCADED |LOCAL| CHECK OPTION]
```

Представлення визначається за допомогою підзапиту subselect, оформленого у вигляді оператора SELECT мови SQL. Існує необов'язкова можливість присвоєння власного імені кожному зі стовпців представлення. Якщо вказується список імен стовпців, то він повинен мати кількість елементів, яка дорівнює кількості стовпців в результуючій таблиці запиту, заданого параметром subselect. Якщо список імен стовпців опущений, кожен стовпець представлення матиме ім'я відповідного стовпчика результуючої таблиці запиту, заданого параметром subselect. Список імен стовпців повинен обов'язково задаватися в тому випадку, якщо в іменах стовпців результуючої таблиці має місце неоднозначність. Подібна ситуація виникає в тих випадках, коли підзапит включає обчислювані поля, а конструкція AS з іменами стовпців результуючої таблиці не містить для них імен або ж коли результуюча таблиця створюється за допомогою операції з'єднання і включає стовпці з однаковими іменами.

Заданий параметром subseJect підзапит прийнято називати визначальним запитом.

Якщо вказана конструкція WITH CHECK OPTION, то гарантується, що в тих випадках, коли рядок даних не задовольняє умові, вказаною в конструкції WHERE визначає запиту представлення, вона не буде додана в його базову таблицю. Слід зазначити, що для успішного створення представлення необхідно мати привілей SELECT для всіх згадуваних у визначальному запиті таблиць, а також привілей USAGE для всіх доменів, які використовуються в зазначених у запиті стовпцях. Хоча всі представлення створюються за допомогою одного і того ж методу, на практиці для різних цілей використовуються різні типи представлень. Призначення представлень різних типів ми продемонструємо на кількох конкретних прикладах.

Приклад 7.2.3. Створення горизонтального представлення

Створіть представлення, що дозволяє викладачеві переглядати дані тільки тих студентів, у яких прохідний бал дорівнює '3,8'. Горизонтальне представлення дозволяє обмежити доступ користувачів лише певними рядками, вибраними з однієї або декількох таблиць.

```
CREATE VIEW БаллСтудент  
AS SELECT *  
FROM Студент  
WHERE СРБАЛ = '3,8';
```

В результаті виконання цього оператора буде створено представлення БаллСтудент, що включає всі стовпці таблиці Студент, але містить тільки ті її рядки, в яких середній бал студентів дорівнює '3,8'. (Строго кажучи, в цьому представленні стовпець СРБАЛ не є необхідним і цілком може бути виключений з визначення

представлення, оскільки він завжди буде містити одне і те ж значення - '3,8'.) Якщо тепер виконати наведений нижче оператор SQL, його результат буде мати вигляд, показаний в табл. 7.2.3.

SELECT * FROM БалСтудент;

Таблиця 7.2.3. Дані, доступні в представленні БалСтудент

Номер студента в групі	Група	ПІБ	Рік народж	Адреса	Середній бал
12	ТМ-31	Бабич Ю.	1986	с.Жмеринка	3,8
19	ТЗ-31	Самойленко Ю.	1986	Бердичів	3,8

Для забезпечення того, щоб викладач міг бачити тільки ті рядки таблиці Студент, доступ до яких йому дозволено, слід повністю заборонити йому прямий доступ до цієї таблиці, але дозволити доступ до представлення БалСтудент. В результаті викладач матиме власне представлення даних таблиці Студент, що містить відомості тільки про тих студентів, у яких середній бал дорівнює 3,8.

Приклад 7.2.4. Створення вертикального представлення

Створіть представлення, що містить дані тільки тих студентів, у яких прохідний бал дорівнює '3.8', але не включає відомостей про їх адресу, оскільки відомості про адресу студентів повинні бути доступні тільки викладачам. Вертикальні представлення дозволяють обмежити доступ користувачів лише певними стовпцями, вибраними з однієї або декількох таблиць бази даних.

```
CREATE VIEW БаллСтудентАдрес
AS SELECT НС, НГ, ПІБ, Годро
FROM Студен
WHERE СРБАЛЛ = '3,8';
```

Зверніть увагу, що визначення цього представлення можна переписати, скориставшись замість таблиці Студент поданням БалСтудент:

```
CREATE VIEW БаллСтудентАдрес
AS SELECT НС, НГ, ПІБ, Годро
FROM БаллСтудент;
```

У будь-якому випадку буде створено уявлення БалСтудентАдреса, що містить всі стовпці таблиці Студент, за винятком стовпців адресу і середній бал. Дані, що містяться в цій виставі, показані в табл. 7.2.4.

Таблиця 7.2.4. Дані, доступні в поданні БалСтудентАдреса

Номер студента в групі	Група	ПІБ	РІК народж
12	ТМ-31	Бабич Ю.	1986
19	ТЗ-31	Самойленко Ю.	1986

Для того щоб гарантувати, що тільки викладач матиме доступ до відомостей про адресу студентів, всім студентам, у яких прохідний бал дорівнює '3.8' має бути

надано право доступу тільки до представлення БалСтудент, але не до самої таблиці Студент або представлення БалСтудентАдрес.

Представлення, в яких опускається кілька стовпців початкової таблиці, часто називають вертикальними представленнями, оскільки при їх створенні таблиця розділяється на частини по вертикалі.

Вертикальні представлення зазвичай використовуються в тих випадках, коли зберігаються в таблиці дані обробляються різними користувачами або групами користувачів, За допомогою вертикальних представлень в розпорядження користувачів кожного типу надається віртуальна таблиця, що складається тільки з тих стовпців, які їм необхідні.

Приклад 7.2.5. Групування за кількома стовпцями

Визначте кількість оцінок, отриманих кожним із студентів.

```
SELECT s.НГ, s.НС, COUNT (*) AS count
FROM Студент s, Успішність y
WHERE s.НС = y.НС AND s.НГ = y.НГ
GROUP BY s.НС, s.НГ
ORDER BY s.НС, s.НГ;
```

Щоб скласти необхідний звіт, перш за все необхідно з'ясувати, хто зі студентів інституту отримував оцінки. Це завдання можна вирішити за допомогою з'єднання таблиць СТУДЕНТ і УСПІШНІСТЬ за стовпцями НГ і НС в конструкціях FROM / WHERE. Потім необхідно сформувати групи, що складаються з номера групи і номерів її студентів, для чого слід застосувати конструкцію GROUP BY. Нарешті, результуюча таблиця повинна бути відсортована за допомогою завдання конструкції ORDER BY. Результати виконання запиту представлені а табл. 7.2.5.

Таблиця 7.2.5. Результат виконання запиту з прикладу

Ном. студента	Номер групи	count
12	ТМ-31	1
15	ТЗ-32	1
19	ТЗ-31	1
20	ТІ-32	1
21	ТМ-33	1
23	ТІ-31	1

Приклад 7.2.6. Представлення, що формується шляхом групування і з'єднання

Створіть представлення, що містить дані про кількість оцінок, отриманих кожним студентом. Воно повинно включати номер студента, номер групи і відомості про кількість оцінок, які він отримав (див. Попередній приклад).

```
CREATE VIEW СтудентУспевCnt (НС, НГ, count)
AS SELECT s.НГ, s.НС, COUNT(*)
FROM Студент s, Успішність y
WHERE s.НС = y.НС
```

GROUP BY s.НС, y.НГ;

В Внаслідок будуть отримані дані, показані в табл. 7.2.6. Цей приклад ілюструє використання підзапиту, що містить конструкцію GROUP BY (в результаті чого створюється представлення, зване згрупованим представленням) і звертається до даних декількох таблиць (подібні представлення називають сполученими представленнями). Однією з найважливіших причин використання представлень є прагнення до спрощення багатотабличних запитів. Після визначення представлення із з'єднанням декількох таблиць можна буде використовувати найпростіші однотобличні запити до цього представлення замість складних запитів з виконанням того ж самого багатотабличного з'єднання. Відзначимо, що в наведене вище визначення представлення потрібно включити список імен стовпців, оскільки в ньому використана агрегує функція COUNT.

Таблиця 7.2.6. Дані, доступні в поданні СтудентУспевCnt

Ном. студента	Номер групи	count
12	ТМ-31	1
15	ТЗ-32	1
19	ТЗ-31	1
20	ТІ-32	1
21	ТМ-33	1
23	ТІ-31	1

Видалення представлень (оператор DROP VIEW)

Представлення видалається з бази даних за допомогою оператора DROP VIEW, що має такий вигляд:

VIEW ViewName [RESTRICT | CASCADE]

При виконанні оператора DROP VIEW визначення представлення видалається з бази даних.

Якщо вказано ключове слово CASCADE, при виконанні оператора DROP VIEW будуть також видалені всі пов'язані з ним або залежні від нього об'єкти. Іншими словами, будуть видалені всі об'єкти, які містять посилання на видалення представлення. Це означає, що такий оператор DROP VIEW видалає також всі представлення, які визначені на основі видалення представлення. Якщо вказується ключове слово RESTRICT і існують будь-які інші об'єкти, що залежать від існування видалається представлення, виконання цього оператора блокується. За замовчуванням приймається значення RESTRICT.

Заміна представлень

Для ілюстрації процесу заміни представлення оператором SQL розглянемо наведений нижче запит, призначений для оцінок, які отримав кожен із студентів, які мають середній бал 3,8. Цей запит звертається до подання СтудентУспевCnt, визначення якого приведення в описі прикладу 6: SELECT НС, count

FROM СтудентУспішCnt
WHERE СРБАЛ = '3,8'
ORDER BY НС;

Заміна представлення полягає в злитті наведеного вище запиту з визначальним запитом уявлення СтудентУспішCnt і виконується наступним чином:

1. Імена стовпців, зазначені в списку конструкції SELECT запиту, перетворюються в відповідні їм імена стовпців запиту, що визначається.

SELECT s.НС AS НС, COUNT (*) AS count

2. Імена представлень, зазначені в конструкції FROM запиту, замінюються відповідними списками з конструкцій FROM запиту, що визначається:

FROM Студент s, Успішність y

3. Конструкція WHERE вихідного запиту користувача об'єднується з конструкцією WHERE з визначального запиту представлення за допомогою логічного оператора AND:

WHERE s.НС = y.НС AND СРБАЛ = '3,8'

4. Конструкції GROUP BY і HAVING з визначального запиту представлення просто копіюються у вихідний запит. У нашому прикладі у визначальному запиті присутній тільки конструкція GROUP BY:

GROUP BY s.НС, y.НГ;

5. У об'єднаний запит копіюється конструкція ORDER BY з початкового запиту, в якому імена стовпців представлення перетворені в імена стовпців запиту, що визначається:

ORDER BY НС;

6. В результаті всіх перерахованих вище операцій об'єднаний запит набуває такого вигляду:

SELECT s.НС AS НС, COUNT (*) AS count

FROM Студент s, Успішність y

WHERE s.НС = y.НС AND СРБАЛ = '3,8'

GROUP BY s.НС, y.НГ;

ORDER BY НС;

Таблиця 7.2.7. Результируюча таблиця запиту, отриманого в результаті заміни представлення

Ном. студента	Номер групи	count
12	ТМ-31	1
19	ТЗ-31	1

Обмеження на використання представлень

Стандарт ISO накладає кілька важливих обмежень на створення та використання представлень, хоча в цьому відношенні між різними діалектами мови SQL існують значні відмінності.

• Якщо стовпець в представленні створюється з використанням агрегуючої функції, він може вказуватися тільки в конструкціях SELECT і ORDER BY тих запитів, в яких здійснюється доступ до даного представлення. Зокрема, подібний стовпець не може використовуватися в конструкції WHERE, а також не може

бути параметром в агрегуючій функції будь-якого із запитів, заснованого на даному представленні.

• Згруповані представлення ніколи не мають поєднуватися з таблицями бази даних або іншими представленнями.

Оновлення даних в уявленнях

Все обновления, выполненные в таблице базы данных, немедленно отражаются на всех представлениях, предусматривающих обращения к этой таблице. Подобным же образом, можно ожидать, что если данные были изменены в представлении, то это изменение будет отображено и в той таблице (таблицах) базы данных, на которой оно построено. Однако такое условие не всегда соблюдается.

В стандарте ISO указано, что представления могут быть обновляемыми только в тех системах, которые отвечают некоторым четко определенным требованиям. Согласно стандарту ISO представление может быть обновляемым только в том случае, если:

- в его определении не используется ключевое слово DISTINCT, т.е. из результатов определяющего запроса не исключаются повторяющиеся строки;
- каждый элемент в списке конструкции SELECT определяющего запроса представляет собой имя столбца (а не константу, выражение или агрегирующую функцию), причем имя каждого из столбцов упоминается в этом списке не более одного раза;
- в конструкции FROM указана только одна таблица, т.е. представление должно быть создано на базе единственной таблицы, по отношению к которой пользователь должен обладать необходимыми правами доступа. Если исходная таблица сама является представлением, то это представление также должно отвечать указанным условиям. Данное требование исключает возможность обновления любых представлений, построенных на базе соединения, объединения (UNION), пересечения (INTERSECT) или разности (EXCEPT) таблиц;
- в конструкцию WHERE не входят какие-либо вложенные запросы типа SELECT, которые ссылаются на таблицу, указанную в конструкции FROM;
- определяющий запрос не должен содержать конструкций GROUP BY и HAVING.

Кроме того, любая строка данных, добавляемая с помощью представления, не должна нарушать требования поддержки целостности данных, установленные для исходной таблицы представления. Например, при добавлении с помощью представления новой строки таблицы во все столбцы таблицы, отсутствующие в этом представлении, будут введены значения NULL. Однако при этом не должны нарушаться все требования NOT NULL, указанные в описании исходной таблицы представления. Основную концепцию, используемую при формулировании обсуждаемых ограничений, можно выразить с помощью следующего определения.

Обновляемое представление. *Для того чтобы представление было обновляемым, СУБД должна иметь возможность однозначно отобразить любую его строку или столбец на соответствующую строку или столбец его исходной таблицы.*

Всі оновлення, виконані в таблиці бази даних, негайно відбиваються на всіх представленнях, які передбачають звернення до цієї таблиці. Подібним же чином, можна очікувати, що якщо дані були змінені в поданні, то ці зміни буде відображено і в тій таблиці (таблицях) бази даних, на якій воно побудовано. Однак така умова не завжди дотримується.

У стандарті ISO зазначено, що представлення можуть бути поновлюваними тільки в тих системах, які відповідають деяким чітко визначеним вимогам. Відповідно до стандарту ISO представлення може бути оновлюваним тільки в тому випадку, якщо:

- в його визначенні не використовується ключове слово **DISTINCT**, тобто з результатів визначає запиту не виключаються повторювані рядки;
- кожен елемент в списку конструкції **SELECT** запиту, що визначається, є ім'ям стовпця (а не константа, вираз або агрегуюча функція), причому ім'я кожного з стовпців згадується в цьому списку не більше одного разу;
- в конструкції **FROM** вказана тільки одна таблиця, тобто представлення має бути створено на базі єдиної таблиці, по відношенню до якої користувач повинен володіти необхідними правами доступу. Якщо вихідна таблиця сама є представленням, то це представлення також має відповідати зазначеним умовам. Дана вимога виключає можливість поновлення будь-яких представлень, побудованих на базі з'єднання, об'єднання (**UNION**), перетину (**INTERSECT**) або різниці (**EXCEPT**) таблиць;
- в конструкцію **WHERE** не належать будь-які вкладені запити типу **SELECT**, які посилаються на таблицю, зазначену в конструкції **FROM**;
- запит, що визначається, не повинен містити конструкцій **GROUP BY** і **HAVING**.

Крім того, будь-який рядок даних, що додається за допомогою представлення, не повинен порушувати вимоги підтримки цілісності даних, встановлених для початкової таблиці представлення. Наприклад, при додаванні за допомогою представлення нового рядка таблиці в усі стовпці таблиці, відсутні в цьому представленні, будуть введені значення **NULL**. Однак при цьому не повинні порушуватися всі вимоги **NOT NULL**, зазначені в описі вихідної таблиці представлення. Основну концепцію, яка використовується при формулюванні обговорюваних обмежень, можна виразити за допомогою наступного визначення.

Оновлюване представлення. Для того щоб представлення було оновлюваним, СУБД повинна мати можливість однозначно відобразити будь-який його рядок або стовпець на відповідний рядок або стовпець його вихідної таблиці.

Використання конструкції WITH CHECK OPTION

У представлення поміщаються тільки ті рядки, які задовольняють умові **WHERE** у визначальному запиті. Якщо рядок в представленні буде змінено таким чином, що перестане задовольняти цій умові, то цей рядок має зникнути з даного представлення. Аналогічним чином, в представленні будуть з'являтися нові рядки щоразу, коли вставка або оновлення даних в представленні призведе до того, що нові рядки будуть задовольняти умові **WHERE**.

Рядки, які додаються або виключаються з представлення в залежності від містяться в них даних, прийнято називати мігруючими.

У загальному випадку для запобігання міграції рядків представлення використовується конструкція WITH CHECK OPTION оператора CREATE VIEW. Необов'язкові ключові слова LOCAL / CASCADED застосовні в разі існування ієрархії представлень, тобто коли представлення створюється на базі іншої думки. Якщо вказана конструкція WITH LOCAL CHECK OPTION, то такі дії, як вставка або оновлення даних в деякому розумінні, на базі якого прямо або побічно визначені інші представлення, не можуть викликати зникнення рядки з даного представлення, за винятком випадків, коли даний рядок зникає також з уявлень або таблиць інших рівнів ієрархії. При вказівці конструкції WITH CASCADED CHECK OPTION (це значення приймається за замовчуванням) в разі вставки або поновлення рядка в даному поданні або в будь-якому іншому поданні, прямо або побічно певному на базі даного, зникнення цього рядка з даного представлення не допускається.

Така функція може виявитися настільки корисною, що працювати з представленнями виявиться зручніше, ніж з таблицями бази даних. У тому випадку, коли оператор INSERT або UPDATE порушує умови, зазначені в конструкції WHERE запиту, що визначається, виконання затребуваної операції скасовується. В результаті з'являється можливість реалізувати в базі даних додаткові обмеження, спрямовані на збереження цілісності даних. Слід зазначити, що конструкцію WITH CHECK OPTION можна вказувати тільки для оновлюваних представлень.

Приклад 7.2.7. Використання опції WITH CHECK OPTION

Ще раз звернемося до представлення, створеному в прикладі 7.2.3:CREATE VIEW БалСтудент

```
AS SELECT *  
FROM Студент  
WHERE СРБАЛ= '3,8';  
WITH CHECK OPTION;
```

Вміст цього представлення наведено в табл. 7.2.3. Тепер спробуємо змінити значення середнього бала студента в кожному з рядків з '3,8' на '4', наприклад, за допомогою наступного чого оператора:

```
UPDATE БалСтудент  
SET СРБАЛЛ = '4'  
WHERE НС = '12';
```

Оскільки у визначенні представлення була вказана конструкція WITH CHECK OPTION, виконання цього оператора буде заблоковано, тому що в іншому випадку відбудеться міграція рядки з даного горизонтального представлення. Таким чином, не матиме успіху і спроба виконання наступного оператора:

```
INSERT INTO БалСтудент  
VALUES (13, 'ТЗ-31', 'Сушко А.', 1986, 'о.Сицилія', 4);
```

Оскільки у визначенні представлення вказана конструкція WITH CHECK OPTION, введення зазначеного рядка в дане представлення буде заблоковано, так як відразу після поміщення його в вихідну таблицю Студент він повинен буде негайно зникнути з даного представлення.

А тепер розглянемо ситуацію, коли представлення БалСтудент створюється не на базі самої таблиці Студент, а на базі іншої думки, створеного для цієї таблиці.

```
CREATE VIEW МладшийСт  
AS SELECT * AS SELECT *
```

```

FROM Студент FROM МолодшийСт
WHERE РІКНАР> 1983; WHERE РІКНАР> тисячу дев'ятсот вісімдесят п'ять
WITH LOCAL CHECK OPTION ;;
CREATE VIEW БалСтудент
AS SELECT *
FROM СтаршийСт
WHERE СРБАЛ = '3,8';
Тепер спробуємо виконати в представленні БалСтудент наступний оператор:
UPDATE Managers Staff
SET РІКНАР = 1 984
WHERE НС = '12';

```

Однак це нам не вдасться. Справа в тому, що після виконання даного оператора змінений рядок зникне тільки з представлення СтаршийСт, але збережеться в представленні МолодшийСт, на базі якого сформовано представлення СтаршийСт. Якщо ж при оновленні встановити нове значення року народження, рівним 1986, то ця операція буде успішною, оскільки змінений рядок повинен буде зникнути і з представлення МолодшийСт. Якщо ж при визначенні представлення СтаршийСт вказати конструкцію WITH CASCADED CHECK OPTION, то при вказівці будь-якого із згаданих значень року народження (як 1984, так і 1986) виконання цього оновлення буде заблоковано, оскільки змінений рядок повинна буде зникнути з представлення СтаршийСт. Отже, для того щоб отримати гарантію, що подібні аномалії оновлення ніколи не матимуть місця, кожне з створюваних представлень про порушення конструкції WITH CASCADED CHECK OPTION.

Переваги та недоліки представлень

Практика обмеження доступу деяких користувачів до даних за допомогою створення спеціалізованих уявлень, безумовно, має значні переваги перед наданням їм прямого доступу до таблиць бази даних. Однак використання представлень в середовищі SQL не позбавлене недоліків. У цьому розділі ми коротко обговоримо як гідності, так і недоліки, властиві уявленням мови SQL. Ці переваги і недоліки перераховані в табл. 7.2.8.

Таблиця 8. Основні переваги і недоліки представлень мови SQL

Переваги	Недоліки
Незалежність від даних	Обмежені можливості поновлення
актуальність	структурні обмеження
Підвищення захищеності даних	зниження продуктивності
зниження складності	
додаткові зручності	
можливість налаштування	
Забезпечення цілісності даних	

Переваги

1. Незалежність від даних.

За допомогою представлень можна створити узгоджену, незмінну картину структури бази даних, яка буде залишатися стабільною навіть в разі зміни формату

вихідних таблиць (наприклад, додавання або видалення стовпців, зміни зв'язків, поділу таблиць, їх реструктуризації або перейменування). Якщо в таблицю додаються або з неї видаляються стовпці, які не використовуються в представленні, то змінювати визначення цього представлення не буде потрібно. Якщо структура вихідної таблиці переупорядковувалась або ця таблиця розділяється, то можна буде створити представлення, що дозволяє користувачам працювати з віртуальною таблицею колишнього формату. У випадку розділення вихідної таблиці колишній формат може бути віртуально відновлений за допомогою представлення, побудованого на основі поєднання новостворених таблиць, - звичайно, якщо це буде можливо. Останню умову можна забезпечити за допомогою переміщення в усі новостворені таблиці первинного ключа колишньої таблиці. Припустимо, в початковому вигляді таблиця Викладач мала такий вигляд:

Викладач (ТАБН, ПІБ, СТ, ЗВ, ККАФ)

Припустимо також, що виникла необхідність розбити цю вихідну таблицю на дві нові таблиці, ВиклДані і ВиклКафедра:

ВиклДані (ТАБН, ПІБ, СТ, ЗВ)

ВиклКафедра (ТАБН, ККАФ)

Користувачі і додатки зможуть як і раніше мати доступ до даних з використанням формату вихідної таблиці, якщо визначити уявлення Викладач, побудоване на базі природного з'єднання таблиць ВиклДані і ВиклКафедра, за допомогою з'єднувального стовпчика ТАБН:

CREATE VIEW Викладач

AS SELECT пд.ТАБН, ПІБ, СТ, ЗВ, ККАФ

FROM ВиклДані пд, ВиклКафедра пк

WHERE пд.ККАФ = ПК.ККАФ;

2. Актуальність

Зміни в будь-якій з таблиць бази даних, зазначених у визначальному запиті, негайно відображаються на вмісті уявлення.

3. Підвищення захищеності даних

Кожному користувачеві права доступу до даних в базі можуть бути надані виключно через обмежений набір представлень, що включають тільки ті підмножини даних, з якими користувачеві необхідно працювати. Подібний підхід дозволяє істотно посилити контроль за доступом окремих категорій користувачів до інформації в базі даних і контролювати такий доступ.

4. Зниження складності

Використання представлень дозволяє спростити структуру запитів, об'єднавши дані з декількох таблиць в єдину віртуальну таблицю. В результаті

багатотабличні запити перетворюються в прості запити до одного представлення.

5. Додаткові зручності

Створення представлень може забезпечувати користувачів додатковими зручностями - наприклад, дозволити їм працювати тільки з тією частиною даних, яка їм дійсно необхідна. В результаті можна домогтися максимального спрощення тієї моделі даних, з якою буде працювати кожен кінцевий користувач.

6. Можливість настройки

Представлення є зручним засобом настройки того образу бази даних, з яким буде працювати кожен з користувачів. В результаті одні й ті ж таблиці можуть бути пред'явлені різним користувачам в абсолютно різному вигляді.

7. Забезпечення цілісності даних

Якщо в операторі CREATE VIEW буде вказана конструкція WITH CHECK OPTION, то СУБД буде здійснювати контроль за тим, щоб у вихідні таблиці бази даних не був введений жоден з рядків, які задовольняють конструкції WHERE у визначальному запиті. Цей механізм гарантує цілісність даних в представленні.

Недоліки

1. Обмежені можливості поновлення

У деяких випадках уявлення не дозволяють вносити зміни в дані, що містяться в таблицях.

2. Структурні обмеження

Структура представлення встановлюється в момент його створення. Якщо запит, що визначається, представлений в формі SELECT * FROM, то символ * посилається на всі стовпці, існуючі в початковій таблиці на момент створення представлення. Якщо згодом у вихідну таблицю бази даних будуть додані нові стовпчики, то вони не з'являться в даному представленні до тих пір, поки це представлення не буде видалено і знову створено.

3. Зниження продуктивності

Використання уявлень пов'язано з певним зниженням продуктивності. В одних випадках вплив цього фактора буде абсолютно незначним, тоді як в інших воно може послужити джерелом істотних проблем. Наприклад, представлення, визначене за допомогою складного багатотабличного запиту, може вимагати значних витрат часу на обробку, оскільки при заміні представлення потрібно виконувати з'єднання таблиць щоразу, коли знадобиться доступ до даного представлення. Виконання заміни представлень пов'язано з використанням додаткових обчислювальних ресурсів.

Матеріалізація представлень

Розроблено альтернативний підхід до використання представлень, який передбачає збереження представлення в тимчасовій таблиці бази даних після

першого звернення до нього. В подальшому ця тимчасова таблиця застосовується для виконання запитів, заснованих на матеріалізованому представленні, тому запити виконуються набагато швидше, не вимагаючи повторного обчислення вихідного представлення. Завдяки такому підходу досягається значне підвищення продуктивності тих додатків, які характеризуються високою інтенсивністю виконання запитів і передбачають застосування складних запитів.

Матеріалізовані представлення широко застосовуються в таких сучасних додатках, як сховища даних, сервери реплікації, засоби візуалізації даних і мобільні системи. При використанні матеріалізованих представлень спрощуються також завдання контролю обмежень цілісності та оптимізації запитів. Складність реалізації цього підходу полягає в підтримці актуальності додатки при оновленні однієї або декількох базових таблиць. Процес оновлення представлення у відповідь на зміни даних відповідних таблиць називається супроводом представлень. Справа в тому, що в процесі супроводу представлення повинні враховуватися тільки ті зміни, які необхідні для забезпечення актуальності уявлення. Як приклад того, які при цьому виникають складності, розглянемо таке уявлення:

```
CREATE VIEW HCCСтудент (НС)
AS SELECT DISTINCT НС
FROM Студент
WHERE СРБАЛ = '3,8' AND РІКНАР > 1985;
```

Припустимо, що в результаті виконання цього представлення отримані дані, наведені в табл. 7.2.9. Якщо в таблицю Студент має бути вставлений рядок, в якому значення РІКНАР = 1985, то представлення має залишитися незмінним. А якщо в таблицю Студент має бути вставлений рядок (1, 'ТМ-22', 'Федот М.', 1982, 'Прага', 3,8), то дані цього рядка повинні з'явитися в матеріалізованому представленні. З іншого боку, при вставці рядка (12, 'ТМ-22', 'Федот М.', 1982, 'Прага', 3,8) в таблицю Студент матеріалізоване уявлення не повинно змінитися, оскільки в ньому вже є рядок зі значенням 12. Зверніть увагу, що у всіх трьох випадках рішення про те, чи повинна бути виконана вставка рядка матеріалізоване уявлення, може бути прийнято без звернення до самої таблиці Студент.

Таблиця 7.2.9. Дані для подання НС Студент

НС
12
19

Якщо тепер потрібно видалити з таблиці Студент знову введений рядок (1, 'ТМ-22', 'Федот М.', 1982, 'Прага', 3,8), то відповідний рядок довелося б також видалити і з матеріалізованого уявлення. Але якби з таблиці Студент потрібно було видалити новий рядок (12, 'ТМ-22', 'Федот М.', 1982, 'Прага', 3,8), то рядок, що відповідає номеру студента 12, не треба було б видаляти з матеріалізованого уявлення, оскільки в базовій таблиці продовжує перебувати рядок, яка відноситься студенту на прізвище Бабич Ю. в цих двох випадках для прийняття рішення про те, чи слід видалити або залишити рядок в матеріалізованому представленні, необхідно мати доступ до базової таблиці Студент.

Використання транзакцій

Стандарт ISO включає визначення моделі транзакцій, побудованої на використанні двох спеціальних операторів - COMMIT і ROLLBACK. Більшість комерційних реалізацій мови SQL (проте не всі) підтримує цю модель, яка вперше була реалізована в СУБД DB2 компанії IBM.

Транзакцією називається логічна одиниця роботи, що складається з одного або декількох операторів SQL, яка з точки зору відновлення даних буде розглядатися і оброблятися системою як єдина неподільна дія.

У стандарті вказується, що в мові SQL транзакція автоматично запускається будь-яким ініціалізуючим транзакцію оператором SQL, виконуваним користувачем або програмою (наприклад, SELECT, INSERT або UPDATE). Зміни, внесені в базу даних в ході виконання транзакції, не сприйматимуться будь-якими іншими виконуваними паралельно транзакціями до тих пір, поки ця транзакція не буде явним чином завершена. Завершення транзакції може бути виконано одним з наступних чотирьох способів:

1. Введення оператора COMMIT означає успішне завершення транзакції. Після його виконання внесені в базу даних зміни набувають постійного характеру. Після обробки оператора COMMIT введення будь-якого ініціюючого транзакцію оператора автоматично викличе запуск нової транзакції.

2. Введення оператора ROLLBACK означає відмову від завершення транзакції, в результаті чого виконується відкат всіх змін в базі даних, внесених при виконанні цієї транзакції. Після обробки оператора ROLLBACK введення будь-якого ініціюючого транзакцію оператора автоматично викличе запуск нової транзакції.

3. При впровадженні операторів SQL в текст програми успішне закінчення її роботи автоматично викличе завершення останньої запущеної програмою транзакції, навіть якщо оператор COMMIT для неї не був введений явно.

4. При впровадженні операторів SQL в текст програми аварійне закінчення її роботи автоматично викличе відкат останньої транзакції, запущеної цією програмою.

У мові SQL заборонено використання вкладених транзакцій.

Модель вкладених транзакцій розглядається як колекція взаємопов'язаних підзадач або субтранзакцій, кожна з яких також може складатися з будь-якої кількості субтранзакцій. У цій моделі вся транзакція розглядається як набір пов'язаних підзадач, званих **субтранзакціями**.

За допомогою оператора SET TRANSACTION користувачі можуть налаштовувати певні характеристики процесу обробки транзакцій. Основний формат цього оператора має наступний вигляд:

```
SET TRANSACTION
[READ ONLY \ READ WRITS] |
[ISOLATION LEVEL READ UNCOMMITTED READ COMMITTED |
REPEATABLE READ SERIALIZABLE]
```

Кваліфікатори READ ONLY і READ WRITE вказують, що в транзакціях допускається виконання тільки операцій читання або читання і запису. За умовчанням передбачається використання кваліфікатора READ WRITE (якщо тільки не обраний рівень ізоляції READ UNCOMMITTED). Ймовірно, багатьох збентежить той факт, що в режимі READ ONLY в транзакціях допускається видача операторів INSERT, UPDATE і DELETE для тимчасових таблиць (але тільки для тимчасових).

Показник рівня ізоляції визначає ту ступінь взаємодії з іншими транзакціями, яка допускається при виконанні транзакції. Відомості про обмеження у відношенні серіалізації (визначення порядку проходження) результатів виконання транзакцій для кожного з існуючих рівнів ізоляції наведені в табл. 7.2.10.

Таблиця 7.2.10. Обмеження на серіалізацію результатів виконання транзакції, які визначаються рівнем ізоляції

Рівень ізоляції	Читання сміття	Неповторюваність читання	Існування фантомних значень
READ UNCOMMITTED	Так	Так	Так
READ COMMITTED	Ні	Ні	Так
REPEATABLE READ	Ні	Ні	Так
SERIALISABLE	Ні	Ні	Ні

Повна безпека гарантується тільки рівнем ізоляції SERIALIZABLE, який передбачає генерацію тимчасових графіків серіалізації. Всі інші рівні ізоляції вимагають, щоб СУБД надавала деякий механізм, який програмісти могли б використовувати для забезпечення серіалізації даних.

Негайні і відкладені обмеження підтримки цілісності даних

У деяких ситуаціях не потрібно негайно виконувати перевірку встановлених вимог підтримки цілісності даних після завершення обробки кожного окремого оператора SQL, але необхідно забезпечити цю перевірку при завершенні кожної транзакції. На початку виконання кожної з транзакцій може бути встановлений режим виконання перевірки обмежень INITIALLY IMMEDIATE або INITIALLY DEFERRED. У першому випадку можна також вказати, чи допускається змінювати встановлений режим згодом, для чого призначений кваліфікатор [NO] DEFERRABLE. За умовчанням передбачається

використання режиму **INITIALLY IMMEDIATE**. Оператор **SET CONSTRAINTS** використовується для установки режиму виконання зазначених перевірок в поточної транзакції. Цей оператор має наступний формат:

SET CONSTRAINTS

{ALL і constraintName [...]} {DEFERRED | IMMEDIATE}

Управління доступом до даних

Кожна СУБД повинна надавати механізм, який гарантує, що доступ до бази даних зможуть отримати тільки ті користувачі, які мають відповідний дозвіл. Мова SQL включає оператори **GRANT** і **REVOKE**, призначені для організації захисту таблиць в базі даних. Застосовуваний механізм захисту побудований на використанні ідентифікаторів користувачів, що надаються їм прав володіння і привілеїв.

Ідентифікатори користувачів і права володіння

Ідентифікатором користувача називається звичайний ідентифікатор мови SQL, що використовується для позначення деякого користувача бази даних.

Кожному користувачеві бази даних повинен бути призначений власний ідентифікатор, який присвоюється адміністратором бази даних (АБД). З очевидних міркувань захисту даних ідентифікатор користувача, як правило, захищається паролем. Кожен виконуваний СУБД оператор SQL виконується від імені будь-якого користувача. Ідентифікатор користувача застосовується для визначення того, на які об'єкти бази даних може посылатися користувач і які операції з цими об'єктами він має право виконувати. Кожен створений в середовищі SQL об'єкт має свого власника. Власник задається ідентифікатором користувача, визначеним у конструкції **AUTHORIZATION** тієї схеми, якій цей об'єкт належить. Спочатку тільки власник об'єкта знає про існування даного об'єкта і має право виконувати з цим об'єктом будь-які операції.

Привілеї

Привілеями називають визначення дій, які користувач має право виконувати по відношенню до цієї таблиці бази даних або представлення.

У стандарті ISO визначається наступний набір привілеїв:

- **SELECT** - право вибирати дані з таблиці;
- **INSERT** - право вставляти в таблицю нові рядки;
- **UPDATE** - право змінювати дані в таблиці;
- **DELETE** - право видаляти рядки з таблиці;
- **REFERENCES** - право посилалися на стовпці вказаної таблиці в описах вимог підтримки цілісності даних;
- **USAGE** - право використовувати домени, перевірки, набори символів і трансляції.

Привілеї **INSERT** і **UPDATE** можуть обмежуватися лише окремими стовпцями таблиці; в цьому випадку користувач може модифікувати значення зазначених стовпців, але не змінювати значення інших стовпців таблиці. Аналогічним чином, привілей **REFERENCES** може поширюватися тільки на окремі стовпці таблиці, що дозволить використовувати їх імена в формулюваннях вимог захисту цілісності даних (наприклад, в конструкціях **CHECK** і **FOREIGN KEY**), що входять в визначення інших таблиць, тоді як застосування для подібних цілей інших стовпців буде заборонено. Коли користувач за допомогою оператора **CREATE TABLE** створює нову таблицю, він автоматично стає її власником і отримує по відношенню до неї повний набір привілеїв. Решта користувачів спочатку не мають яких-небудь привілеїв щодо новоствореної таблиці. Щоб забезпечити доступ до неї, власник повинен явним чином надати їм необхідні права, для чого використовується оператор **GRANT**. Коли користувач створює уявлення за допомогою оператора **CREATE VIEW**, він автоматично стає власником цього подання, однак зовсім не обов'язково отримує по відношенню до нього повний набір прав. Для створення уявлення користувачеві досить мати привілей **SELECT** для всіх вхідних в дане представлення таблиць і привілей **REFERENCES** для всіх стовпців, що згадуються в визначенні цього подання. Але привілеї **INSERT**, **UPDATE** і **DELETE** щодо створеного уявлення користувач отримає тільки в тому випадку, якщо він має відповідні привілеї щодо всіх використовуваних в представленні таблиць.

Надання привілеїв іншим користувачам (оператор GRANT)

Оператор **GRANT** використовується для надання зазначеним користувачам привілеїв щодо названих об'єктів бази даних. Цей оператор зазвичай застосовується власником таблиці з метою надання доступу до неї іншим користувачам. Оператор **GRANT** має наступний формат:

```
GRANT {PrivilegeList \ ALL PRIVILEGES}
ON ObjectName
TO {AuthorizationIdList \ PUBLIC}
[WITH GRANT OPTION]
```

Параметр **PrivilegeList** є список, що складається з однієї або більше привілеїв, розділених комами:

```
SELECT
DELETE
INSERT [(columnName [, ... ] ) ]
UPDATE [(columnName [, ... ] ) ]
REFERENCES [(columnName [, ... ])]
USAGE
```

Крім того, для спрощення в операторі GRANT можна вказати ключове слово ALL PRIVILEGES, що дозволить надати зазначеному користувачеві всі шість існуючих привілеїв без необхідності їх перерахування. У цьому операторі можна також вказати ключове слово PUBLIC, що означає надання доступу зазначеного типу не тільки всім існуючим користувачам, але і всім тим користувачам, які будуть визначені в базі даних згодом. Параметр ObjectName може являти собою ім'я таблиці бази даних, уявлення, домену, набору символів, перевірки або трансляції.

Конструкція WITH GRANT OPTION дозволяє всім зазначеним у списку параметра AuthorizationIdList користувачам передавати іншим користувачам всі надані їм у відношенні зазначеного об'єкта привілеї. Якщо ці користувачі також передадуть власні повноваження іншим користувачам із зазначенням конструкції WITH GRANT OPTION, то останні, в свою чергу, також отримають право передавати свої повноваження іншим користувачам. Якщо ця конструкція не буде вказана, одержувач привілеїв не зможе передати свої права іншим користувачам. Таким чином, власник об'єкта може чітко контролювати, хто отримав право доступу до об'єкта і які повноваження йому надані.

Приклад 7.2.8. Надання всіх привілеїв

Надайте викладачеві, який має вчене звання Професор всі привілеї доступу до таблиці Викладач.

```
GRANT ALL PRIVILEGES
ON Викладач
TO Професор WITH GRANT OPTION;
```

В результаті користувач з ідентифікатором Професор отримав право вибирати дані з таблиці Викладач, а також вставляти, оновлювати або видаляти рядки з неї. Крім того, користувач Професор тепер може посилатися на таблицю Викладач і все її стовпці в будь-якій таблиці, створюваної ним згодом. У наведеному вище операторі присутня конструкція WITH GRANT OPTION, тому користувач Викладач зможе передавати отримані ним привілеї будь-яким іншим користувачам на свій розсуд.

Приклад 7.2.9. Надання тільки деяких привілеїв

Надайте користувачам Професор і Доцент привілеї SELECT і UPDATE на стовпець АДРЕСА таблиці Студент.

GRANT SELECT, UPDATE (АДРЕСА)

ON Студент

TO Професор, Доцент;

Оскільки в наведеному вище операторі конструкція WITH GRANT OPTION опущена, користувачі Професор і Доцент не зможуть передати отримані ними привілеї іншим користувачам.

Приклад 7.2.10. Надання певних привілеїв всім користувачам типу PUBLIC

Надайте всім користувачам бази даних привілей PUBLIC для таблиці Студент.

GRANT SELECT

ON Студент

TO PUBLIC;

Використання в наведеному вище операторі ключового слова PUBLIC означає, що всі користувачі (як вже існуючі, так і ті, які будуть створені в базі даних згодом) отримують право вибирати всі дані, наявні в таблиці Студент. Зверніть увагу на те, що в даному випадку немає сенсу використовувати конструкцію WITH GRANT OPTION, оскільки зазначені права отримують всі користувачі бази даних і надавати їх більше просто нікому.

Скасування наданих користувачам привілеїв (оператор REVOKE)

У мові SQL для скасування наданих користувачам за допомогою оператора GRANT привілеїв використовується оператор REVOKE. За допомогою цього оператора можуть бути скасовані всі або деякі з привілеїв, наданих зазначеному користувачеві раніше. Оператор REVOKE має такий вигляд:

```
REVOKE [GRANT OPTION FOR] (PrivilegeList \ ALL PRIVILEGES)
ON ObjectName
FROM {AuthorizationIdList \ PUBLIC}
[RESTRICT \ CASCADE]
```

Ключове слово ALL PRIVILEGES означає, що для зазначеного користувача скасовуються всі привілеї, надані йому раніше тим користувачем, який ввів цей оператор. Необов'язкова конструкція GRANT OPTION FOR дозволяє для всіх привілеїв, переданих в вихідному операторі GRANT конструкції WITH GRANT OPTION, скасовувати можливість їх передачі незалежно від самих цих привілеїв.

Призначення ключових слів RESTRICT і CASCADE аналогічно тому, яке вони мають в операторі DROP TABLE.

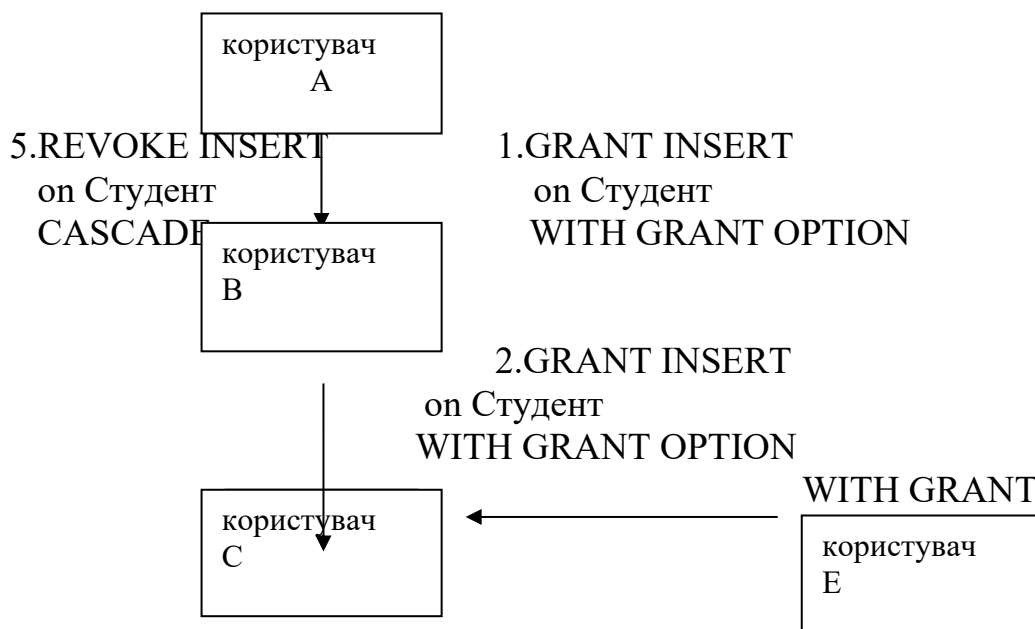
Нагадуємо.

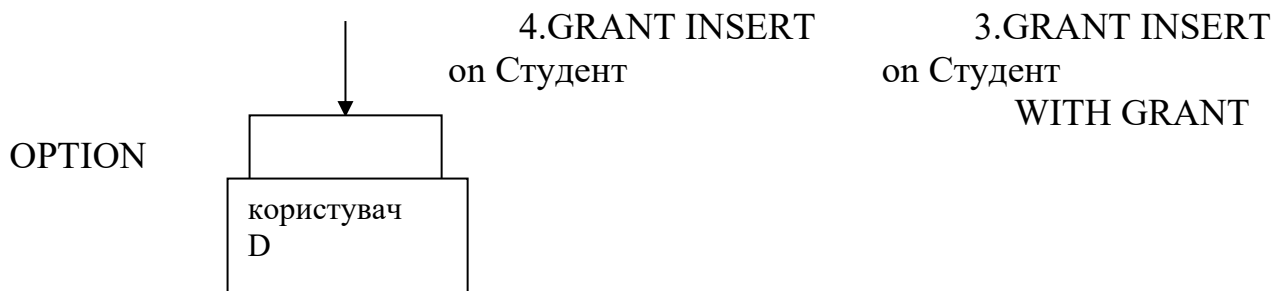
- **RESTRICT.** Операція DROP відкидається, якщо на даний стовпець є посилання в іншому об'єкті бази даних (наприклад, у визначенні подання). Це значення опції передбачено за замовчуванням.

- **CASCADE.** Рятувальна операція DROP триває в будь-якому випадку і посилання на стовпець автоматично видаляються з будь-яких об'єктів бази даних, де вони є. Ця операція виконується каскадно, тому якщо стовпець видаляється з об'єкту, що містить посилання, то в базі даних виконується перевірка того, чи є посилання на цей стовпець в будь-якому іншому об'єкті, такі посилання знищуються і в цьому об'єкті, і т.д.

Оскільки для створення деяких об'єктів необхідна наявність привілеїв, в результаті видалення привілеї користувач може втратити право, на підставі якого ним було створено той чи інший об'єкт (подібні об'єкти, позбавлені власника, прийнято називати закинутими). Виконання оператора REVOKE буде скасовано, якщо в результаті можуть з'явитися занедбані об'єкти (наприклад, уявлення), якщо тільки в ньому не вказано ключове слово CASCADE. Якщо в операторі присутня ключове слово CASCADE, то для будь-яких покинутих об'єктів (уявлень, доменів, обмежень або перевірок), що виникають при виконанні вихідного оператора REVOKE, будуть автоматично видані оператори DROP.

Привілеї, які були надані вказаному користувачу іншими користувачами, не можуть бути порушені даними оператором REVOKE. Отже, якщо інший користувач також надав даному користувачеві відкликає привілей, то право доступу до відповідної таблиці у зазначеного користувача збережеться. Наприклад, на малюнку показано, що користувач А надав користувачеві В привілей INSERT для таблиці Студент, причому із зазначенням конструкції WITH GRANT OPTION (крок 1). Користувач В передав цей привілей користувачеві С (крок 2). Потім користувач З отримав цю ж привілей від користувача Е (крок 3). Далі користувач З надав згадану привілей користувачеві D (крок 4). Коли користувач А скасовує привілей INSERT для користувача В (крок 5), цей привілей не може бути скасована і для користувача С, оскільки він вже отримав її раніше від користувача Е. Якби користувач Е не надав даної привілеї користувачеві С, то відгук привілеї користувача В мав би наслідком каскадне видалення привілеїв для користувачів С і D.





Мал. 7.2.1. Приклад результатів виконання оператора REVOKE

Приклад 7.2.11. Скасування певних привілеїв, наданих всім користувачам.

Скасуйте привілей SELECT, яка була надана всім користувачам для таблиці Студент.

REVOKE SELECT

ON Студент

FROM PUBLIC;

Приклад 7.2.12. Скасування певних привілеїв, наданих зазначеному користувачеві

Скасуйте всі привілеї, які були надані користувачеві з ідентифікатором Доцент для таблиці Студент.

REVOKE ALL PRIVILEGES

ON Студент

FROM Доцент;

Цей оператор за результатами виконання еквівалентний оператору REVOKE SELECT, оскільки користувачеві Доцент раніше була надана тільки цей привілей.

Додаток

Група		
Номер групи	К-сть студентів в групі	Середній бал
ТЗ-31	1	3,5
ТЗ-32	1	4,1
ТІ-31	1	3,8
ТІ-32	1	4,1
ТМ-31	1	4,5
ТМ-33	1	4

Вивчення					
Номер групи	Код предмета	Таб. номер викладача	Вид заняття	Години	Середній бал по предмету
ТЗ-31	01	2	лаба	120	3,4
ТЗ-32	03	33	лекція	30	4,8
ТІ-31	01	44	практика	120	4
ТІ-32	04	11	практика	150	3
ТМ-31	06	55	практика	15	3
ТМ-33	03	44	лаба	200	4,4

Кафедра				
Код	Назва	Телефон	ПІБ завідувача кафедри	фото завідувача
01	математики	123-45-67	Піфагор А.М.	
02	креслення	891-23-45	Начералкін Р.Н.	
03	фізики	678-91-23	Фуко Н.І.	
04	Укр. яз.	456-78-91	Шевченко Т.Г.	
05	хімії	234-56-78	Пробірін А.М.	
06	Фіз-ри	912-34-56	Клочко Я.П.	

Предмет						
Код предмета	Назва предмета	Кількість годин	Лекції	Практика	Кількість семестрів	Програма
01	алгебра	120	65	55	2	
02	геометрія	80	60	20	1	
03	начерталки	200	120	80	4	
04	Англ. яз	150	75	75	3	
05	Інформатика	100	50	50	2	
06	фізика	15	13	2	1	

Викладач				
Таб номер	ПІБ викладача	Вчений ступінь	Вчене звання	Код кафедри
11	Глоба Л.С	канд. техн. наук	професор	01
22	Максимов В.В.	канд. техн. наук	доцент	02
33	Вульпе О.А.	д-р техн. наук	професор	03
44	Плотников О.О.	канд. фил. наук	асистент	01
55	Дроздович Н.Ю.	канд. фил. наук	доцент	05
66	Лебедев О.М.	д-р техн. наук	професор	02

Студент					
Номер студента в групі	Група	ПІБ	рік народження	Адреса	середній бал
12	ТМ-31	Бабич Ю.	1986	с. Жмеренка	3,8
15	ТЗ-32	Чучмий П.	1985	Севастополь	3,1
19	ТЗ-31	Самойленко Ю.	1986	Одеса	3,8
20	ТІ-32	Святненко Ю.	1986	Львів	3,6
21	ТМ-33	Терпило Д.	1986	Київ	3,5
23	ТІ-31	Мурзик К.	1986	Херсон	3,2

успішність					
Номер групи	Номер студента	Код предмета	Таб номер викладача	Вид заняття	Оцінка
ТЗ-31	19	01	22	лаба	5
ТЗ-32	15	03	33	лекція	5
ТІ-31	23	01	44	практика	4
ТІ-32	20	04	11	практика	4
ТМ-31	12	06	55	практика	3
ТМ-33	21	03	44	лаба	2

Питання:

- 7.2.1. Опишіть функціональні можливості і призначення засобів підтримки цілісності даних.
- 7.2.2. Перерахуйте переваги і недоліки, властиві уявленням.
- 7.2.3. Опишіть хід виконання процесу заміни уявлення.
- 7.2.4. Яким вимогам має задовольняти подання, щоб воно могло бути оновлюваних?
- 7.2.5. Що таке матеріалізоване уявлення і в чому полягають переваги застосування матеріалізованого уявлення замість використання процесу заміни уявлення?
- 7.2.6. Поясніть, як працює механізм контролю за доступом до даних мови SQL.

7.3 Збережені процедури і функції. Тригери.

У цьому розділі буде представлений стандарт, званий *Persistent, Stored Modules* (SQL / PSM, або PSM-96, або просто PSM) - «**постійно збережені модулі**». Кожна комерційна СУБД пропонує ті чи інші способи написання і збереження в схемі бази даних деяких функцій і / або процедур, до яких допустимо звертатися з SQL-запитів або інших SQL-виразів. Подібні фрагменти коду створюються з використанням простої мови загального призначення, що дозволяє описувати найрізноманітніші обчислювальні конструкції, які не можна виразити засобами мови SQL як такого.

Спираючись на стандарт PSM, програміст створює модулі (modules), що містять набори визначень процедур і функцій, оголошень тимчасових таблиць і інших елементів схеми бази даних.

Створення збережених процедур і функцій

Оголошення процедури, що в своїй базовій формі виглядає так:

```
CREATE PROCEDURE <ім'я> (<параметри>)
```

```
<Оголошення локальних змінних>
```

```
<Тіло процедури>;
```

Подібна конструкція досить знайома з багатьох мов програмування; вона складається з пари службових слів CREATE PROCEDURE, імені процедури, списку параметрів, укладеного в круглі дужки, ряду оголошень локальних змінних і фрагмента коду, що виконується, який, власне, і є «ядром» процедури. Збережена функція визначається майже аналогічно; відмінності пов'язані з використанням замість слова PROCEDURE службового слова FUNCTION і наявністю обов'язкового пропозиції, задає тип повертається функцією значення. Іншими словами, конструкцію оголошення збереженої функції можна представити в такий спосіб:

```
CREATE FUNCTION <ім'я> (<параметри>) RETURNS <тип>
```

```
<Оголошення локальних змінних>
```

```
<Тіло функції>;
```

Параметрами процедури є трійки виду «режим-ідентифікатор-тип». Для позначення параметра слід вказати не тільки його ім'я-ідентифікатор і тип, як це прийнято в традиційних мовах програмування, але і додатковий кваліфікатор, що визначає «режим» (mode) використання параметра - IN («вхідний»), OUT («вихідний») і INOUT («вихідний / вихідний»). Кваліфікатор IN, передбачений за замовчуванням, дозволено не здавати.

Параметри збереженої функції, однак, можуть використовуватися тільки в режимі IN. Іншими словами, стандарт PSM забороняє збереженої функції повертати значення яким би то не було способом, відмінним від традиційного, що описується командою RETURN. Визначаючи параметри функцій, кваліфікатор IN будемо опускати; в оголошеннях процедур, однак, його все-таки корисно задавати.

Приклад 7.3.1. Хоча ще не було знайомства з внутрішнім «пристроєм» топової збереженої процедури або функції, одна з категорій конструкцій, які допустимо застосовувати в їх тілі, аж ніяк не здивує: мова йде про вирази SQL. Обмеження, яким має відповідати такий вислів, аналогічні тим, що прийняті для «впроваджуваних» SQL-виразів (запитів, тригерів, процедур і т.д.); тут дозволено використовувати команди вибору єдиного кортежу і конструкції на основі курсорів. Нижче наведено приклад процедури, яка в якості параметрів приймає два адреси, «старий» і «новий»,

знаходить все кортежі відносини СТУДЕНТ з компонентами Адреса, що містять «старий» адреса, і замінює їх новим значенням адреси.

- 1) CREATE PROCEDURE Move (
- 2) IN СтарАдрес VARCHAR (255),
- 3) IN НовАдрес VARCHAR (255),
- 4))
- 5) UPDATE СТУДЕНТ
- 6) SET Адрес = НовАдрес
- 7) WHERE Адрес = СтарАдрес;

Рядок 1 містить заголовок оголошення процедури і її ім'я, Move. Рядки 2, 3 представляють опис параметрів: обидва параметри є вхідними і відносяться до рядкового типу змінної довжини 255. Ці визначення узгоджуються з оголошенням атрибута Адреса в схемі відносини СТУДЕНТ. У рядках 4 - 6 приведена звичайна команда UPDATE. СЛІД Звернути увагу на те, що ідентифікатори параметрів використовуються так само, як якби замість них задавалися константи. На відміну від загальних змінних, які формують інтерфейс між базовим кодом і «впровадженими» SQL-командами і позначаються префіксом-двокрапкою, параметри і локальні змінні процедур і функцій в позначенні за допомогою двокрапки не потребують.

Прості форми виразів

Виклик процедури

CALL <ім'я процедури> (<список аргументів>);

Службове слово CALL має назви процедури і списком аргументів, укладеними в круглі дужки (як і в багатьох інших мовах). Подібним викликом, проте, можна користуватися в різних контекстах:

- всередині програми, написаної на базовому мовою; прикладом може служити конструкція

EXEC SQL CALL Foo (: x, 3);

- всередині іншої процедури або функції;
- як інтерпретується командою базового інтерфейсу SQL; наприклад, виклик CALL Foo (1, 3);

змушує систему звернутися до збережених процедурі FOO, передавши їй в якості аргументів значення 1 і 3.

Зауважимо, що за допомогою CALL можна викликати тільки процедури, але не функції. Виклик функції в PSM виконується так само, як і в С - ім'я функції і список аргументів зазвичай є частиною більш великого вираження.

Повернення значення

RETURN <вираз>;

Цю команду дозволено використовувати тільки в тілі зберігається функції. Вираз, що служить аргументом команди, обчислюється, і отриманий результат трактується як значення, що підлягає поверненню до код-ініціатор, звідки функція була викликана. На відміну від звичайних мов програмування, команда RETURN в PSM завершує виконання коду функції. Управління передається наступним виразом, якщо таке існує, і тому в процесі роботи функції повертається значення може неодноразово змінюватися.

Оголошення локальної змінної

Вираз виду

DECLARE <ім'я> <тип>;

Представляє оголошення змінної із заданими ім'ям і типом. Змінна є локальною - це значить, що система не зберігає інформацію про неї по завершенні виконання процедури або функції. Вирази об'влення локальних змінних повинні передувати будь-яким виконуваним інструкціям в тілі процедури або функції.

оператор присвоювання

SET <змінна> = <вираз>;

Оператор присвоювання в PSM дуже схожий з аналогічними конструкціями в інших мовах (незвичайним, мабуть, є лише наявність службового слова SET). Спочатку система обчислює значення виразу в правій частині оператора, а потім отриманий результат стає значенням змінної, зазначеної в лівій частині. До числа допустимих виразів правій частині відносяться NULL, а також запит виду «select - from - where», якщо такий повертає єдине значення

Блоки коду

Список виразів, розміщений між парою службових слів BEGIN і END, інтерпретується системою як єдине вираження, яке може використовуватися в будь-якому місці коду, де допускається присутність окремого виразу. Зокрема, тіло процедури або функції також воспрінімається системою як єдиний вираз.

Мітки

Мітка складається з імені і суфікса-двокрапки. Нижче буде розглянуто ситуацію, в якій застосування міток необхідне і виправдано.

Розгалуження

Перейдемо до більш складним виразами PSM і для початку розглянемо умовний вираз IF, яке відрізняється від аналогічних конструкцій інших мов (таких як C) в двох аспектах:

1) вираз завершується пропозицією END IF;

2) вираз IF, вкладені в пропозиції ELSE інших виразів IF, вводяться за допомогою службового слова ELSEIF.

Синтаксична форма умовного виразу IF PSM приведена нижче. У якості «умов» можуть задаватися будь-які вирази (подібні таким, як в реченні WHERE SQL-запиту), які повертають значення булева типу. Кожен «список виразів» складається з ряду виразів, завершує символом крапки з комою (використовувати блоки коду, обрамлені службовими словами BEGIN і END, необов'язково). Пропозиції ELSEIF і ELSE також не є обов'язковими, тобто прийнятною вважається і конструкція виду IF ... THEN ... END ... IF.

IF <умова> THEN

<Список виразів>

ELSEIF <умова> THEN

<Список виразів>

ELSEIF

...

ELSE <список виразів>

END IF;

Приклад 7.3.2. Нехай необхідно створити збережену процедуру, яка приймає в якості параметрів значення року «у» і назва кафедри «к» і повертає логічне значення

рівне TRUE в тому і тільки в тому випадку, якщо кафедра «k» протягом року «y» опублікувала по щонайменше один науковий проект або не опублікувала наукові проекти зовсім.

- 1) CREATE FUNCTION PROEKT (y INT, f CHAR (30)) RETURNS
BOOLEAN
- 2) IF NOT EXISTS (
- 3) SELECT * FROM ФАКУЛЬТЕТ WHERE Рік = y AND
НазвКафедри = k)
- 4) THEN RETURN TRUE;
- 5) ELSEIF 1 <=
- 6) (SELECT COUNT (*) FROM ФАКУЛЬТЕТ WHERE Рік = y
AND НазвКафедри = k AND НаучнийПроект)
- 7) THEN RETURN TRUE;
- 8) ELSE RETURN FALSE;
- 9) END IF;

Рядок 1 містить заголовок функції зі списком параметрів і типом повертаючого значення. Кваліфікатори «режиму» використання параметрів не задаються, оскільки, в збережених функціях завжди маєтись на увазі режим IN. У рядках 2, 3 перевіряється умова «кафедра k протягом року y не випустила жодної публікації»; якщо умова справедливо, в рядку 4 в якості значення, що підлягає поверненню, встановлюється TRUE. Нагадаємо, що інструкція RETURN сама по собі не завершує виконання функції - в даному випадку управління передається рядку 9, а оскільки та є останньою, робота функції припиняється природним чином.

Якщо відносно ФАКУЛЬТЕТ все-таки є якась інформація про публікації, виданих кафедрою k в році y, в рядках 5, 6 перевіряється, чи існує серед цих публікацій хоча б один науковий проект. При позитивній відповіді в рядку 7 задається значення, що повертається, рівне TRUE. Якщо ж наукових проектів серед публікацій кафедри k за рік y немає в рядку 8 встановлюється значення, що повертається, рівне FALSE.

Запити

Існує кілька способів використання запитів виду «select - from - where» в коді PSM; всі вони перераховані нижче.

1. Підзапити дозволено застосовувати в контексті умов (або, взагалі кажучи, в будь-якому «місці», де це дозволено з точки зору правил SQL). Зразки підзапитів можна бачити, наприклад, в рядках 3 і 6 прикладу 7.3.2.

2. Запити, які повертають єдине значення, можуть бути використані в правих частинах операторів присвоювання.

3. У PSM знаходять застосування і команди вибору єдиного кортежу. Нагадаємо, що подібна команда SELECT містить пропозицію INTO, що перераховує імена змінних, яким повинні бути присвоєні значення відповідних компонентів обраного кортежу. У якості змінних пропозиції INTO в PSM можна використовувати локальні змінні і параметри (останні - тільки в процедурах).

4. У тілі збережених процедур і функцій допустимо використовувати курсори. Формати оголошення курсору і команд OPEN, FETCH і CLOSE аналогічні колишнім, але з урахуванням кількох винятків:

- не ставить префікс EXEC SQL;
- при посиланні на змінні не використовується двокрапка.

цикли LOOP

Базова конструкція циклу в PSM виглядає так:

LOOP

<Список виразів>

RND LOOP;

Часто заголовок циклу LOOP постачають міткою, що дає можливість перервати виконання циклу командою

LEAVE <мітка LOOP>;

Якщо цикл використовується для послідовного вилучення (FETCH) кортежів засобами курсора, переривати цикл зазвичай необхідно в ситуації, коли кортежі, призначене зчитуванню, відсутні. У подібних випадках корисне застосування знаходить конструкція умовного найменування (condition name) для значення змінної SQLSTATE (нехай код умови «кортеж не знайдений» дорівнює '02000'):

DECLARE Not_Found CONDITION FOR SQLSTATE '02000';

Взагалі кажучи, можна визначити умовне найменування для будь-якого значення змінної SQLSTATE, якщо скористатися командою наступного формату:

DECLARE <найменування> CONDITION FOR SQLSTATE <значення>;

Покажемо приклад, який ілюструє особливості використання циклів і курсорів в збережених процедурах і функціях.

Приклад 7.3.4. Нижче наведено текст процедури, що, яка приймає в якості «вхідного» параметра рядок pr назви предмета і повертає

- 1) CREATE PROCEDURE MeanVar(
2) IN pr CHAR (30),
3) OUT mean REAL,
4) OUT variance REAL
5))
6) DECLARE Not_Found CONDITION FOR SQLSTATE '02000';
7) DECLARE ParaCursor CURSOR FOR
8) SELECT length FROM ПРЕДМЕТ WHERE НазвПредмета = pr;
9) DECLARE newLength INTEGER;
10) DECLARE paraCount INTEGER;
11) BEGIN
12) SET mean = 0.0;
13) SET variance = 0.0;
14) SET paraCount = 0;
15) OPEN ParaCursor;
16) paraLoop: LOOP
17) FETCH ParaCursor INTO newLength;
18) IF Not_Found THEN LEAVE paraLoop END IF;
19) SET paraCount = SET paraCount + 1;

```

17)      SET mean = mean + newLength;
18)      SET variance = variance + newLength * newLength;
19)      END LOOP;
20)      SET mean = mean / paraCount;
21)      SET variance = variance / paraCount - mean * mean;
22)      CLOSE ParaCursor;
      END;

```

Цикли FOR, WHILE и REPEAT

Стандарт PSM пропонує можливість використання циклів і інших видів. Цикл FOR, наприклад, знаходить єдине застосування - для «переміщення» в межах курсору. Синтаксична форма циклу FOR така:

```

FOR <ім'я циклу> AS <ім'я курсору> CURSOR FOR
  <Запит>
DO
  <Список виразів>
END FOR;

```

Зазначена конструкція не тільки містить оголошення курсору, але і робить багато «чорнової» роботи, пов'язаної з відкриттям (OPEN) і закриттям (CLOSE) курсора, а також з операціями вилучення (FETCH) кортежів і перевіркою умов їх завершення. Інтерпретатор PSM зберігає для локальних змінних, використовуваних їм для зберігання вмісту компонентів зчитаних кортежів, ті ж імена і типи, з якими оголошені відповідні атрибути відносини, службовця результатом запиту.

Приклад 7.3.5. Замінімо цикл LOOP в прикладі процедури, показаної в прикладі 7.3.4, конструкцією циклу FOR.

```

1) CREATE PROCEDURE MeanVar (
2) IN pr CHAR (30),
3) OUT mean REAL,
4) OUT variance REAL
5) )
5) DECLARE paraCount INTEGER;
   BEGIN
6) SET mean = 0,0;
7) SET variance = 0,0;
8) SET paraCount = 0;
9) FOR paraLoop AS ParaCursor CURSOR FOR
      SELECT length FROM ПРЕДМЕТ WHERE
      НазвПредмета = pr;
10) DO
11) SET paraCount = SET paraCount + 1;
12) SET mean = mean + newLength;
13) SET variance = variance + length * length;
14) END FOR;
15) SET mean = mean / paraCount;
16) SET variance = variance / paraCount - mean * mean;
   END;

```

Багато лишилося незмінним: наприклад, збереглися заголовки процедури в рядках 1 - 4 і визначення локальної змінної paraCount (див. Рядок 5).

Тепер, однак, в секції оголошень не потрібно визначати курсор і умовне найменування Not_Found. Рядки 6 - 8 містять колишні інструкції ініціалізації локальних змінних. Рядок 9 представляє заголовок циклу FOR з оголошенням того ж курсора ParaCursor. Рядки 11 - 13 утворюють тіло циклу. Зверніть увагу, що в рядках 12, 13 для посилання на значення тривалості пари, лічений з витягнутого курсора, використовується ім'я атрибута length, а не newLength, як раніше, - локальна змінна newLength в цій версії процедури не використовується. У рядках 15, 16, як і в попередній версії процедури, обчислюються підсумкові значення середнього арифметичного і дисперсії.

У стандарті PSM передбачені форми циклів WHILE і REPEAT, за змістом нагадування циклічні конструкції мови С «while» і «do-while» відповідно. Синтаксис циклу WHILE виглядає так:

```
WHILE <умова> DO
<Список виразів>
END WHILE;
а циклу REPEAT - так:
REPEAT
<Список виразів>
UNTIL <умова>
END REPEAT;
```

Між іншим, якщо необхідно забезпечити міткою будь-який з циклів - LOOP, FOR, WHILE або REPEAT, - мітку можна розміщувати і безпосередньо після відповідної команди END (скажімо, END LOOP). Подібний стиль використання міток дозволяє швидко визначати, де завершується той чи інший цикл; крім того, інтерпретатор PSM отримує можливість виявляти деякі синтаксичні помилки, обумовлені відсутністю пропозицій END.

Винятки

Система повідомляє про виняткові ситуації, або винятки (exception), зустріла в процесі виконання SQL-команд, привласнюючи змінної SQLSTATE відповідні символічні представлення послідовностей з п'яти десяткових цифр (код '00000' означає, що помилок немає). Один із прикладів кодів помилок вже був представлений: '02000' - «кортеж не знайдено». Як інший приклад можна привести код '21000', який свідчить про те, що команда вибору єдиного кортежу повернула більше одного кортежу.

Стандарт PSM передбачає можливість оголошення спеціального фрагмента коду, званого обробником винятків (exception handler), який викликається всякий раз, коли будь-яке значення зі списку заданих кодів помилок присвоюється системою змінної SQLSTATE в процесі виконання команд поточного блоку коду. Кожен обробник виключень асоціюється з певним блоком коду виду BEGIN ... END (а саме з тим, якому належить сам обробник) і обмежує своє «вплив» тільки вмістом цього блоку.

Компонентами конструкції обробника винятків служать:

- список умов винятків, провокують виклик обробника;

- фрагмент коду, що підлягає виконанню, якщо згенеровано одне з вказаних винятків;
- одне із службових слів, що вказують на те, куди слід передати управління по завершенні виконання коду обробника.

Вираз визначення обробника виключення можна описати таким чином

DECLARE <куди йти> HANDLER FOR <хто винен>

<Що робити>

Альтернативні варіанти відповіді на питання «куди йти» такі:

- CONTINUE - після виконання виразу «що робити» перейти до чергового висловом поточного блоку коду BEGIN ... END;
- EXIT - по завершенні вираження «що робити» покинути поточний блок коли і передати управління висловом, наступного за блоком;
- UNDO - варіант, аналогічний EXIT, за «винятком» того, що все через трансформаційних змін, які торкнулися вміст бази даних і локальних змінних в період виконання команд поточного блоку коду до моменту виникнення виняткової ситуації, підлягають скасуванню, ніби ці команди не виконувалися зовсім.

Секція «хто винен» являє собою список умов винятків, розділених комами. В якості умов частіше використовуються умовні найменування, подібне Not_Found.

Приклад 7.3.6. Розглянемо функцію, яка повинна приймати як параметр рядок найменування предмета і дату здачі заліку з даного предмету. Якщо з усіх предметів передбачений іспит і заліків немає або помилково призначене кілька заліків з предмета, функція зобов'язана повернути NULL.

```

1) CREATE FUNCTION GetData (pr VARCHAR (30)) RETURNS
INTEGER
2) DECLARE Not_Found CONDITION FOR SQLSTATE '02000';
3) DECLARE Too_Many CONDITION FOR SQLSTATE '21000';
   BEGIN
4) DECLARE EXIT HANDLER FOR Not_Found, Too_Many
5) RETURN NULL;
6) RETURN (SELECT Дата FROM ПРЕДМЕТ WHERE
           НазвПредмета = pr);
   END;
```

Рядки 2, 3 містять оголошення умовних найменувань кодів винятків; взагалом, їх можна було і не оголошувати - тоді в рядку довелося б перерахувати коди безпосередньо. Рядки 4 - 6 утворюють блок виразів, першим з яких оголошений обробник, що реагує на дві виняткові ситуації - «кортеж не знайдено» ('02000') і «занадто багато кортежів» ('21000'). Секція «дії» обробника (рядок 5) містить єдиний оператор повернення значення NULL.

Рядок 6 є "ядром" функції GetData, які виконують основну роботу. У ній міститься команда повернення одного значення, одержуваного в результаті виконання запиту виду «select - from - where»; тип (в даному випадку - INTEGER) повертається повинен збігатися з тим, що зазначений в пропозиції RETURNS заголовка оголошення функції. Якщо відносно ПРЕДМЕТ існує єдиний кортеж, компонент НазвПредмета якого містить значення pr, передане функції у вигляді аргументу, запит, як і очікується, поверне значення компонента Дата цього кортежу. Якщо ж при виконанні запиту система генерує виняток, пов'язане або з відсутністю

кортежу для предмета з вказаною назвою, або з тим, що таких кортежів кілька, викликається обробник виключень, який в якості значення, що підлягає поверненню, встановлює NULL. Крім того, оскільки, в оголошенні обробника заданий режим EXIT, управління передається за межі поточного блоку коду, але оскільки після інструкції END ніяких команд більше немає, функція завершує роботу, повертаючи значення NULL.

Використання збережених процедур та тригерів

Як вже говорилося раніше до збереженої процедури або функції можна звернутися з програми, написаної на традиційному мовою програмування, з коду іншої процедури або функції PSM або засобами базового інтерфейсу SQL. Способи використання збережених процедур і функцій мало чим відрізняються від технологій програмування на «звичайних» мовах - процедури PSM викликаються за допомогою службового слова CALL, а функції часті застосовуються в составе більших виразів (наприклад, в якості правобічних операндів операторів присвоювання). Нижче показана, як збережена функція може бути викликана з-поміж базового інтерфейсу SQL.

Приклад 7.3.7. Припустимо, що в складі бази даних Навчальний процес є модуль, який містить визначення функції GetData, наведене в попередньому прикладі. Припустимо також, що засобами базового інтерфейсу SQL необхідно зафіксувати в базі даних той факт, що несвоєчасна здача заліку з предмету Інформатика впливає на стипендію. Значення дати здачі заліку, проте, невідомо. Якщо такий предмет дійсно існує і інформація про нього присутня відносно ПРЕДМЕТ, немає необхідності виконувати попередній запит, щоб дізнатися відсутні відомості. Простіше відразу виконати наступну команду вставки кортежу в відношення СТИПЕНДІЯ.

```
INSERT INTO СТИПЕНДІЯ (НазваПредмета, ТермінСдачі)  
VALUES ( 'Інформатика', GetData ( 'Інформатика'));
```

Оскільки функція GetData здатна повернути NULL, якщо кортежу з інформацією про предмет Інформатика немає або якщо таких кортежів кілька, можлива ситуація, коли компоненту ТермінСдачі кортежу, що вставляється в СТИПЕНДІЯ, буде присвоєно значення NULL.

Тригери

Тригер - це оператор SQL, який автоматично виконується в СУБД і супроводжує процес модифікації зазначеної таблиці. Тригер подібний процедурі SQL тим, що являє собою

іменований блок SQL з розділами оголошення, виконання і обробки умов. Однак, на відміну від звичайної процедури, тригер виконується неявно в кожному випадку виникнення тригерної події, до того ж не має параметрів. Приведення тригера в дію іноді називають запуском (firing), або **активізацією**, тригера. Тригери можуть використовуватися для досягнення наступних цілей:

- перевірка правильності введених даних і перевірка виконання складних обмежень цілісності даних, які важко, якщо взагалі можливо, підтримувати за допомогою обмежень цілісності, встановлених для таблиці;
- видача попереджень (наприклад, за допомогою електронної пошти), які нагадують про необхідність виконати деякі дії при оновленні таблиці певним чином;
- накопичення інформації аудиту за допомогою фіксації відомостей про внесені зміни і тих особах, які їх виконали;

- підтримка **реплікації** (реплікація - процес формування і відтворення численних копій даних на одному або декількох вузлах).

Основний формат оператора CREATE TRIGGER показаний нижче.

```
CREATE TRIGGER TriggerName  
BEFORE | AFTER <triggerEvent> ON <TableName>  
[REFERENCING <oldOrNewValuesAliasList>]  
[FOR EACH {ROW | STATEMENT}]  
[WHEN (triggerCondition)]  
<TriggerBody>
```

Тригерні події включають вставку, видалення і оновлення рядків в таблиці. Але тільки в останньому випадку для тригерної події можна також вказати конкретні імена стовпців таблиці. Час запуску тригера визначається за допомогою ключових слів BEFORE і AFTER: при вказівці ключового слова BEFORE тригер запускається до виникнення пов'язаних з ним подій, а при вказуванні ключового слова AFTER - після їх виникнення. Що їх тригером дії задаються оператором SQL, який може бути виконаний одним з двох наступних способів:

- для кожного рядка (FOR EACH ROW), охопленої цією подією (так званий тригер на рівні рядка);

- тільки один раз для кожної події (FOR EACH STATEMENT); саме цей спосіб використовується за умовчанням (так званий тригер на рівні оператора).

Позначення <oldOrNewValuesAliasList> відноситься до таких компонентів:

- стара або нова рядок (OLD / NEW або OLD ROW / NEW ROW), якщо використовується тригер на рівні рядка;

- стара або нова таблиця (OLD TABLE / NEW TABLE), якщо використовується тригер AFTER.

Ясно, що старі значення не застосовні для подій вставки, а нові - для подій видалення. Тіло тригера не може містити

- оператори SQL, що застосовуються при обробці транзакцій, наприклад COMMIT і ROLLBACK;

- оператори SQL, які служать для встановлення з'єднання, наприклад CONNECT і DISCONNECT;

- оператори SQL для визначення схеми і управління нею, наприклад команди створення або видалення таблиць, призначених для користувача типів або інших тригерів;

- оператори SQL для управління сеансом, наприклад SET SESSION CHARACTERISTICS, SET ROLE, SET TIME ZONE.

Крім того, стандарт SQL3 не передбачає можливості створення тригерів, які можуть динамічно змінюватися (тобто тригерів, що дозволяють при обробці події вносити зміни в код самого тригера, а потім викликати його знову в циклі, який може стати нескінченним). Оскільки для таблиці може бути визначено декілька тригерів, то велике значення має порядок їх запуску. Запуск тригерів відбувається в міру виникнення тригерних подій (INSERT, UPDATE, DELETE) в наступному порядку:

1. Виконання табличного тригера BEFORE на рівні оператора.

2. Для кожного рядка, охопленої даними оператором:

- а) виконання будь-якого тригера BEFORE на рівні рядка;

- б) виконання самого оператора;
- в) застосування обмежень посилальної цілісності;
- г) виконання будь-якого тригера AFTER на рівні рядка.

3. Виконання табличного тригера AFTER на рівні оператора.

Слід зазначити, що при використанні такої послідовності запуску тригери BEFORE активізуються перед перевіркою обмежень посилальної цілісності, тому не можна виключити таку можливість, що внесення зміни, що викликав активізацію тригера, призведе до порушення обмежень посилальної цілісності бази даних і тому має бути заборонено. Таким чином, при розробці тригерів BEFORE необхідно керуватися правилом, що вони не повинні вносити додаткових змін в базу даних.

Нижче наведені приклади створення і застосування тригерів.

Приклад 7.5.1. Використання тригера AFTER INSERT

Створимо набір записів поштової розсилки для кожного нового запису таблиці НАДХОДЖЕННЯ. Для цього прикладу припустимо, що існує таблиця Mailshot (список поштової розсилки), в якій зберігаються відомості про абітурієнтів, їх середньому вступному балі і необхідному балі надходження на даний факультет.

```
CREATE TRIGGER InsertMailshotTable
AFTER INSERT ON НАДХОДЖЕННЯ
REFERENCING NEW ROW AS пост
BEGIN
    INSERT INTO Mailshot VALUES
    (SELECT а.ФІО Абітурієнта, а.ГодРожд, а.Город, а.Уліца,
     а.Дом, а.Телефон, а.СрБалл
     FROM Абітурієнт а
     WHERE а.НомерАбіт = пост.НомерАбіт AND
     (А.СрБалл => пост.СрБалл))
END;
```

Цей тригер виконується після вставки нового запису. Конструкція FOR EACH опущена, тобто за замовчуванням використовується конструкція FOR EACH STATEMENT, оскільки оператор INSERT вставляє кожен раз тільки один рядок. Тіло тригера утворює оператор INSERT, побудований на підзапиті, який знаходить все записи з відомостями про абітурієнтів.

За умови правильного використання тригери можуть стати дуже потужним механізмом. Основна перевага тригерів полягає в тому, що стандартні функції можуть зберігатися всередині бази даних і незмінно активізуватися при кожному оновленні її даних. Це дозволяє істотно спростити додаток. Проте слід згадати і про властиві триггерам недоліки:

- Складність. При переміщенні з програми в базу даних деяких функцій ускладнюються завдання її проектування, реалізації та адміністрування.
- Приховані функціональні засоби. Переміщення деяких функцій в базу даних і збереження їх у вигляді одного або декількох тригерів може привести до приховування від користувача деяких функціональних можливостей. Хоча це певною мірою спрощує роботу користувача, але, на жаль, може привести до появи незапланованих, потенційно небажаних і шкідливих побічних ефектів, оскільки в

цьому випадку користувач не може контролювати деякі процеси, що відбуваються в базі даних.

- Вплив на продуктивність. Перед виконанням кожного оператора зі зміни стану бази даних СУБД повинна перевірити тригерні умова з метою з'ясування необхідності запуску тригера для цього оператора. Дана операція позначається на загальній продуктивності СУБД. Очевидно, що при зростанні кількості тригерів накладні витрати, пов'язані з такими діями, також зростають. У моменти пікового навантаження вони можуть викликати помітне зниження продуктивності системи.

Привілеї

Для створення тригера користувач повинен володіти привілеєм TRIGGER для заданої таблиці, привілеєм SELECT для будь-яких таблиць, зазначених у тригерній умови в конструкції WHEN, а також привілеями для виконання всіх операторів SQL в тілі тригера.

Питання:

- 7.3.1. Як виглядає оголошення процедури, що зберігається в функції?
- 7.3.2. Визначте основні параметри процедури і функції.
- 7.3.3. Як здійснюється виклик процедури? Опишіть використання цього виклику для різних контекстів.
- 7.3.4. Які команди використовуються для повернення значення і оголошення локальної змінної. Опишіть оператор присвоювання.
- 7.3.5. Розкажіть, що ви знаєте про розгалуження.
- 7.3.6. Які існують способи використання запитів виду «select - from - where »в кодї PSM?
- 7.3.7. Опишіть основні цикли стандарту PSM.
- 7.3.8. Що розуміється під поняттям «обробник виключень»?
- 7.3.9. Дайте визначення поняттю тригер.
- 7.3.10. Поясніть схожість і відмінність тригера і процедури SQL?
- 7.3.11. Для досягнення яких цілей можуть використовуватися тригери?
- 7.3.12. Опишіть основний формат CREATE TRIGGER і поясніть суть, що входять в нього, операторів.
- 7.3.13. Визначте порядок запуску тригерів в міру виникнення тригерних подій (INSERT, UPDATE, DELETE).
- 7.3.14. Проаналізуйте основні переваги та недоліки тригерів?

Тема 8

Управління транзакціями

У типових СУБД представлені функції, призначення яких полягає в гарантованому підтримці бази даних в достовірному і узгодженому стані, а саме: служби підтримки транзакцій, управління паралельним доступом і засоби відновлення баз даних. Характеристики надійності, достовірності та узгодженості даних повинні зберігатися при відмовах обладнання або програмних компонентів, а також при експлуатації бази даних в багатокористувацької середовищі. Ці три функції докладно розглядаються в даній темі.

Незважаючи на те, що кожна з цих функцій буде обговорюватися окремо, всі вони знаходяться у взаємній залежності. І механізм контролю при паралельному доступі, і засоби відновлення призначені для захисту баз даних від втрати даних або їхнього переходу в неузгоджений стан. Багато СУБД допускають одночасне виконання кількох користувачами різних операцій в базі даних. Якщо ці операції здійснюються безконтрольно, то виконувані користувачами дії можуть довільним чином впливати друг на друга, внаслідок чого база даних може перейти в неузгоджений стан. Для виключення подібних явищ в кожній СУБД реалізується протокол управління паралельним доступом.

Відновлення бази даних являє собою процедуру перекладу бази даних в деякий допустимий стан, виконувану після відмови. Відмова бази даних може виникнути в результаті аварійної зупинки системи, збою обладнання, виявлення програмних помилок або несправності носіїв інформації.

Будь-яка СУБД повинна мати засоби відновлення системи (незалежно від причини відмови), а також забезпечувати повернення бази даних в узгоджений стан.

8.1 Підтримка транзакцій

Транзакція - дія або ряд дій, які виконуються одним користувачем або прикладною програмою, які здійснюють читання або зміни змісту бази даних.

Транзакція є логічною одиницею роботи, що виконується в базі даних. Вона може бути представлена окремою програмою, частиною програми або навіть окремою командою (наприклад, командою INSERT або UPDATE мови SQL) і включати довільну кількість операцій, які виконуються в базі даних. З точки зору адміністратора бази даних експлуатація будь-якої програми може розцінюватися як ряд транзакцій, в проміжках між якими виконується обробка даних, яка здійснюється поза середовищем бази даних.

Будь-яка транзакція завжди повинна переводити базу даних з одного узгодженого стану в інше, хоча допускається, що узгодженість стану бази може порушуватися під час виконання транзакції.

Будь-яка транзакція завершується одним з двох можливих способів. У разі успішного завершення результати транзакції фіксуються в базі даних, і остання переходить в новий узгоджений стан. Якщо виконання транзакції не увінчалось успіхом, вона скасовується.

Відкат (скасування) транзакції - відновлення узгодженого стану бази даних, в якому вона перебувала до початку транзакції, що не завершилася успішно.

Зафіксована транзакція не може бути відмінена. Якщо виявиться, що зафіксована транзакція була помилковою, буде потрібно виконати іншу транзакцію, що скасовує дії, виконані першої транзакцією. Така транзакція є *компенсуючою*.

Компенсуюча транзакція - транзакція, що виконує скасування будь-якої іншої виконаної транзакції.

Але аварійно завершена транзакція, для якої виконано відкат, може бути викликана на виконання пізніше і, в залежності від причин попередньої відмови, цілком успішно завершена і зафіксована в базі даних.

В жодній СУБД не може бути передбачений апріорний спосіб визначення того, які саме операції оновлення можуть бути згруповані для формування єдиної логічної транзакції. Тому повинен застосовуватися метод, що дозволяє вказувати межі кожної з транзакцій ззовні, з боку користувача.

Порядок виконання операцій транзакції прийнято позначати за допомогою діаграми переходів.

Властивості транзакцій

Існують певні властивості, якими повинна володіти будь-яка з транзакцій. Нижче представлені чотири основних властивостей транзакцій, які прийнято позначати аббревіатурою НУІС (нерозривність, узгодженість, ізолюваність, стійкість), що складається з перших букв назв цих властивостей.

1. Нерозривність

Це властивість, для опису якого можна застосувати вираз "все або нічого". Будь-яка транзакція є неподільною одиницею роботи, яка може бути або виконана повністю, або не виконана взагалі. За забезпечення нерозривності відповідає підсистема відновлення СУБД.

2. Узгодженість

Кожна транзакція повинна переводити базу даних з одного узгодженого стану в інше. Відповідальність за забезпечення узгодженості покладається і на СУБД, і на розробників додатків.

В СУБД узгодженість може забезпечуватися шляхом виконання всіх обмежень, заданих в схемі бази даних, таких як обмеження цілісності і обмеження предметної області.

3. Ізолюваність

Усі транзакції виконуються незалежно один від одного. Іншими словами, проміжні результати незавершеної транзакції не повинні бути доступні для інших транзакцій. За забезпечення ізолюваності відповідає підсистема управління паралельним виконанням.

4. Стійкість

Означає, що результати успішно завершеної (зафіксованої) транзакції повинні зберігатися в базі даних постійно і не повинні бути втрачені в результаті наступних збоїв. За забезпечення стійкості відповідає підсистема відновлення.

Архітектура бази даних

На рис. 8.1 представлений фрагмент повної структурної схеми СУБД, що включає чотири високорівневих модуля бази даних, які відповідають за обробку транзакцій, управління паралельним доступом і відновлення системи.

Диспетчер транзакцій координує роботу транзакцій за запитом від прикладних програм. Він взаємодіє з *планувальником*, що відповідає за реалізацію обробки

стратегії управління паралельним доступом. У деяких випадках планувальник називають диспетчером блокувань, якщо використовуваний протокол управління паралельним виконанням будується на основі системи блокувань.



Рис. 8.1. Подсистема обробки транзакцій типичної СУБД

Мета роботи планувальника полягає в досягненні максимально можливого рівня розпаралелювання роботи, за умови виключення впливу паралельно виконуючих транзакцій один на одного, оскільки це може послужити джерелом порушення цілісності або узгодженості бази даних.

Якщо в процесі виконання транзакції відбувається відмова, то база даних може виявитися в несогласованному стані. Завданням диспетчера відновлення є забезпечення того, що в подібних випадку база даних буде автоматично повернена в той стан, в якому вона перебувала до початку даної транзакції, і, отже, залишиться узгодженою. Нарешті, диспетчер буферів відповідає за передачу даних між оперативною пам'яттю комп'ютера і зовнішньою дисковою пам'яттю.

8.2 Управління паралельним доступом

Управління паралельним доступом - процес організації одночасного виконання в базі даних різних операцій доступу, які гарантують запобігання їх впливу один на одного.

Найважливішою метою створення баз даних є організація паралельного доступу багатьох користувачів до загальних даних, що використовуються ними спільно. Забезпечити паралельний доступ відносно нескладно, якщо всі користувачі будуть тільки читати дані, які вміщені в базу. У цьому випадку робота кожного з них не впливає на роботу інших користувачів. Але якщо два або декілька користувачів одночасно звертаються до бази даних і хоча б один з них бажає оновити збережену в базі інформацію, можливо взаємний вплив процесів доступу, здатне привести до неузгодженості даних.

Дане завдання подібна завданням, які стоять перед будь-якою багатокористувацькою комп'ютерною системою, коли кілька програм (або транзакцій) отримують можливість одночасно виконувати операції завдяки використанню мультипрограмної організації роботи, що дозволяє двом або декільком програмам (чи транзакціям) виконуватися в один і той же час.

Розглянемо три приклади потенційних проблем, які можуть мати місце при паралельному виконанні транзакцій: проблему втраченого оновлення, проблему

залежності від незафіксованих результатів і проблему аналізу неузгодженості. Розглядатимуться транзакції, які ми будемо вважати об'єктами управління паралельним виконанням.

Проблема втраченого оновлення

Результати цілком успішно завершеної операції оновлення однієї транзакції можуть бути перекриті результатами виконання іншої транзакції. Це порушення відомо як проблема втраченого оновлення. Її суть проілюстрована прикладом, наведеним у табл. 8.1. У цьому прикладі транзакція T1 виконується паралельно з транзакцією T2. Транзакція T1 полягає у відрахуванні 5 студентів з групи ТМ-32, в якій спочатку знаходиться 30 студентів, а транзакція T2 передбачає зарахування 10 студентів в цю ж групу. Якщо ці транзакції будуть виконуватися послідовно, одна за одною, без чергування їх операцій, то після завершення роботи на рахунку буде знаходитися сума в 35 студентів, незалежно від порядку виконання транзакцій.

Але припустимо, що транзакції T1 і T2 починаються практично одночасно і кожна з них зчитує початкове значення кількості студентів в групі рівне 30 студентам. Потім транзакція T2 збільшує кількість студентів на 10 і записує результат, рівний 40 студентам в базу даних - до групи ТМ-32.

Тим часом транзакція T1 зменшує значення своєї копії початкової кількості студентів в групі ТМ-32 і записує отримане значення, рівне 25 студентам в базу даних в ту ж групу, перекриваючи результат попереднього оновлення. В результаті з групи "зникає" 10 студентів, доданих при виконанні попередньої операції.

Можна уникнути втрати результатів виконання транзакції T2, заборонивши транзакції T1 зчитувати вихідне значення в групі ТМ-32 аж до завершення виконання транзакції T2.

Таблиця 8.1. Приклад проблеми втраченого оновлення

Час	Транзакція T1	Транзакція T2	Поле «Кількість студентів» записи ТМ-32
t1		Початок транзакції	30
t2	Початок транзакції	Зчитування КС з «Кількості студентів»	30
t3	Зчитування КС з «Кількості студентів»	$КС = КС + 10$	40
t4	$КС = КС - 5$	Запис КС в «Кількість студентів»	25
t5	Запис КС в «Кількість студентів»	Закінчення транзакції	25
t6	Закінчення транзакції		25

Проблема залежності від незафіксованих результатів (або "брудного" читання)

Проблема залежності від незафіксованих результатів виникає в тому випадку, якщо одна з транзакцій отримує доступ до проміжних результатів виконання іншої транзакції до того, як вони будуть зафіксовані в базі даних. У табл. 8.2 наведено приклад залежності від незафіксованих результатів, що викликає появу помилки. У цьому прикладі використовуються ті ж початкові дані для кількості студентів в ТМ-32, що і в попередньому прикладі.

В цьому випадку транзакція Т4 збільшує значення кількості студентів в групі ТМ-32 до 40 осіб, після чого виконання транзакції скасовується, тому СУБД повинна виконати відкат транзакції з відновленням первісного значення кількості студентів в ТМ-32, рівного 30 осіб. Проте до цього моменту транзакція Т3 вже встигла вважати змінене значення «Кількість студентів» (40 осіб) і використовувала саме це значення при виконанні операції відрахування 5 студентів з групи, після чого зафіксувала в базі даних невірний результат, рівний 35 студентам (замість правильного - 25 студентів). Значення «Кількість студентів», лічене в транзакції Т4, називається брудними даними. Від цього терміна походить друга назва розглянутої проблеми - проблема брудного читання.

Таблиця 8.2. Приклад проблеми залежності від незафіксованих результатів

Час	Транзакція Т3	Транзакція Т4	Поле «Кількість студентів» записи ТМ-32
t1		Начало транзакції	30
t2		Зчитування КС з «Кількість студентів»	30
t3		КС=КС+10	40
t4	Начало транзакції	Запис КС в «Кількість студентів»	40
t5	Зчитування КС з «Кількість студентів»		40
t6	КС=КС-5	Відкат	35
t7	Запис КС в «Кількість студентів»		35
t8	Закінчення транзакції		35

Проблему можна усунути, заборонивши транзакції Т3 зчитувати значення «Кількість студентів» до прийняття рішення про те, чи повинна бути виконана фіксація або відкат транзакції Т4.

Проблема аналізу неузгодженості

Проблема аналізу неузгодженості виникає в тих випадках, коли транзакція зчитує кілька значень з бази даних, після чого друга транзакція оновлює деякі з них

значень безпосередньо під час виконання першої транзакції. Наприклад, транзакція, що підсумовує дані, вибрані з бази (скажімо, що обчислює загальну суму на рахунках), отримає невірне значення, якщо під час її виконання інша транзакція змінить зчитані нею значення. Приклад подібної помилки наведено в табл. 8.3. Тут транзакція Т6, що обчислює підсумкове значення, виконується паралельно з транзакцією Т5.

Транзакція Т6 обчислює підсумовування кількості студентів в групі ТМ-32 (30 осіб), ТМ-31 (25 студентів) та ТЗ-31 (20 студентів). Однак у цей же час транзакція Т5 здійснює переведення 5 студентів з групи ТМ-32 в ТМ-31. В результаті обчислене транзакцією Т6 значення виявляється невірним (більше на 5 студентів). Цю проблему можна усунути, заборонивши транзакції Т6 зчитувати значення кількості студентів ТМ-32 і ТМ-31 до тих пір, поки транзакція Т5 не зафіксує виконані нею поновлення.

Таблиця 8.3. Приклад проблеми усунення неузгодженості

Час	Транзакція Т5	Транзакція Т6	Поле «Кількість студентів» записи ТМ-32	Поле «Кількість студентів» записи ТМ-31	Поле «Кількість студентів» записи ТЗ-31	Загальна кількість студентів
t1		Початок транзакції	30	25	20	
t2	Початок транзакції	Сума=0	30	25	20	0
t3	Зчитування значення «Кількість студентів» з ТМ-32	Зчитування значення «Кількість студентів» з ТМ-32	30	25	20	0
t4	КС=КС-5	Сума=Сума+КС	25	25	20	30
t5	Запис «Кількість студентів» в ТМ-32	Зчитування значення «Кількість студентів» з ТЗ-31	25	25	20	30
t6	Зчитування значення «Кількість студентів» з ТМ-31	Сума=Сума+КС	25	25	20	50
t7	КС=КС+5		25	30	20	50
t8	Запис в «Кількість студентів» в ТМ-31		25	30	20	50
t9	Закінчення транзакції	Зчитування значення	25	30	20	50

		«Кількість студентів» з ТМ-31				
t10		Сума=Сума + КС	25	30	20	80
t11		Закінчення транзакції	25	30	20	80

Ще одна проблема може виникнути, якщо в деякій транзакції Т відбувається повторне читання раніше зчитуваного елемента даних, але між цими операціями читання була виконана модифікація цього елемента даних в іншій транзакції. Таким чином, в транзакції Т будуть отримані два різних значення одного і того ж елемента даних. Таку ситуацію іноді характеризують як проблему неповторюваного (або нечіткого) читання. Аналогічна проблема може статися, якщо транзакція Т виконує запит, в якому відбувається вибірка з відносини ряду рядків, що задовольняють деякому предикату, а при повторному виконанні цього запиту в більш пізній момент часу виявляється, що отримана множина рядків містить додаткові (фантомні) рядки, які були вставлені іншою транзакцією в період між двома операціями читання. Таку проблему іноді називають фантомним читанням.

Упорядкування та відновлення

Призначення протоколів управління паралельним доступом складається у підготовці такого графіка виконання транзакцій, який виключить можливість їхнього впливу на результати роботи один одного, що дозволить запобігти виникненню проблем, описаних в попередньому розділі. Одне з очевидних рішень полягає у виконанні в кожен момент часу тільки однієї транзакції - попередня транзакція обов'язково повинна бути зафіксована, перш ніж буде дозволено навіть почати виконання наступної транзакції. Однак призначення багатокористувацьких СУБД полягає в забезпеченні максимальної степені розпаралелювання транзакцій користувачів, тому ті транзакції, які не впливають на роботу один одного, цілком можуть виконуватися одночасно (терміни "одночасне" і "паралельне" виконання транзакцій, які тут вживаються як синоніми, повинні розглядатися в контексті багатозадачної організації роботи). Наприклад, транзакції, які звертаються до різних частин бази даних, не вплинуть на роботу один одного і, отже, можуть виконуватися паралельно. В цьому розділі познайомимося з поняттям упорядкування, здатним допомогти виявити ті транзакції, які гарантовано не викличуть порушення узгодженості даних при одночасному виконанні.

Наведемо кілька визначень.

Графік - послідовність запуску операцій декількох паралельно виконуваних транзакцій, що зберігає черговість виконання операцій в кожній окремій транзакції.

Кожна транзакція складається з послідовності операцій, що включають читання і запис даних в базу, які повинні завершуватися або фіксацією, або відкатом отриманих результатів. Графік S являє собою послідовність операцій, що входять до складу множини з n транзакцій T1, T2, ..., Tn, на яку накладається обмеження, що вимагає, щоб послідовність операцій кожної з первинних транзакцій зберігалася в графіку. Тому для кожної транзакції Ti повинен бути збережений порядок її операцій в графіку S.

Послідовний графік - графік, в якому операції кожної з транзакцій виконуються строго послідовно і не можуть чергуватися з операціями, що здійснюються в інших транзакціях.

У послідовному графіку транзакції виконуються строго по черзі.

Наприклад, якщо є дві транзакції (T1 і T2), послідовний порядок виконання транзакцій може передбачати застосування транзакцій T1 потім T2 (або T2, потім T1). Таким чином, при послідовному виконанні ніякий взаємовплив транзакцій неможливий, оскільки в кожен момент часу виконується тільки одна з транзакцій. Однак немає гарантії, що результати застосування всіх варіантів послідовного виконання заданого набору транзакцій завжди будуть однакові. Наприклад, в разі банківських операцій має значення, на якій саме залишок був нарахований відсоток (тобто до або після зняття великої суми з рахунку).

Непослідовний графік - графік, в якому чергуються операції з деякого набору одночасно виконуваних транзакцій.

Причиною проблем, описаних в прикладах 8.1-8.3, є неправильна організація паралельного виконання транзакцій, що призводить до переходу бази даних в неузгоджений стан (в перших двох прикладах) або до видачі користувачеві невірних результатів (в третьому прикладі). Послідовне виконання транзакцій запобігає виникненню подібних проблем. Не має значення, який саме послідовний графік буде обраний, оскільки при послідовному виконанні транзакцій база даних ніколи не переходить в неузгоджений стан. Тому будь-яка послідовність виконання транзакцій із заданої множини буде правильною, хоча можуть бути отримані різні кінцеві результати. Суть упорядкування полягає в пошуку таких непослідовних графіків, які дозволять транзакціям виконуватися паралельно, але без впливу один на одного, і тому переводять базу даних в стан, що досягається при використанні послідовного графіка.

Впорядкований графік - непослідовний графік, який призводить до отримання тих же результатів, які досягаються при використанні певного виду послідовного графіка.

Для запобігання внесення в базу даних неузгодженості, викликаній впливом, транзакцій один на одного, надзвичайно важливо гарантувати упорядкування одночасно виконуваних транзакцій. Розглядаючи умови досягнення упорядкування, вельми важливо враховувати послідовність виконання операцій читання і запису даних.

- Якщо дві транзакції тільки зчитують деякий елемент даних, вони не будуть конфліктувати між собою і послідовність їх виконання не має значення.

- Якщо дві транзакції зчитують або записують абсолютно незалежні елементи даних, вони не будуть конфліктувати між собою і послідовність їх виконання не має значення.

- Якщо одна транзакція записує елемент даних, а інша зчитує або записує цей же елемент даних, послідовність їх виконання має істотне значення.

Конфліктна упорядкованість

Розглянемо тип упорядкування, яка називається конфліктною упорядкованістю. У конфліктно упорядкованому графіку порядок виконання будь-яких конфліктуючих операцій відповідає їх розміщення в послідовному графіку. Згідно з правилом запису, що підкоряється обмеженням (де говориться, що транзакції

повинна оновлювати елемент даних виходячи з його колишнього значення, яке було прочитано транзакцією на самому початку), для перевірки конфліктного упорядкування можна використовувати граф передування (або граф впорядкування). Для графіка S граф передування є спрямований граф $G = (N, E)$, що складається з безлічі вершин N , безлічі спрямованих ребер E і, який формується таким чином:

- Створюється вершина, яка відповідає кожній з транзакцій.
- Створюються направлені ребра $T_i \rightarrow T_j$, якщо транзакція T_j зчитує значення елемента, який записано транзакцією T_i .
- Створюються направлені ребра $T_i \rightarrow T_j$, якщо транзакція T_j записує значення в елемент даних після того, як він був зчитаний транзакцією T_i .
- Створюються направлені ребра $T_i \rightarrow T_j$, якщо транзакція T_j записує значення в елемент даних після того, як він був записаний транзакцією T_i .

Якщо в графі передування, який відповідає графіку S , існує ребро $T_i \rightarrow T_j$, то в будь-якому послідовному графіку S' , еквівалентний графіку S , транзакція T_i повинна передувати транзакції T_j .

Якщо граф передування містить цикли, то відповідний йому графік не є конфліктно упорядкованим.

Упорядкованість за переглядом

Існує і кілька інших типів упорядкованості, які дозволяють сформулювати менш суворе визначення еквівалентності графіків, ніж те, що передбачено в разі конфліктної упорядкованості. Одне з цих менш суворих визначень називають упорядкованість за переглядом. Два графіка S_1 і S_2 , що складаються з одних і тих же операцій, що входять до складу n транзакцій $T_1, T_2 \dots T_n$, є еквівалентними за переглядом, якщо дотримуються наступні три умови.

- Для кожного елемента даних x : якщо транзакція T_i зчитує первісне значення x в графіку S_1 ця ж транзакція T_i повинна зчитувати те ж початкове значення x і в графіці S_2 .
- Для кожної операції читання елемента даних x транзакцією T_i в графіку S_1 : якщо зчитання значення елемента x було записано транзакцією T_j , то і в графіці S_2 транзакція T_i повинна зчитувати значення елемента x , записане транзакцією T_j .
- Для кожного елемента даних x : якщо в графіку S_1 остання операція запису значення x була виконана транзакцією T_i , ця ж транзакція повинна виконувати останній запис значення елемента даних x і в графіці S_2 .

Графік, який упорядковується за перегляду - це графік, який еквівалентний за переглядом до деякого послідовного графіку.

Кожен конфліктно упорядкований графік в той же час є упорядкованим за переглядом, проте зворотне твердження невірне. Може бути строго доведено, що будь-який упорядкований за переглядом графік, який не є конфліктно упорядкованим, містить одну або кілька операцій сліпий записи.

Сліпий запис - запис, при якому не дотримуються правила записи, які підкоряються обмеженням, описаним вище.

У загальному випадку перевірка того, чи є графік упорядкованим за переглядом, відноситься до класу NP-повних задач (комбінаторних задач з нелінійної поліноміальної оцінкою числа варіантів), тому малоймовірно, що коли-то вдасться знайти цілком ефективний алгоритм його вирішення.

На практиці графіки не перевіряються в СУБД на упорядкованість. Це було б недоцільно, оскільки чергування операцій, які паралельно виконуються транзакціями, визначаються операційною системою. Замість цього в СУБД використовуються спеціальні протоколи, що дозволяють створювати упорядковані графіки.

Відновлюваність

Упорядковані графіки - графіки, які дозволяють зберегти узгодженість бази даних за умови, що жодна з транзакцій цього графіка не буде скасована.

Протилежний підхід дозволяє проаналізувати відновлюваність транзакцій, які входять у даний графік. У тому випадку, якщо виконання транзакції було скасовано, властивість нерозривності вимагає, щоб були скасовані всі результати її виконання. Крім того, властивість стійкості вимагає, щоб після фіксації результатів транзакції виконані нею зміни не можна було скасувати (без запуску іншого, який компенсує транзакції).

Сформулюємо поняття відновлюваного графіка.

Відновлюваний графік - графік, в якому для кожної пари транзакцій T_i і T_j виконується наступне правило: якщо транзакція T_j зчитує елемент даних, попередньо записаний транзакцією T_i , то фіксація результатів транзакції T_i повинна виконуватися до фіксації результатів транзакції T_j .

Методи управління паралельним доступом

Упорядкування транзакцій може бути досягнуто кількома різними способами. Існують два основних методи управління паралельним доступом, що дозволяє організувати одночасне безпечне виконання транзакцій при дотриманні певних обмежень: метод блокування і метод тимчасових відміток.

По суті, і блокування, і використання тимчасових відміток є *консервативними* (або *песимістичними*) підходами, оскільки вони відкладають виконання транзакцій, здатних в майбутньому в той чи інший момент часу увійти у конфлікт з іншими транзакціями.

Оптимістичні методи будуються на припущенні, що ймовірність конфлікту невисока, тому вони допускають асинхронне виконання транзакцій, а перевірка на наявність конфлікту відкладається на момент їх завершення і фіксації в базі даних.

Методи блокування

Блокування - процедура, яка використовується для управління паралельним доступом до даних.

Коли деяка транзакція отримує доступ до бази даних, механізм блокування дозволяє (з метою запобігання отримання некоректних результатів) заборонити спроби отримання доступу до цих же даних з боку інших транзакцій.

Саме методи блокування найчастіше використовуються на практиці для забезпечення упорядкованості паралельно виконуваних транзакцій. Існує кілька різних варіантів цього механізму, проте всі вони побудовані на одному і тому ж фундаментальному принципі: транзакція повинна вимагати виконати блокування для читання (що розділяється) або для запису (виняткову) деякого елемента даних перед тим, як вона зможе виконати в базі даних відповідну операцію читання або запису. Встановлене блокування перешкоджає модифікації елемента даних іншими транзакціями або навіть його зчитуванню, якщо це блокування є винятковим. Блокування може бути виконано для елементів різного розміру - починаючи з бази

даних в цілому і закінчуючи окремим полем конкретного запису. Розмір блокуемого елемента визначається ступенем деталізації встановлюваного блокування. Реально блокування може здійснюватися за допомогою встановлення деякого біта у відповідному елементі даних, що означає, що цей фрагмент бази даних є заблокованим. Інший підхід полягає в організації списку заблокованих елементів бази. Існують і інші методи реалізації даного механізму. Нижче представлені основні правила блокування.

Роз'єднуване блокування - блокування елемента даних, при якій може виконуватися читання цього елемента, але не його оновлення.

Виняткове блокування - блокування елемента даних, при якій можуть виконуватися і читання, і оновлення цього елемента.

Оскільки роз'єднувані блокування не можуть служити причиною конфлікту, допускається встановлювати роз'єднувані блокування для читання одного і того ж елемента одночасно з боку відразу декількох транзакцій. У той же час виняткове блокування надає транзакції виключне право доступу до певного елемента даних. Отже, до тих пір, поки транзакція має виняткове блокування, яке поширюється на певний елемент даних, більше ні в одній транзакції неможна виконувати операції читання або оновлення цього елемента даних. Блокування зазвичай використовуються, як показано нижче.

- Будь-яка транзакція, якій необхідно отримати доступ до елементу даних, повинна спочатку виконати блокування цього елемента. Для цього може бути затребувана роз'єднуване блокування, яке забезпечує доступ до елементу тільки для читання, або виняткове блокування, яке надає доступ і для читання, і для запису.

- Якщо елемент ще не заблокований будь-якої іншої транзакцією, блокування елемента буде виконано успішно.

- Якщо елемент даних в реальний час вже заблокований, СУБД проаналізує, чи є тип отриманого запиту сумісним з типом вже існуючого блокування. Якщо запитується роз'єднуване блокування для доступу до елементу, на який вже поширюється інше роз'єднуване блокування, запит на надання блокування виконується успішно. В іншому випадку транзакція переходить в стан очікування, яке триватиме до тих пір, поки існуюче блокування не буде знято.

- Транзакція продовжує утримувати блокування елемента даних до тих пір, поки явно не звільнить її - або в ході свого виконання, або після закінчення (успішному або неуспішному). Тільки після того як з елемента даних буде зняте виняткове блокування, результати виконаної операції записи стануть доступними для інших транзакцій.

Крім цих правил, в деяких системах транзакціям дозволяється встановлювати роз'єднуване блокування на деякий елемент даних, а потім підвищувати рівень цього блокування до виняткової. Така організація роботи фактично дозволяє спочатку визначити в транзакції стан оброблюваних даних, а потім прийняти рішення, чи слід їх змінювати. Якщо механізм підвищення рівня блокування не підтримується, то будь-яка транзакція повинна встановлювати виняткове блокування на всіх елементах даних, які можуть вимагати оновлення на деякому етапі виконання транзакції. Але така організація роботи може привести до зменшення ступеня розпаралелювання операцій в системі. З цієї ж причини в деяких системах транзакціям дозволяється встановлювати виняткове блокування елемента даних з подальшим переходом

більш низький рівень блокування, тобто переходити від виняткової до роз'єднуваного блокування.

Як описано вище, використання в транзакціях блокувань саме по собі не гарантує упорядкованості графіків виконання транзакції.

Для забезпечення упорядкованості слід використовувати додатковий протокол, що визначає моменти установки і зняття блокування для кожної з транзакцій. Найвідомішим з таких протоколів є метод *двофазного блокування* (2PL).

Двофазне блокування

Двофазне блокування - протокол, який передбачає, що всі операції блокування передують першій операції розблокування.

Відповідно до основного правила цього протоколу кожна транзакція може бути розділена на дві фази: фазу розширення, в якій встановлюються усі необхідні блокування і не звільняється жодна з них, і фазу звуження, в якій звільняються всі встановлені раніше блокування, але не може бути встановлено жодне нове блокування. При цьому немає ніякої необхідності в тому, щоб всі необхідні блокування були встановлені одночасно. Як правило, транзакція встановлює деякі блокування, виконує певну обробку, після чого може затребувати встановлення додаткових необхідних для неї блокувань. Однак вона не може звільнити ні однієї з блокувань, поки не буде досягнута стадія виконання, на якій більше не буде потрібно встановлювати нових блокувань. Робота ведеться за наведеними нижче правилами.

- Перш ніж почати роботу з елементом даних, транзакція повинна встановити для нього блокування. Блокування може встановлюватися для читання або запису, в залежності від типу необхідного доступу.

- Як тільки транзакція звільнить хоча б одне встановлене блокування, вона вже не має права вимагати блокування нових елементів.

Якщо СУБД підтримує операції підвищення рівня блокувань, то таке підвищення допускається тільки в фазі розширення. Подібні дії можуть затребувати переходу транзакції в стан очікування на той час, поки інші транзакції скасують встановлені ними роз'єднані блокування даного елемента. Зниження рівня блокування допускається тільки в фазі звуження.

Розглянемо, як протокол двофазного блокування може допомогти в усуненні проблем втраченого оновлення, залежно від незафіксованих результатів і аналізу неузгодженості.

Приклад 8.4. Використання протоколу двофазного блокування для усунення проблеми втраченого оновлення.

Спосіб усунення проблеми втраченого оновлення за допомогою протоколу двофазного блокування показаний в табл. 8.4. Щоб уникнути втрати виконаного оновлення, транзакція T2 повинна спочатку встановити виняткове блокування на елементі «Кількість студентів». Потім вона може вважати з бази даних поточне значення цього елемента, збільшити його на 10 чоловік і записати результат в базу даних. У момент запуску транзакція T1 також потребує встановити виняткове блокування елемента «Кількість студентів». Але, оскільки на елемент даних «Кількість студентів» до цього моменту вже встановлене виключне блокування в транзакції T2, цей запит з боку транзакції T1 не тягне за собою негайного надання необхідного блокування, тому транзакція T1 переходить від стану очікування до

звільнення блокування транзакцією T2. Однак це станеться тільки після фіксації результатів виконання транзакції T2 в базі даних.

Таблиця 8.4. Приклад вирішення проблеми втраченого оновлення

Час	Транзакція T1	Транзакція T2	Поле «Кількість студентів» записи ТМ-32
t1		Початок транзакції	30
t2	Початок транзакції	Виключне блокування «Кількість студентів»	30
t3	Виключне блокування «Кількість студентів»	Зчитування КС з «Кількість студентів»	30
t4	Очікування	$КС = КС + 10$	40
t5	Очікування	Запис КС у «Кількість студентів»	40
t6	Очікування	Закінчення транзакції/ зняття блокування	40
t7	Зчитування КС з «Кількість студентів»		40
t8	$КС = КС - 5$		35
t9	Запис КС у «Кількість студентів»		35
t10	Закінчення транзакції/ зняття блокування		35

Приклад 8.5. Використання протоколу двофазного блокування для усунення проблеми залежності від незафіксованих результатів.

Спосіб усунення проблеми залежності від незафіксованих результатів транзакцій продемонстрований в табл. 8.5. Щоб уникнути виникнення даної проблеми транзакція T4 повинна спочатку потребувати надання їй виняткового блокування на елемент даних «Кількість студентів». Далі вона може прочитати з бази даних поточне значення цього елемента, збільшити його на 10 осіб, а потім записати нове значення в базу даних. При виконанні відкату цієї транзакції виконане нею оновлення значення елемента «Кількість студентів» буде скасовано і цьому елементу даних буде повернуто колишнє значення, що складає 30 чоловік. У момент початку транзакції T3 вона також вимагатиме надання їй виняткового блокування елемента «Кількість студентів». Але, оскільки на цей елемент даних вже поширюється виключне блокування, встановлена транзакцією T4, запит транзакції T3 задовольнити

не вдасться і вона буде переведена в стан очікування аж до звільнення транзакцією T4 необхідного їй елемента даних. Це станеться тільки після відкату результатів виконання транзакції T4 і приведення бази даних в первісний стан.

Таблиця 8.5. Приклад вирішення проблеми залежності від незафіксованих результатів

Час	Транзакція T3	Транзакція T4	Поле «Кількість студентів» записи ТМ-32
t1		Початок транзакції	30
t2		Виключне блокування «Кількість студентів»	30
t3		Зчитування КС з «Кількість студентів»	30
t4	Початок Транзакції	$КС = КС + 10$	40
t5	Виключне блокування «Кількість студентів»	Запис КС в «Кількість студентів»	40
t6	Очікування	Відкат/ зняття блокування	30
t7	Зчитування КС з «Кількості студентів»		30
t8	$КС = КС - 5$		25
t9	Запис КС в «Кількість студентів»		25
t10	Закінчення транзакції/зняття блокування		25

Приклад 8.6. Використання протоколу двофазного блокування для усунення проблеми аналізу неузгодженості

Спосіб усунення проблеми аналізу неузгодженості показаний в табл. 8.6. Для усунення цієї проблеми в транзакції T5 операції читання повинна передувати встановлення виняткових блокувань, тоді як в транзакції T6 операції читання повинна передувати встановлення роз'єднуваних блокувань. Тому при запуску на виконання транзакція T5 видає запит і отримує виняткове блокування на елемент «Кількість студентів». В результаті при спробі отримання роз'єднуваного блокування на елемент «Кількість студентів» в транзакції T6 виконання запиту відкладається і ця транзакція переходить в стан очікування, аж до звільнення необхідного їй елемента даних, тобто до фіксації результатів транзакції T5.

Таблиця 8.6. Приклад вирішення проблеми аналізу неузгодженості

Час	Транзакція T5	Транзакція T6	Поле «Кількість студентів» записи ТМ-32	Поле «Кількість студентів» записи ТМ-31	Поле «Кількість студентів» записи ТЗ-31	Загал ьна кільк ість студе нтів
t1		Початок транзакції	30	25	20	
t2	Початок транзакції	Сума=0	30	25	20	0
t3	Виключне блокування «Кількість студентів» ТМ-32		30	25	20	0
t4	Зчитування значення «Кількість студентів» з ТМ-32	Розділюване блокування «Кількість студентів» в ТМ-32	25	25	20	0
t5	КС=КС-5	Очікування	25	25	20	0
t6	Запис «Кількість студентів» в ТМ-32	Очікування	25	25	20	0
t7	Виключне блокування «Кількість студентів» ТМ-31	Очікування	25	30	20	0
t8	Зчитування значення «Кількість студентів» з ТМ-31	Очікування	25	30	20	0
t9	КС=КС+5	Очікування	25	30	20	0
t10	Запис в «Кількість студентів» в ТМ-31	Очікування	25	30	20	0
t11	Закінчення транзакції/ зняття усіх блокувань T5	Очікування				0 364

t12		Зчитування значення «Кількість студентів» з ТМ-32				0
t13		Сума = Сума +КС				25
t14		Розділюване блокування «Кількість студентів» в ТЗ-31				25
t15		Зчитування значення «Кількість студентів» з ТЗ-31				25
t16		Сума = Сума +КС				45
t17		Розділюване блокування «Кількість студентів» в ТМ-31				45
t18		Зчитування значення «Кількість студентів» з ТМ-31				45
t19		Сума = Сума +КС				75
t20		Закінчення транзакції /зняття усіх блокувань Т6	25	30	20	75

Можна довести, що якщо у всіх транзакціях графіка дотримується протокол двофазного блокування, цей графік буде конфліктно упорядкованим. Але, не дивлячись на те, що протокол двофазного блокування гарантує упорядкованість, можуть виникати проблеми при неправильному виборі моменту звільнення блокувань. Це виникає при каскадному відкаті.

Каскадний відкат - ситуація, в якій скасування єдиної транзакції призводить до цілого ряду відкатів, які залежать від її транзакцій.

Каскадні відкати - явище небажане, оскільки потенційно вони здатні привести до скасування великого обсягу виконаної роботи. Цілком очевидно, що було б дуже корисно розробити протокол, що виключає виникнення каскадних відкатів. Одним з

можливих варіантів є доповнення звичайного протоколу двофазного блокування вимогою відкладати виконання скасування всіх встановлених блокувань до кінця транзакції, як в попередніх прикладах. У подібному випадку продемонстрована в останньому прикладі проблема ніколи не виникне. Цей варіант протоколу називають строгим протоколом двофазного блокування. Можна довести, що при використанні суворого протоколу двофазного блокування транзакції можуть бути упорядковані в тій послідовності, в якій відбувається їх фіксація.

Інший варіант протоколу 2PL, названий обмеженим протоколом двофазного блокування, передбачає припинення до кінця транзакції звільнення тільки виняткових блокувань. У більшості СУБД реалізується один з цих двох варіантів протоколу 2PL.

Ще одна проблема, пов'язана з двофазним блокуванням, може мати місце при будь-яких схемах звільнення заблокованих елементів. Ця проблема носить назву взаємоблокування і є наслідком того факту, що будь-яка транзакція може бути переведена в стан очікування до звільнення необхідного їй елемента даних. Якщо дві транзакції будуть очікувати звільнення елементів, заблокованих іншою транзакцією з цієї ж пари, то виникне стан взаємоблокування. Крім того, транзакції можуть також входити в стан активного блокування, при якій вони залишаються в стані очікування невизначено тривалий час і не мають можливості встановлювати будь-які нові блокування, хоча в СУБД не спостерігається стан взаємоблокування. Ця ситуація можлива у випадках, коли алгоритм перекладу транзакцій в стані очікування не надає рівні умови всіх транзакціях і не враховує час, проведений транзакціями в стані очікування. Для запобігання активного блокування може використовуватися система пріоритетів, в якій пріоритет транзакції тим вище, чим довше вона перебуває в стані очікування. Наприклад, для очікування транзакцій може бути передбачена послідовна чергу, діє за принципом "першим надійшов, першим обслуговується".

Управління паралельним виконанням при використанні індексних структур

Завдання управління паралельним доступом до індексних структур може бути вирішена таким чином, що кожна сторінка індексу розглядається як елемент даних і застосовується описаний вище протокол двофазного блокування. Але оскільки ймовірність частого звернення до індексів вельми велика, особливо якщо мова йде про верхні рівні деревовидних індексів (так як пошук в індексі виконується від його кореня до листя), така проста стратегія управління паралельним виконанням може привести до високої конкуренції за блокуванням. Тому для індексів потрібно більш ефективний протокол блокування. Вивчення порядку проходження за елементами деревовидних індексів дозволяє зробити наступні висновки:

- Шлях пошуку ключа в індексі починається від кореня і проходить до лист-вузлів дерева, але ніколи не повертається на верхні рівні дерева. Це означає, що після звернення до вузла низького рівня в цьому шляху більше не зустрічаються вузли високих рівнів.

- Після вставки в лист-вузол нового значення індексу (ключа і показника) зміни в вузлах верхніх рівнів виникають тільки після остаточного заповнення цього лист-вузла. Це означає, що виняткове блокування лист-вузла встановлюється, тільки якщо в нього повинно бути вставлено нове значення індексу, а виняткове блокування вузлів

верхнього рівня потрібно, якщо в лист-вузол більше немає вільного місця і він повинен бути розділений на два.

На основі цих висновків може бути запропонована наступна стратегія блокування індексу:

- При проведенні пошуку встановлювати роз'єднуване блокування на вузлах, починаючи від кореневого і закінчуючи вузлом найнижчого рівня уздовж шляху, необхідного для досягнення цього вузла. Звільняти блокування батьківського вузла відразу після отримання блокування, яке поширюється на дочірній вузол.

- При виконанні операції вставки можуть застосовуватися консервативний і оптимістичний підходи.

Перший (консервативний) з них передбачає установку виняткових блокувань на всіх вузлах у міру проходження від кореневого вузла дерева до лист-вузла, в якому повинні бути внесені зміни. Такий підхід забезпечує можливість при поділі лист-вузла внести відповідні зміни у вузли, розташовані вздовж шляху до нього аж до кореневого вузла дерева. Але якщо виявляється, що дочірній вузол не заповнений, блокування, яке встановлене на батьківському вузлі, може бути звільнена.

Другий (оптимістичний) підхід передбачає отримання роз'єднаних блокувань для всіх вузлів, за якими проходить шлях до лист-вузла, що піддається змінам, а потім встановлюється виняткове блокування на самому лист-вузлі. При виникненні необхідності поділу лист-вузла рівень роз'єднуваного блокування на батьківському вузлі підвищується до виключного блокування. А якщо при цьому виникає також необхідність розділити вузол на два вузла, підвищення рівня блокувань триває на наступних, більш високих рівнях дерева.

Практика показує, що в більшості випадків поділ вузлів не потрібен, тому останній варіант є найбільш прийнятним.

Защіпки

У багатьох СУБД підтримуються також блокування особливого типу, які називаються **защіпками**, які встановлюються на набагато менш тривалий період, ніж звичайні блокування. Защіпка може бути застосована перед виконанням операції читання або запису сторінки пам'яті на диск для забезпечення нерозривності (або атомарності) такої операції введення-виведення. Наприклад, перед записом сторінки з буферів бази даних на диск може бути встановлена защіпка, сторінка записана на диск, а потім защіпка негайно знята. Оскільки защіпки служать тільки для запобігання конфлікту операцій, які вимагають доступу такого роду, зазвичай вони не застосовуються для реалізації протоколу управління паралельним виконанням на зразок протоколу двофазного блокування.

Взаємоблокування

Взаємоблокування - *тупикова ситуація, яка може виникнути, коли дві (або більше) транзакції знаходяться у взаємному очікуванні звільнення блокувань, які утримуються один одним.*

У табл. 8.7 показані дві транзакції, T7 і T8, виконання яких призводить до взаємоблокування, оскільки кожна з них входить в стан очікування звільнення необхідного ресурсу іншою транзакцією. У момент часу t1 транзакція T7 запитує і отримує виняткове блокування на елемент даних «Кількість студентів» в запису ТМ-32, а в момент часу t2 транзакція T8 отримує виняткове блокування на елемент даних «Кількість студентів» в ТМ-31. Коли в момент часу t6 транзакція T7 запи-

виняткове блокування на елемент даних КС в запису ТМ-31, вона перетворюється на стан очікування, оскільки цей елемент даних вже заблокований транзакцією Т8. У момент часу t_7 транзакція Т8, в свою чергу, запитує виняткове блокування на елемент даних КС в запису ТМ-32 і також переходить в стан очікування, оскільки цей елемент виявляється заблокованим транзакцією Т7. Жодна з транзакцій не в змозі продовжити свою роботу, оскільки кожна очікує завершення роботи іншої. Якщо в системі виникає стан взаємоблокування, то залучені в нього додатки не зможуть вирішити дану проблему власними силами. Відповідальність за виявлення взаємоблокувань і виходу тим чи іншим чином з цієї тупикової ситуації повинна бути покладена на СУБД.

Таблиця 8.7. Приклад взаємоблокування двох транзакцій.

Час	Транзакція Т7	Транзакція Т8
t_1	Початок транзакції	
t_2	Виключне блокування «Кількість студентів» ТМ-32	Початок транзакції
t_3	Зчитування КС з «Кількість студентів» ТМ-32	Виключне блокування «Кількість студентів» ТМ-31
t_4	$КС = КС - 5$	Зчитування КС з «Кількість студентів» в ТМ-31
t_5	Запис КС в «Кількість студентів» в ТМ-32	$КС = КС + 10$
t_6	Виключне блокування «Кількість студентів» ТМ-31	Запис КС в «Кількість студентів» в ТМ-31
t_7	Очікування	Виключне блокування «Кількість студентів» ТМ-32
t_8	Очікування	Очікування
t_9	Очікування	Очікування
t_{10}	Очікування	Очікування

На жаль, існує тільки один спосіб усунути стан взаємоблокування: виконання однієї або декількох транзакцій має бути скасовано. Подібна дія буде супроводжуватися відкатом всіх змін, внесених скасованими транзакціями. Для наведеного в табл.8.7 прикладу можна, припустимо, скасувати виконання транзакції T8. Як тільки її відкат буде завершено, всі встановлені цією транзакцією блокування будуть звільнені і транзакція T7 зможе продовжити своє виконання. Ситуація взаємоблокування повинна бути абсолютно прозорою для кінцевих користувачів, тому СУБД зобов'язана автоматично перезапустити всі скасовані нею транзакції.

Існують три загальних метода обробки взаємоблокувань: встановлення тайм-аутів, запобігання взаємоблокувань, виявлення взаємоблокувань і відновлення нормальної роботи. При використанні тайм-аутів транзакція, яка вимагала встановити яке-небудь блокування, очікує протягом періоду часу, який не перевищує деякого встановленого значення. При використанні методу запобігання взаємоблокувань СУБД заздалегідь визначає ситуації, в яких транзакція зможе викликати появу цієї помилки, і запобігає її виникненню. При використанні методу виявлення взаємоблокувань і відновлення нормальної роботи в СУБД допускається виникнення подібних ситуацій, а потім розпізнаються випадки взаємоблокувань і усуваються їх наслідки. Оскільки метод запобігання взаємоблокувань складніше в порівнянні з застосуванням тайм-аутів або контролю за появою взаємоблокувань і їх усунення, як правило, метод запобігання взаємоблокувань застосовується рідко.

Тайм-аути

Один з найбільш простих методів усунення взаємоблокувань полягає у використанні тайм-аутів. При такому підході транзакція, яка запитує блокування, може очікувати її отримання тільки протягом певного періоду часу, встановленого в системі. Якщо протягом цього періоду блокування не надається, відбувається скасування запиту блокування по тайм-ауту. В цьому випадку СУБД діє згідно з припущенням, що дана транзакція могла виявитися в стані взаємоблокуванні, навіть якщо це не відповідає дійсності, тому відбувається аварійне завершення і автоматичний перезапуск транзакції. Це - дуже просте і зручне рішення задачі усунення взаємоблокувань, реалізоване в деяких комерційних СУБД.

Запобігання взаємоблокувань

Ще один з можливих підходів до усунення взаємоблокувань полягає в упорядкуванні транзакцій на основі використання тимчасових відміток. Були запропоновані два можливих алгоритма. Перший алгоритм, який отримав назву "очікування-скасування", вимагає, щоб тільки більш старі транзакції очікували завершення новіших. В іншому випадку транзакція скасовується і перезапускається з тієї ж тимчасовою оцінкою. Проте рано чи пізно вона стане найстарішою з активних транзакцій і вже не буде скасована. Другий алгоритм, "скасування-очікування", використовує діаметрально протилежний підхід: тільки більш нові транзакції можуть очікувати завершення старішої транзакції. Якщо старіша транзакція буде вимагати виконання блокування елемента даних, вже заблокованого більш новою транзакцією, остання буде скасована.

Виявлення взаємоблокувань

Виявлення взаємоблокувань зазвичай виконується за допомогою **графа очікування** (Wait-For Graph - WFG). Цей **граф** відображає залежність транзакцій один від одного. Транзакція T_i вважається залежною від транзакції T_j в тому випадку,

якщо транзакція T_j заблокувала елемент даних, необхідний для продовження роботи транзакції T_i . Граф очікування являє собою спрямований граф $G = (N, E)$, який складається з множини вершин N , множини спрямованих ребер E і формується таким чином:

- Створюється вершина, яка відповідає кожній транзакції.
- Створюється спрямоване ребро $T_i \rightarrow T_j$, якщо транзакція T_i очікує звільнення елемента даних, заблокованого в даний час транзакцією T_j .

Взаємоблокування має місце в тому і тільки в тому випадку, якщо граф очікування містить цикл. На рис. 8.2 показаний граф очікування для транзакцій, представлених в табл. 8.7. Як показує цей рисунок, граф містить цикл ($T7 \rightarrow T8 \rightarrow T7$), тому можна зробити висновок, що в системі існує взаємоблокування.

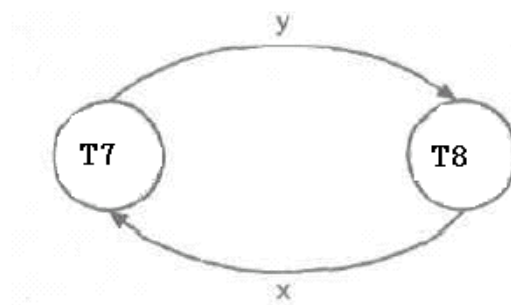


Рис. 8.2. Граф очікування, який показує наявність взаємоблокувань між двома транзакціями

Частота виконання операції виявлення взаємоблокувань

Оскільки наявність циклу в графі очікування є необхідною і достатньою умовою існування в системі взаємоблокувань, алгоритм виявлення цих помилкових ситуацій передбачає формування через регулярні проміжки часу графа очікування і перевірку наявності в цьому циклу. Вибір тривалості між послідовними сеансами виконання алгоритму має важливе значення. Якщо встановлений проміжок виявиться занадто коротким, виявлення взаємоблокувань потребує значних додаткових витрат. Якщо обраний проміжок виявиться занадто тривалим, наявність взаємоблокувань може залишатися непоміченим протягом тривалого часу. Ще один варіант передбачає застосування динамічного алгоритму виявлення взаємоблокувань, виконання якого починається з певного початкового значення проміжку часу. Після кожного сеансу виконання алгоритму контролюється наявність виявленої взаємоблокування.

У разі негативної відповіді проміжок часу може бути збільшений, наприклад удвічі в порівнянні з попереднім значенням, а при кожному виявленні взаємоблокування проміжок часу може бути скорочений, наприклад наполовину (з урахуванням деяких верхніх і нижніх меж).

Відновлення нормальної роботи після виявлення взаємоблокування

Як зазначено вище, після виявлення в СУБД взаємоблокування необхідно виконати аварійне завершення однієї або декількох транзакцій. Для цього слід вирішити декілька завдань.

1. Вибрати транзакції, які скасовуються в результаті взаємоблокувань. При певних обставинах вибір транзакцій, які завершуються аварійно, може виявитися очевидним. Але в інших ситуаціях таке рішення вимагає аналізу додаткових

обставин. У подібних випадках, як правило, слід вибирати транзакції, скасування яких потребують мінімальних витрат. При цьому необхідно враховувати наступні міркування:

а) тривалість виконання транзакції (може виявитися більш доцільним аварійне завершення транзакції, яка була тільки що розпочато, а не транзакції, яка вже виконувалася протягом деякого часу);

б) кількість елементів даних, оновлюваних в транзакції (може виявитися більш доцільним аварійне завершення транзакції, в якій були внесені менші зміни в базу даних, а не транзакції, в якій були внесені значні зміни в базу даних);

в) кількість елементів даних, які все ще підлягають оновленню в транзакції (може виявитися більш доцільним рішення скасувати транзакцію, в якій ще належить внести в базу даних велику кількість змін, а не транзакцію, для завершення якої залишилося внести лише кілька змін). На жаль, саме ці відомості не завжди є в СУБД.

2. Визначити ступінь відкату транзакції. Після прийняття рішення про скасування конкретної транзакції необхідно визначити, до якої міри повинен бути виконаний її відкат. Безумовно, найпростішим рішенням є скасування всіх змін, внесених в транзакції, але воно не завжди виявляється найбільш ефективним. Іноді для усунення взаємоблокувань досить виконати скасування тільки частини змін, внесених в ході виконання транзакції.

3. Запобігти виникненню ситуації виснаження ресурсів. Виснаження ресурсів виникає, якщо для скасування завжди вибирається одна і та ж транзакція, тому її так і не вдається виконати. Виснаження ресурсів багато в чому аналогічно активному блокуванню, яке виникає, якщо протокол управління паралельним виконанням не дозволяє вибрати конкретну транзакцію, яка очікує звільнення блокування. В СУБД можна уникнути ситуації виснаження ресурсів шляхом реєстрації кількості таких випадків, коли для скасування була обрана певна транзакція, і переходу до застосування іншого критерію відбору після досягнення деякої верхньої межі кількості випадків скасування.

Використання тимчасових відміток

Використання методів блокування в поєднанні з протоколом двофазного блокування гарантує упорядкованість графіків. Порядок проходження транзакцій в еквівалентному послідовному графіку заснований на тій черговості, в якій транзакції виконують блокування необхідних їм елементів даних. Якщо транзакції потрібен елемент, який вже заблокований іншою транзакцією, вона перетворюється на стан очікування аж до звільнення необхідного елемента. В іншому підході, який також гарантує досягнення упорядкованості, для визначення черговості виконання транзакцій в еквівалентному послідовному графіку використовуються тимчасові відмітки транзакцій.

Методи використання тимчасових відміток (timestamp), які застосовуються для управління паралельним виконанням, абсолютно відрізняються від методів з використанням блокувань. Не потрібно застосовувати будь-які блокування, і, отже, виключається можливість виникнення взаємоблокувань. Методи блокування зазвичай усувають можливі конфлікти за допомогою переказу транзакцій в стан очікування. Методи з використанням тимчасових відміток не передбачають будь-якого очікування - задіяні у конфлікті транзакції просто скасовуються, після чого запускаються заново.

Тимчасова відмітка - унікальний ідентифікатор, який створюється СУБД з метою позначення відносного моменту часу запуску транзакції.

Тимчасова відмітка може бути створена з використанням системного годинника для фіксації моменту запуску транзакції або за допомогою збільшення значення деякого логічного лічильника при кожному запуску чергової транзакції.

Метод використання тимчасових відміток - протокол управління паралельним виконанням, основна мета якого полягає у встановленні глобальної черговості виконання транзакцій, при якій більш старі транзакції (транзакції з меншим значенням тимчасової відмітки) мають більш високий пріоритет при вирішенні виникаючих конфліктів.

При використанні протоколу тимчасових відміток, якщо транзакція робить спробу читання або запису елемента даних, операція читання або запису виконується тільки в тому випадку, якщо останнє оновлення необхідного елемента даних було виконано більш старою транзакцією. В іншому випадку транзакція, що запросила операцію читання/запису, скасовується і перезапускається з присвоєнням їй нової тимчасової позначки. Нова тимчасова відмітка повинна бути присвоєна транзакції, яка перезапускається для того, щоб запобігти її потраплянню в цикл постійного скасування і перезапуску. Без отримання нової тимчасової відмітки транзакція з старішої тимчасовою оцінкою не зможе завершити свою роботу шляхом фіксації, оскільки новіша транзакція вже встигла зафіксувати свої результати в базі даних.

Крім тимчасових відміток для транзакцій в системі повинні використовуватися тимчасові відмітки для елементів даних. Кожен елемент даних повинен мати тимчасову відмітку читання (`read_timestamp`), яка містить тимчасову відмітку останньої з транзакцій, які виконали його читання, і тимчасову відмітку запису (`write_timestamp`), яка містить тимчасову відмітку останньої транзакції, яка записала (оновила) зміст цього елемента даних. Згідно з протоколом з упорядкуванням за тимчасовими відмітками, обробка транзакції T , що має часову позначку $ts(T)$, виконується в такий спосіб.

Варіант 1. Транзакція T видає команду `read(x)`

- Транзакція T запитує операцію читання елемента даних (x), який вже оновлювався більш новою (пізньою) транзакцією: $ts(T) < write\_timestamp(x)$. Це означає, що старіша транзакція запитує операцію читання елемента, який вже був оновлений більш новою транзакцією. Більш стара транзакція була викликана на виконання занадто пізно і вже не може вважатися попереднім, застаріли значенням, тому мабуть, що будь-які інші зчитані нею значення даних можуть бути не погоджені зі зміненним значенням запитаного елемента даних. У подібній ситуації виконання транзакції T повинно бути аварійно припинено, після чого її слід перезапустити з новою тимчасовою оцінкою.

- В іншому випадку ($ts(T) > write\_timestamp(x)$) операція читання може бути виконана. Може бути встановлено рівність: $read\_timestamp(x) = \max(ts(T), read\_timestamp(x))$.

Варіант 2. Транзакція T видає команду `write(x)`

- Транзакція T запитує запис елемента даних (x), значення якого вже було зчитано більш новою транзакцією: $ts(T) < read\_timestamp(x)$. Це означає, що новіша транзакція вже використовувала поточне значення елемента даних і в результаті його оновлення даною транзакцією може виникнути помилка. Подібна ситуація

відбувається в тих випадках, коли транзакція була викликана на виконання занадто пізно, щоб мати можливість зробити запис, і новіша транзакція вже зчитала старе значення елемента даних або навіть перезаписала його. Єдиним правильним рішенням в цій ситуації це виконати відкат транзакції T і перезапустити її з використанням нової тимчасової відмітки.

- Транзакція T запитує запис елемента даних (x), значення якого вже було перезаписано більш новою транзакцією: $ts(T) < write\_timestamp(x)$. Це означає, що транзакція T намагається помістити в елемент даних (x) застаріле значення. Виконання даної транзакції має бути припинено, після чого слід виконати її відкат і перезапустити з новою тимчасовою оцінкою.

- В іншому випадку операція запису може бути виконана, а тимчасовій відмітці запису оновлюваного елемента даних повинно бути присвоєно нове значення: $write\_timestamp(x) = ts(T)$.

Ця схема, яка називається **базовим протоколом упорядкування по тимчасовим відміткам**, гарантує, що графік виконання транзакцій буде конфліктно упорядкованим, а результати його виконання будуть еквівалентні послідовному графіку, в якому транзакції виконуються в хронологічному порядку, до яких присвоюються їх тимчасові відмітки. Іншими словами, результати виконання даного графіка будуть такими ж, як у випадку почергового запуску транзакцій в порядку номерів, без чергування їх операцій. Однак базовий протокол упорядкування по тимчасовим відміткам не забезпечує відновлюваності графіків.

Правило запису Томаса

Для забезпечення підвищеного рівня розпаралелювання може використовуватися модернізований базовий протокол упорядкування по тимчасовим відміткам, який дозволяє досягти менш жорсткою конфліктною упорядкованості за рахунок скасування застарілих операцій запису. Це розширення, відоме як правило запису Томаса, передбачає наступну зміну правил виконання транзакцією T операцією запису.

- Транзакція T запитує запис елемента даних (x), значення якого вже було зчитано більш новою транзакцією: $ts(T) < read\_timestamp(x)$. Як і раніше, повинен бути виконаний відкат транзакції T і її перезапуск з більш пізньою тимчасовою відміткою.

- Транзакція T запитує запис елемента даних (x), значення якого вже було перезаписано більш новою транзакцією: $ts(T) < write\_timestamp(x)$. Це означає, що новіша транзакція вже перезаписала значення цього елемента даних, і значення, яке старіша транзакція збирається записати в даний елемент, було отримано на основі застарілого вихідного значення даного елемента. У подібному випадку операція запису цілком безпечно може бути проігнорована. Цей метод іноді називають правилом ігнорування застарілого запису. Його застосування дозволяє підвищити рівень розпаралелювання обробки в системі.

- В іншому випадку операція запису може бути виконана, а тимчасовій відмітці запису оновлюваного елемента даних повинно бути присвоєно нове значення: $write\_timestamp(x) = ts(T)$.

Застосування правила запису Томаса дозволяє формувати графіки, які неможливо створити при використанні будь-яких інших протоколів управління паралельним виконанням, описаних в цьому розділі.

Порівняння методів

На рис. 8.3 показаний зв'язок між методами формування конфліктно упорядкованих графіків (Conflict Serializability - CS), графіків, упорядкованих за переглядом (View Serializability - VS), графіків з двофазним блокуванням (Two-Phase Locking - 2PL) і графіків з тимчасовими відмітками (TimeStamping - TS). Відповідно до цього малюнку метод формування графіків, упорядкованих за переглядом, дозволяє формувати всі графіки, що створюються за допомогою трьох інших методів, метод формування конфліктно упорядкованих графіків дозволяє створювати графіки, що формуються за допомогою методів (2PL і TS), а графіки, що створюються за допомогою методів 2PL і TS, частково збігаються.

Слід зазначити, що в останньому випадку існують не тільки графіки, що створюються за допомогою обох методів (2PL і TS), але і графіки, які можуть бути сформовані тільки за допомогою методу 2PL, але не TS, і навпаки.

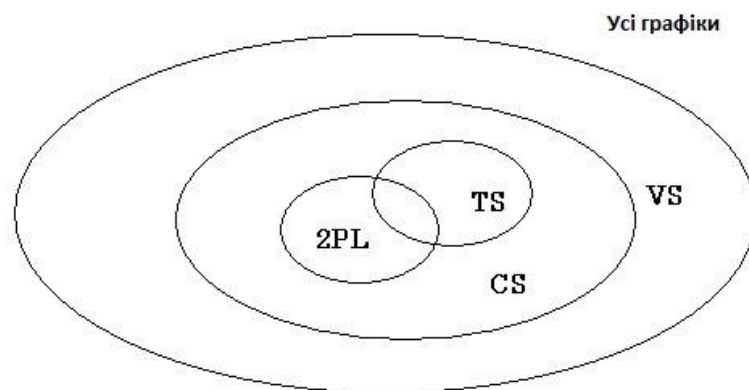


Рис. 8.3. Порівняння конфліктно упорядкованих графіків (CS), графіків, упорядкованих за переглядом (VS), графіків з двофазним блокуванням (2PL) і графіків з тимчасовими відмітками (TS).

Упорядкування тимчасових відміток у разі багатьох версій

Управління версіями даних може також використовуватися для підвищення рівня розпаралелювання, оскільки різні проектувальники працюють з різними версіями одного і того ж об'єкта, не чекаючи завершення транзакцій інших користувачів. У разі, коли в ході проектування знаходять помилку, яка зроблена раніше на якомусь етапі, існує можливість виконати відкат до деякої прийнятної версії проекту. У цьому випадку управління версіями застосовується як альтернативний механізм контролю паралельним доступом при використанні вкладених та багаторівневих протоколів контролю паралельним доступом. Тут коротко розглядається одна з схем управління паралельним доступом, в якій використовується версія для підвищення рівня розпаралелювання обробки на основі тимчасових відміток. У базовому протоколі упорядкування за тимчасовими відмітками, передбачається, що існує тільки одна версія елемента даних і тому тільки одна транзакція може здійснювати доступ до елементу даних в певний момент часу. Це обмеження можна пом'якшити, якщо дозволити кільком транзакціям читати і записувати різні версії одного і того ж елемента даних, гарантуючи, що кожна транзакція буде оперувати з несуперечливим набором версій всіх елементів даних, до яких вона здійснює доступ. При управлінні паралельним доступом в середовищі з багатьма версіями елементів даних кожна операція запису створює нову версію елемента даних зі збереженням старої. Якщо якась транзакція намагається зчитати

елемент даних, система вибирає одну з цих версій, що в цілому гарантує в системі упорядкованість.

Припустимо, що для кожного елемента даних x в базі даних міститься n версій $x_1, x_2, \dots, x_n$, причому для кожної i — i версії в системі зберігаються наступні три величини.

- Значення елемента версії x_i .
- Тимчасова відмітка читання, $read\_timestamp(x_i)$, яка бере найбільше значення серед тимчасових відміток усіх транзакцій, які успішно зчитали версію x_i .
- Тимчасова відмітка запису, $write\_timestamp(x_i)$, що має значення тимчасової відмітки тієї транзакції, яка успішно створила версію x_i .

Нехай $t_s(T)$ є тимчасовою відміткою поточної транзакції T . Тоді в протоколі упорядкування на основі тимчасових відміток для середовища з багатьма версіями можуть використовуватися такі два правила для забезпечення упорядкованості.

1. Транзакція T ініціює операцію запису елемента даних x , $write(x)$.

Якщо в транзакцію T необхідно записати елемент даних x , то слід переконатися в тому, що елемент даних ще не був зчитаний деякою іншою транзакцією T_j , за умови, що $t_s(T) < t_s(T_j)$. Якщо в транзакції T дозволено виконати цю операцію запису, то для забезпечення упорядкованості виконана зміна має бути доведена до відома транзакції T_j . Однак очевидно, що транзакція T_j вже зчитала початкове значення і не в змозі виявити зміну, яка виконалася в транзакції T .

Таким чином, якщо версія x_j має найбільш тимчасову відмітку запису для елемента даних x , яка менше або дорівнює $t_s(T)$ (тобто $write\_timestamp(x_j) < t_s(T)$) і $read\_timestamp(x_j) > t_s(T)$, то транзакція T повинна бути скасована і перезапущена з новою тимчасовою відміткою.

В іншому випадку слід створити нову версію x_i елемента x і встановити йому тимчасову відмітку $read\_timestamp(x_i) = write\_timestamp(x_i) = t_s(T)$.

2. Транзакція T ініціює операцію читання елемента даних x , $read(x)$.

Якщо в транзакції T необхідно зчитати елемент даних x , то їй слід надати версію елемента даних x_j з найбільшою тимчасовою відміткою запису елемента даних x , яка менше або дорівнює $t_s(T)$, іншими словами, повинна бути повернена версія з тимчасовою відміткою $write\_timestamp(x_j)$, такий, що $write\_timestamp(x_j) < t_s(T)$. При цьому слід встановити значення тимчасової відмітки $read\_timestamp(x) = \max(t_s(T), read\_timestamp(x_j))$. Слід зазначити, що при використанні цього протоколу операції читання ніколи не відмінюються.

Версії можуть бути видалені відразу після того, як вони стають непотрібними. Для визначення затребуваності деякої версії елемента слід визначити тимчасову відмітку найстарішою транзакції в системі. Далі, для будь-яких двох версій x_i і x_j , елемента даних x з тимчасовими відмітками менше найстарішої тимчасової відмітки в системі слід видалити стару версію.

Оптимістичні методи упорядкування

У деяких типах обчислювальних систем конфлікти між транзакціями відбуваються дуже рідко, тому додаткова обробка, викликана підтримкою протоколів з блокуванням або з використанням тимчасових відміток, виявляється абсолютно зайвою для більшої частини транзакцій.

Оптимістичні методи упорядкування - методи, засновані на припущенні, що конфлікти в системі виникають рідко, тому ефективніше буде організувати виконання транзакцій, виключивши всі затримки, пов'язані з досягненням гарантованої упорядкованості.

Перед фіксацією результатів транзакції виконується перевірка з метою визначення, чи мав місце конфлікт. Якщо це так, транзакція робить відкат і перезапускається. Оскільки прийнята гіпотеза, що конфлікти в даній системі виникають дуже рідко, то і відкатів потрібно виконувати трохи. Додаткове навантаження, пов'язане з перезапуском деяких транзакцій, може виявитися досить значною, оскільки, по суті, вона буде пов'язана з повторним виконанням всієї транзакції в цілому.

Тому застосування даної схеми має сенс тільки в тому випадку, якщо відкати будуть відбуватися досить рідко, а велика частина транзакцій в системі буде виконуватися без будь-яких додаткових затримок. Подібні методи в принципі дозволяють досягти значно більш високого рівня розпаралелювання в порівнянні з традиційними протоколами, оскільки вони не вимагають використання механізму блокувань.

Оптимістичний протокол управління паралельним виконанням включає дві або три стадії, в залежності від того, чи виконується в даній транзакції тільки читання або оновлення інформації.

- Стадія читання охоплює період від початку транзакції і до моменту, що йде до фіксації результатів. Транзакція зчитує з бази даних значення всіх необхідних їй елементів даних і розміщує їх в локальні змінні. Будь-які оновлення застосовуються тільки до локальної копії даних, але не до інформації, яка зберігається в самій базі даних.

- Стадія перевірки слідує за стадією читання. Виконуються перевірки, необхідні для отримання гарантій відсутності порушення упорядкованості в разі перенесення в базу даних змін, виконаних транзакцією. Для транзакцій, що включають тільки операції читання, перевірка складається в підтвердженні того, що використані транзакцією значення як і раніше залишаються поточними значеннями відповідних елементів даних. Якщо порушень не виявлено, транзакція завершує своє виконання шляхом фіксації. Якщо знайдені змінені значення, транзакція скасовується і перезапускається. Якщо в транзакції виконувалося оновлення даних, то перевірка включає виконання контролю, чи збережеться база даних в узгодженому стані після внесення в неї результатів даної транзакції, а також чи не буде порушено умови упорядкованості. У разі виявлення помилки транзакція скасовується і перезапускається.

- Стадія запису виконується після успішного завершення стадії перевірки, але тільки щодо транзакцій, які включають операцію оновлення. На цій стадії всі зміни, внесені в локальні копії даних, переносяться в базу даних.

На стадії перевірки аналізуються операції, що входять в транзакцію, читання і запису, здатні вплинути на виконання інших транзакцій. Кожній транзакції на початку її виконання присвоюється певна часова відмітка, *start(T)*. Додаткові тимчасові відмітки присвоюються транзакцією на початку стадії перевірки (*validation(T)*) і в момент завершення її виконання (*finish(T)*) - включаючи і стадію

запису, якщо така є). Для того щоб всі перевірки пройшли успішно, необхідно дотримуватися наступних умов:

1. Усі транзакції S з більш старими тимчасовими відмітками повинні бути вже завершені до початку виконання транзакції T : $finish(S) < start(T)$.

2. Якщо транзакція T почала виконуватися до завершення будь-якої з транзакцій S , то повинні дотримуватися вимоги:

а) множина елементів даних, записаних транзакцією, яка почалася раніше, не повинно збігатися з жодним з елементів даних, прочитаних зазначеної транзакцією;

б) стадія запису транзакції, яка почалася раніше, повинна бути завершена до переходу поточної транзакції на стадію перевірки: $start(T) < finish(S) < validation(T)$.

Правило 2.а гарантує, що дані, записані транзакцією, яка почалася раніше, не зчитувалися поточною транзакцією. Правило 2.б гарантує, що операції запису виконуються послідовно, а це гарантує відсутність конфліктів.

Хоча застосування оптимістичних методів вельми ефективно в разі незначної кількості конфліктів в системі, вони можуть викликати відкат окремих транзакцій. Слід зазначити, що відкат стосується тільки локальних копій даних і виникнення каскадного відкату виключається, оскільки будь-які виконані зміни не вносилися в саму базу даних. Однак, якщо скасована транзакція виявиться досить тривалою, багато цінного процесорного часу буде витрачено даремно, так як транзакцію доведеться запускати ще раз. Якщо відкати виникають досить часто, це може вказувати на те, що оптимістичні методи не зовсім підходять для управління паралельним виконанням в даній конкретній системі.

Ступінь деталізації заблокованих елементів даних

Ступінь деталізації - *розмір елементів даних, обраних в якості захищаючої одиниці для протоколу управління паралельним виконанням.*

У всіх протоколах, які обговорювалися вище, управління паралельним виконанням передбачало, що база даних складається з певної кількості "елементів даних", причому додатково не уточнювалося, що означає цей термін. Як правило, в якості елемента даних вибирається один з перерахованих нижче об'єктів, розміри яких знаходяться в межах від дуже великих до найдрібніших (під найдрібнішими об'єктами маються на увазі елементи даних з найменшими, а під великими - з найбільшими розмірами).

- Вся база даних.
- Окремий файл.
- Окрема сторінка даних (іноді її називають областю або блоком бази даних; це - сектор на фізичному диску, що використовується для зберігання таблиць).
- Окремий запис.
- Окреме поле у записі.

Розмір (або ступінь деталізації) елемента даних, який може бути заблокований окремою операцією транзакції, яка має сильний вплив на загальну продуктивність системи і ефективність роботи протоколу контролю паралельним доступом. Однак завжди можна досягти певних компромісів, що слід враховувати при виборі розміру елемента даних в системі. Ми обговоримо можливі компроміси в контексті використання механізму блокувань, хоча ті ж самі міркування будуть справедливими і в разі застосування інших протоколів контролю паралельним доступом.

Розглянемо транзакцію, яка оновлює єдиний рядок у відношенні. Алгоритм протоколу контролю паралельним доступом може дозволити транзакції заблокувати тільки цей рядок, і в даному випадку ступінь деталізації блокування буде встановлена рівною окремому запису. Однак можна дозволити транзакції заблокувати і всю базу даних в цілому, і на цей раз ступінь деталізації блокування буде дорівнювати всій базі даних. Цілком очевидно, що в іншому випадку обрана ступінь деталізації не дозволить виконувати будь-яку іншу транзакцію аж до скасування встановленого блокування.

Безумовно, що така ситуація вельми небажана. З іншого боку, якщо транзакція виконує оновлення 95% записів файлу, то краще дозволити їй заблокувати відразу весь файл і не змушувати систему виконувати блокування кожної з його записів окремо. Але підвищення ступеня деталізації з послідовним переходом від поля або запису до файлу може привести до збільшення ймовірності виникнення взаємоблокування.

Таким чином, чим більше розмір елемента даних, тим нижче ступінь розпаралелювання в системі. З іншого боку, чим дрібніший розмір елемента даних, тим більший обсяг інформації про виконані блокування доведеться зберігати.

Оптимальний розмір елемента даних залежить від характеру виконуваних транзакцій. Якщо типова транзакція обробляє незначну кількість записів, доцільно встановити ступінь деталізації блокування елемента даних на рівні окремому запису. У той же час, якщо типовій транзакції потрібен доступ до великої кількості записів одного і того ж файлу, більш вигідним рішенням буде встановити ступінь деталізації на рівні окремого блоку або навіть файлу. У цьому випадку для отримання доступу до необхідних даних транзакції потрібно заблокувати лише один (або декілька) елементів даних.

Було запропоновано кілька методів динамічного встановлення розміру елемента даних. Відповідно до цих методами, розмір елемента даних залежить від типу виконуваних в поточний момент транзакцій і встановлюється з міркувань оптимальної відповідності їх вимогам. В ідеальному випадку СУБД повинна підтримувати комбінований ступінь деталізації, що дозволяє блокувати окремі записи, сторінки і цілі файли. Деякі системи автоматично підвищують ступінь деталізації блокувань від рівня окремого запису до рівня сторінки або файлу, якщо певна транзакція блокує більше встановленого відсотка записів або сторінок деякого файлу.

Ієрархія ступенів деталізації

Існуючі ступеня деталізації заблокованих елементів можна представити у вигляді ієрархічної структури, в якій кожен вузол буде представляти елемент даних певного розміру, як показано на рис. 8.4. На цій діаграмі кореневий вузол являє всю базу даних, вузли 1-го рівня - це окремі файли, а вузли 2-го рівня - сторінки. На третьому рівні розташовуються вузли, що представляють окремі записи файлів, а на четвертому - поля цих записів. Якщо якийсь з елементів даних блокується, то автоматично виявляються заблокованими і всі його вузли-нащадки.

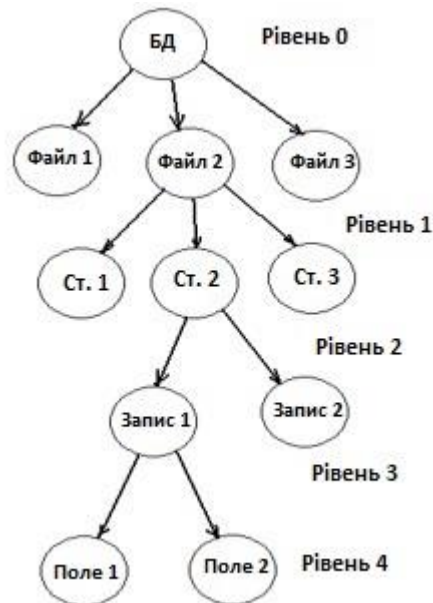


Рис. 8.4. Ієрархічна схема рівнів блокувань

Наприклад, якщо транзакція виконає блокування сторінки (Сторінка 2), то всі розташовані в ній записи (Запис 1 і Запис 2) і їх поля (Поле 1 і Поле 2) також виявляться заблокованими. Якщо інша транзакція потребує встановленню блокування несумісного типу для цього ж вузла, СУБД легко визначить, що цю вимогу не вдасться задовольнити.

Якщо деяка транзакція потребує встановити блокування для будь-якого з нащадків вже заблокованого вузла, то СУБД, перш ніж прийняти рішення про можливість задоволенні цього запиту, потрібно обстежити ієрархічний шлях від кореня схеми до того вузла, який представляє запитаний елемент, щоб перевірити, чи є заблокованим будь-який з предків необхідного вузла. Таким чином, при надходженні запиту на виняткове блокування елемента Запис 1 СУБД буде потрібно перевірити його батьківський вузол (Сторінка 2), потім - батьківський вузол його батьківського вузла (Файл 2) і, нарешті, - кореневий вузол ієрархії (База даних) з метою з'ясування, чи не є один з них заблокованим. Якщо буде виявлено, що вузол Сторінка 2 заблокований, виконання запиту слід відхилити.

Крім того, транзакція може потребувати встановити блокування вузла, який має заблокований вузол-нащадок. Наприклад, якщо надійде вимога заблокувати елемент Файл 2, СУБД буде потрібно перевірити стан кожної зі сторінок цього файлу, кожній запис в окремих сторінках і кожне поле всіх записів файлу, щоб з'ясувати, чи не є один з цих елементів заблокованим.

Блокування з урахуванням декількох ступенів деталізації

Для скорочення обсягу пошуку необхідних для виявлення блокувань, які виконані для нащадків вузла, СУБД може використати іншу спеціалізовану стратегію виконання блокування, яке називається блокуванням з урахуванням декількох ступенів деталізації. У цій стратегії використовується новий тип блокування, який називається наміченим блокуванням. Коли блокується будь-який з вузлів схеми, намічене блокування встановлюється на всі вузли, які є предками даного вузла. Тому якщо деякий з нащадків вузла Файл 2 (наприклад, Сторінка 2) є заблокованим і видається запит на установку блокування самого вузла Файл 2, наявність наміченого блокування, яке встановлено для цього вузла, вкаже системі, що один з нащадків даного вузла є заблокованим.

Намічене блокування може бути або розділеним (Shared - для читання), або винятковим (Exclusive - для запису). Намічене розділене блокування (IS) конфліктує тільки з винятковим блокуванням, тоді як намічене виняткове блокування (IX) конфліктує як з розділеним, так і з винятковим блокуванням. Крім того, транзакції можуть встановлювати розділене і намічене виняткове блокування (SIX), яке логічно еквівалентне установці і розділене блокування, і блокування IX. Іншими словами, блокування SIX сумісне тільки з блокуванням IS. Зведені дані про сумісність різних блокувань при використанні методу, що передбачає застосування декількох ступенів деталізації блокувань, наведені в табл. 8.8.

Таблиця 8.8. Сумісність різних типів блокувань при використанні методу, що передбачає застосування декількох ступенів деталізації блокувань

	IS	IX	S	SIX	X
IS	+	+	+	-	-
IX	+	+	-	-	-
S	+	-	+	-	-
SIX	+	-	-	-	-
X	-	-	-	-	-

+ - сумісні, - - несумісні

Для забезпечення упорядкованості графіків в разі використання декількох ступенів блокування протокол двофазного блокування застосовується наступним чином.

- Жоден елемент даних не може бути заблоковано, поки не буде розблоковано вузол, який його представляє.
- Жоден з вузлів не може бути заблоковано, поки його батьківський вузол залишається заблокованим в наміченому блокуванні.
- Жоден вузол не може бути розблоковано, поки не будуть розблоковані всі його нащадки.

У цьому випадку блокування встановлюється, починаючи з кореня ієрархії вниз з використанням намічених блокувань аж до досягнення вузла, що представляє елемент даних, що блокується для запису або читання. Звільнення блокувань виконується знизу вгору. Слід зазначити, що як і раніше зберігається можливість виникнення взаємоблокування, і усунення цих помилок повинно здійснюватися за розглянутими правилами.

8.3 Відновлення бази даних

Відновлення бази даних - процес повернення бази даних в прийнятний стан, втрачене в результаті збою або відмови.

В даному розділі ознайомимося з організацією служб відновлення баз даних. Щоб краще зрозуміти потенційні проблеми, з якими можна зіткнутися при створенні надійних систем, спочатку з'ясуємо, навіщо може знадобитися відновлювати систему, а потім ознайомимося з різними типами відмов, які можуть мати місце в обчислювальному середовищі з базами даних.

Необхідність відновлення

Для зберігання даних в загальному випадку можуть бути використані чотири різних типи носіїв, які тут перераховані в порядку зростання їх надійності: оперативна пам'ять, магнітний диск, магнітна стрічка і оптичний диск. Оперативна пам'ять

тимчасове сховище інформації, вміст якого в разі відмови системи зазвичай руйнується. Магнітні диски представляють собою оперативне постійне сховище інформації. Диски більш надійні і значно дешевше, ніж основна пам'ять, проте швидкість доступу до інформації у них менше на три-чотири порядки. Магнітна стрічка представляє собою автономний постійний носій інформації, надійність якого істотно вище, ніж у магнітного диска, а вартість набагато нижче.

Однак ці носії надають тільки послідовний доступ до інформації, причому швидкість його відносно невелика. Оптичні диски є більш надійними носіями інформації, ніж магнітні стрічки, досить недорогими, більш швидкодіючими і до того ж допускають довільний доступ до інформації. Оперативну пам'ять часто називають первинної пам'яттю, а магнітні диски і стрічки - вторинної (або зовнішньої) пам'яттю. Стійкі сховища забезпечують надійне збереження інформації, розмішуючи кілька її копій на різних постійних носіях (зазвичай на дисках), одночасну відмова яких мало ймовірна. Зокрема, стійке зберігання інформації може бути організовано з використанням RAID-технології, яка гарантує, що відмова окремого дискового пристрою (навіть в процесі передачі даних) не викличе втрату даних.

Існує безліч різних типів відмов, здатних вплинути на функціонування бази даних, кожен з яких вимагає особливих способів обробки. Одні відмови впливають тільки на вміст оперативної пам'яті, інші можуть впливати і на постійну (вторинну) пам'ять системи. Нижче перераховані деякі причини, які викликають відмови.

- Аварійне припинення роботи системи, викликане помилкою обладнання або програмного забезпечення, яке призвело до руйнування вмісту оперативної пам'яті.

- Відмова носіїв інформації, наприклад, руйнування магнітної головки або поява незворотного збою читання, що призводить до втрати частини вмісту вторинної пам'яті системи.

- Помилки прикладних програм, наприклад, логічні помилки в програмах, які отримують доступ до бази даних, що послужили причиною збоїв при виконанні однієї або декількох транзакцій.

- Стихійні лиха - пожежі, повені, землетруси або відмови в мережі електроживлення.

- Недбале або легковажне звернення, яке послужило причиною ненавмисного руйнування даних або програм з боку операторів або користувачів системи.

- Диверсії, або навмисне руйнування і знищення даних, обладнання або програмного забезпечення.

Якою б не була причина відмови системи, існують два принципових сліdstва, які треба враховувати: втрата вмісту оперативної пам'яті, в тому числі буферів бази даних, і втрата копії бази даних на дисках. Далі в цій темі описані концепції та технології, що дозволяють мінімізувати наслідки аварій і успішно відновити систему після збою.

Транзакції і відновлення

Транзакції представляють собою основну одиницю відновлення в системах з базами даних. Саме диспетчер відновлення СУБД забезпечує підтримку двох з чотирьох основних властивостей ACID транзакцій (нерозривність і стійкість) навіть при наявності збоїв в системі. Диспетчер відновлення повинен забезпечити, що при відновленні після збою для кожної окремої транзакції в базі даних будуть постійно фіксуватися або всі внесені нею зміни, або жодне з них. Ситуація ускладнюється тим

фактом, що запис в базу даних не є нерозривною дією (виконуваного за один крок), і тому існує ймовірність, що, коли виконання транзакції буде завершено шляхом фіксації, внесені нею зміни не будуть реально відображені в базі даних з тієї простої причини, що ще не досягли файлів бази даних.

Ще раз повернемося до першого з розглянутих в даній темі прикладу, в якому вимагалось збільшити кількість студентів в групі ТМ-32 (див, табл. 8.1). При виконанні операції читання СУБД здійснює такі дії:

- Визначає дисковий адрес блоку даних, що містить запис з первинним ключем

х.

- Зчитує блок даних з диска і поміщає його в буфер СУБД в оперативній пам'яті.
- Робить копії даних про кількість студентів з буфера СУБД в змінну КС.

При виконанні операції запису СУБД здійснює такі дії:

- Визначає дисковий адресу блоку даних, що містить запис з первинним ключем

х.

- Зчитує блок даних з диска і поміщає його в буфер СУБД в оперативній пам'яті.
- Робить копії даних про кількість студентів з змінної КС в буфер СУБД.
- Виводить блок даних з буфера СУБД в оперативній пам'яті на диск.

Буфера СУБД займають певну частину оперативної пам'яті і використовуються для обміну даними із вторинної пам'яті системи. Тільки після того, як відповідний буфер вивантажений у вторинну пам'ять, можна вважати, що виконані операції оновлення набули сталого характеру. Вивантаження буферів в базу даних може ініціювати за спеціальною командою (наприклад, по команді фіксації транзакції) або ж автоматично, як тільки буфер буде заповнений. Видачу явної вказівки про необхідність запису вмісту буферів у вторинну пам'ять називають примусовим записом.

Якщо відбудеться відмова системи між записом даних в буфер і вивантаженням буфера у вторинну пам'ять, диспетчер відновлення повинен уточнити стан транзакції, яка виконувала запис в момент аварії. Якщо транзакція вже видала команду фіксації, то для забезпечення стійкості її результатів диспетчер відновлення повинен виконати її повторно (redo) (цю операцію часто називають накатом), щоб відновити всі внесені нею зміни.

З іншого боку, якщо на момент відмови системи транзакція ще не була зафіксована, диспетчер відновлення повинен скасувати (undo) будь-які її результати (виконати їх відкат), що гарантуватиме дотримання її нерозривності. Якщо потрібно виконати відкат тільки однієї транзакції, то це - частковий відкат. Частковий відкат може ініціювати планувальником, коли транзакція робить віткат і перезапускається на вимогу протоколу контролю паралельним доступом, як описано в попередньому розділі. Крім того, транзакція може скасовуватися в односторонньому порядку, наприклад, на вимогу користувача або в результаті виникнення виняткової ситуації в прикладній програмі. Якщо потрібно виконати відкат всіх активних транзакцій, то це - глобальний відкат.

Приклад виконання операцій відкату і накату

На рис. 8.5 показано кілька паралельних транзакцій Т1, ..., Т6, які виконуються.

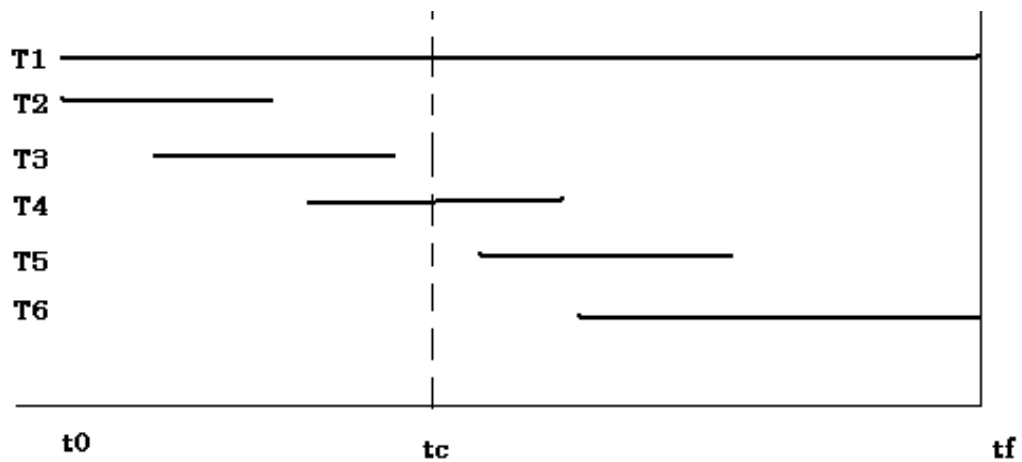


Рис. 8.5. Приклад виконання операцій відкату і накату

Робота СУБД починається в момент часу t_0 , а в момент часу t_f відбувається збій. Припустимо, що результати транзакцій T_2 і T_3 вже були вивантажені у вторинну пам'ять до моменту відмови системи.

Очевидно, що транзакції T_1 і T_6 ще не були зафіксовані на момент відмови системи. Отже, при перезапуску СУБД диспетчер відновлення повинен скасувати ці транзакції. Однак залишається не цілком зрозумілим, які зміни, виконані іншими (зафіксованими) транзакціями T_4 і T_5 , були перенесені в базу даних у вторинній пам'яті. Причина цієї невизначеності полягає в тому, що одна частина вмісту тимчасових буферів СУБД могла бути вже виведена на диск, а інша - ні. При відсутності будь-якої додаткової інформації диспетчеру відновлення буде потрібно виконати примусовий накат транзакцій T_2 , T_3 , T_4 і T_5 .

Управління буферами бази даних

Організація управління буферами бази даних відіграє важливу роль в процесі відновлення, тому перед переходом до подальшого викладу коротко розглянемо застосовувані при цьому методи. Як згадувалося на початку цієї теми, відповідальність за ефективне управління буферами бази даних, які застосовуються для читання і запису сторінки у вторинну пам'ять, покладено на такий програмний компонент, як диспетчер буферів. Ці методи повинні передбачати читання сторінок з диска в буфера до повного їх заповнення, а потім застосування тієї чи іншої стратегії заміщення для визначення того, який буфер (буфера) необхідно примусово записати на диск, щоб звільнити місце для нових сторінок, які повинні бути зчитані з диска. Як приклади стратегії заміщення можна вказати алгоритми з організацією послідовної черги (First-In-First-Out - FIFO) і з витісненням за давністю користування (Least Recently Used - LRU). Крім того, диспетчер буферів не повинен зчитувати сторінку з диска, якщо вона вже знаходиться в одному з буферів бази даних.

Один з можливих підходів передбачає застосування двох змінних ($pinCount$ і $dirty$) в складі інформації управління для кожного буфера бази даних. Цим змінним, які відповідають кожному буферу бази даних, спочатку присвоюється значення 0. При отриманні запиту на читання сторінки з диска диспетчер буферів виконує перевірку для визначення того, чи не знаходиться вже ця сторінка в одному з буферів бази даних. У разі негативної відповіді диспетчер буферів виконує наступні дії.

1. Використовує прийняту стратегію заміщення для вибору буфера, призначеного для заміщення (який в подальшому називається заміщеним буфером), і збільшує значення його змінної $pinCount$ ($pin\ count$ - кількість випадків закріплення),

Тепер затребувана сторінка закріплена в буфері бази даних до тих пір, поки залишається такою, і не може бути знову записана на диск. Застосовуваний алгоритм заміщення не дозволяє вибрати для заміщення буфер, який був закріплений.

2. Якщо змінною *dirty* для буфера заміщення присвоєно нульове значення, виконується запис буфера на диск.

3. Сторінка зчитується з диска в буфер заміщення, і змінною *dirty* цього буфера знову присвоюється значення нуль.

При надходженні повторного запиту до однієї і тієї ж сторінці відповідне значення *pinCount* збільшується на 1. Після того як система повідомить диспетчеру буферів, що застосування цієї сторінки закінчено, відповідне значення *pinCount* зменшується на 1. Разом з тим, система може повідомити диспетчеру буферів, що в зміст сторінки внесено зміну, тому змінною *dirty* присвоюється нульове значення, тобто сторінка відзначається як "брудна" (що вимагає записи на диск). Після того як значення *pinCount* досягає нуля, сторінка стає незакріпленою і може бути записана на диск, якщо в неї внесені зміни (тобто змінна *dirty* має нульове значення).

Нижче описані правила, які можуть застосовуватися з урахуванням необхідності відновлення бази даних при записі сторінок на диск.

- Правило конфіскації дозволяє диспетчеру буферів записувати буфер на диск перш, ніж транзакція виконає фіксацію (буфер не закріплений). Іншими словами, диспетчер буферів може "забрати" сторінку, що належить транзакції. Протилежне йому правило забороняє конфіскацію сторінок.

- Правило примусового запису гарантує, що всі сторінки, модифіковані в ході транзакції, будуть негайно записані на диск після фіксації транзакції. Протилежне йому правило не вимагає примусового запису сторінок.

На практиці найпростіше може бути реалізований підхід, який передбачає використання правила, що забороняє конфіскацію сторінок, і правила примусового запису. Справа в тому, що якщо застосовується правило, що забороняє конфіскацію сторінок, то не потрібно і виконувати відкат транзакції, що завершилася аварійно, оскільки внесені в неї зміни ще не були записані на диск, а застосування правила примусового запису гарантує, що при виникненні аварії системи не доведеться виконувати накат змін, внесених зафіксованою транзакцією, так як всі зміни після фіксації транзакції негайно записуються на диск. Правило, що забороняє конфіскацію сторінок, використовується в протоколі відстроченого відновлення результатів оновлення.

З іншого боку, правило, що допускає конфіскацію сторінок, має певну перевагу перед протилежним йому правилом в тому, що дозволяє уникнути необхідності розподіляти величезний обсяг буферного простору, яке може знадобитися для зберігання всіх копій сторінок, оновлюваних численними паралельними транзакціями (до того ж за певних обставин така спроба виділити буфера для всіх оновлюваних сторінок може виявитися практично нездійсненною). Крім того, перевага над протилежним йому правилом має і правило, яке не потребує примусового запису, оскільки при його використанні сторінка з даними, зафіксованими в попередній транзакції, може залишатися в буфері бази даних, тому її не доведеться зчитувати у пам'ять, якщо вона буде потрібна для наступної транзакції. З цих причин в більшості СУБД використовуються правила, що допускають конфіскацію сторінок і не потребують примусового запису.

Функції відновлення

Типова СУБД повинна надавати такі функції відновлення:

- механізм резервного копіювання, призначений для періодичного створення резервних копій бази даних;
- засоби ведення журналу, в якому фіксуються поточний стан транзакцій і зміни, які вносяться в базу даних;
- функція створення контрольних точок, що забезпечує перенесення виконуваних в базі даних змін у вторинну пам'ять з метою зробити їх постійними;
- диспетчер відновлення, що забезпечує відновлення узгодженого стану бази даних, порушеного в результаті відмови.

Механізм резервного копіювання

Будь-яка СУБД повинна надавати механізм, що дозволяє створювати резервні копії бази даних і її файлу журналу (він описаний в наступному розділі) через певний період часу і без необхідності зупиняти систему.

Резервна копія бази даних використовується в разі пошкодження або руйнування файлів бази даних у вторинній пам'яті. Резервне копіювання може виконуватися для всієї бази даних в цілому або для зміненої її частини (тобто інкрементний). В останньому випадку в копії розміщують відомості тільки про зміни, що накопичилися з моменту створення попередньої повної або інкрементної резервної копії системи. Як правило, резервні копії створюються на автономних носіях, наприклад на магнітних стрічках.

Файл журналу

Для фіксації ходу виконання транзакцій в базі даних СУБД використовує спеціальний файл, який називають **журналом**. Він містить відомості про всі оновлення, які були виконані в базі даних. У файлі журналу може знаходитися наступна інформація.

1) Записи про транзакціях, що включають:

- ідентифікатор транзакції;
- тип запису журналу (початок транзакції, операції вставки, оновлення або видалення, скасування або фіксація транзакції);
- ідентифікатор елемента даних, залученого в операцію обробки бази даних (операції вставки, видалення і оновлення);
- копію елемента даних до операції, тобто його значення до зміни (тільки операції оновлення та видалення);
- копію елемента даних після операції, тобто його значення після зміни (тільки для операцій оновлення і вставки);
- службову інформацію файлу журналу, що включає показники на попередню і наступну запис журналу для цієї транзакції (всі операції).

2) Записи контрольних точок, мова про які піде трохи пізніше.

Дуже часто файл журналу використовується і для інших цілей, відмінних від завдань відновлення (наприклад, для збору відомостей про поточну продуктивність, для аудиту і т.д.). В цьому випадку в файл журналу може розміщуватися багато додаткової інформації (наприклад, відомості про операції читання, про реєстрацію користувачів, завершення сеансів користувачів і т.д.). Однак ця інформація не має відношення до проблеми відновлення бази даних і тому тут не розглядається.

Оскільки файл журналу транзакцій має велике значення для процесів відновлення, він може створюватися в двох і навіть в трьох примірниках, які автоматично підтримуються системою. У разі пошкодження однієї копії при відновленні буде використовуватися інша. Раніше файли журналу створювалися на магнітних стрічках, оскільки цей носій був надійніший і дешевший магнітних дисків. Однак в даний час потрібно, щоб СУБД була здатна швидко відновлювати нормальну роботу в разі незначних пошкоджень. У зв'язку з цим необхідно, щоб файл журналу був доступний оперативному і створювався на пристрої, що підтримує довільний доступ з досить високою швидкістю.

У деяких великих системах, в яких кожен день виробляється величезний обсяг розміщуваного в файл журналу інформації (наприклад, не є чимось незвичайним щоденний обсяг в 10 Мбайт), не представляється можливим утримувати всі ці дані постійно доступними протягом усього необхідного часу. Оперативний доступ до файлу журналу потрібен тільки в разі відновлення після незначних відмов (наприклад, при відкаті транзакції в результаті взаємоблокування). У разі більш серйозних аварій (наприклад, обриві головки магнітного диска) на відновлення системи буде потрібно більш тривалий час і доступ до значної частини всього файлу журналу. У подібних випадках простої, пов'язані з приведенням окремих фрагментів журналу в оперативний стан, виявляються цілком прийнятними.

Один з підходів до автономної обробки файлу журналу полягає в поділі оперативного файлу журналу на дві незалежні частини, організовані у вигляді файлів з довільним доступом. Записи журналу розміщуються у першому файлі до тих пір, поки він не буде заповнений до встановленого рівня (наприклад, на 70%). Потім відкривається другий файл, і всі записи журналу для нових транзакцій записуються вже в нього. Відомості про старі транзакції продовжуються записувати в перший файл до тих пір, поки обробка всіх старих транзакцій не буде завершена. У цей момент перший файл закривається і перекладається в автономний стан. Подібний підхід спрощує відновлення окремих транзакцій, оскільки записи про кожну окрему транзакцію завжди розміщається в одному фрагменті файлу журналу - або в оперативному, або в автономному. Слід зазначити, що файл журналу потенційно є вузьким місцем з точки зору продуктивності будь-яких систем, тому швидкість запису інформації в файл журналу може виявитися одним з найважливіших факторів, що визначають загальну продуктивність системи з базою даних.

Створення контрольних точок

Інформація, яка розміщується в файлі журналу, призначена для використання в процесі відновлення системи після відмови. Але при виникненні відмови може бути відсутня інформація про те, з якого моменту в минулому необхідно почати пошук у файлі журналу, щоб не виконувати накат транзакцій, які вже завершилися з успішною фіксацією в базі даних. Щоб обмежити обсяг пошуку і послідовну обробку інформації в файлі журналу використовується метод створення контрольних точок.

Контрольна точка - момент синхронізації між базою даних і журналом реєстрації транзакцій.

У цей момент усі буфера системи примусово записуються у вторинну пам'ять системи.

Контрольні точки організовуються через встановлений інтервал часу і передбачають виконання наступних дій.

- Перенесення всіх наявних в оперативній пам'яті записів журналу у вторинну пам'ять.
- Запис всіх модифікованих блоків в буферах бази даних у вторинну пам'ять.
- Розміщення у файлі журналу запису про контрольну точку. Ця запис містить ідентифікатори всіх транзакцій, які були активні в момент створення контрольної точки.

Якщо транзакції виконуються послідовно, то після виникнення відмови файл журналу проглядається з метою виявлення останньої з транзакцій, яка почала свою роботу до моменту створення останньої контрольної точки.

Будь-яка більш рання транзакція успішно зафіксована в базі даних, а це означає, що її зміни були перенесені на диск в момент створення останньої контрольної точки. Тому необхідно виконати накат тільки тих транзакцій, які були активні в момент створення контрольної точки, а також всіх подальших транзакцій, які почали свою роботу пізніше і для яких в журналі є записи, як початка, так і фіксації. Та транзакція, яка була активна в момент відмови, повинна бути скасована. Якщо транзакції виконуються в системі паралельно, буде потрібно виконати накат всіх транзакцій, які були зафіксовані з часу створення контрольної точки, і відкат усіх транзакцій, які були активні в момент аварії.

Приклад виконання операцій відкату і накату при наявності контрольної точки.

Звернемося до прикладу виконання операцій відкату і накату. Якщо припустити, що в момент часу t_c була створена контрольна точка, то це дозволить встановити, що результати виконання транзакцій T2 і T3, вже внесені у вторинну пам'ять. Тому диспетчеру відновлення не буде потрібно виконувати накат цих транзакцій.

Однак виконання накату потрібно для транзакцій T4 і T5, результати яких були зафіксовані після створення контрольної точки. Крім того, буде потрібно виконати відкат транзакцій T1 і T6, які все ще були активні в момент виникнення аварії. Зазвичай створення контрольних точок являє собою відносно недорогого операцію, тому часто виявляється можливим створити три або навіть чотири контрольні точки на годину. В результаті при збою потрібно відновити роботу, виконану всього лише за останні 15-20 хвилин.

Методи відновлення

Тип процедури, яка буде використана для відновлення бази даних, залежить від розміру пошкоджень, нанесених цій базі в результаті збою.

Розглянемо два варіанти.

1) Якщо базі даних нанесені великі ушкодження (наприклад, зруйнувалася магнітна головка диска), то буде потрібно відновити її останню резервну копію, після чого повторити в ній усі зафіксовані транзакції, відомості про яких присутні в журналі. Безумовно, передбачається, що файл журналу пошкоджений не був.

2) Якщо база даних не отримала фізичних ушкоджень, але лише втратила узгодженість розміщених в ній даних (наприклад, через аварійної зупинки системи в процесі обробки транзакцій), то досить виконати відкат тих змін, які викликали перехід бази даних в неузгоджений стан. Крім того, може знадобитися виконати накат деяких транзакцій, щоб внесені в них зміни були дійсно зафіксовані у вторинній пам'яті. В даному випадку немає необхідності звертатися до резервної копії бази

даних, оскільки повернути базу в узгоджений стан можна за допомогою інформації про вміст полів до і після модифікації, що зберігається у файлі журналу.

Нижче детально розглядаються два методи відновлення, які можуть бути застосовані в останньому із зазначених вище випадків, тобто коли база даних не була повністю зруйнована, але лише втратила узгоджений стан.

Пропоновані методи, відомі як метод відкладеного оновлення та метод негайного оновлення, відрізняються один від одного способом внесення оновлень у вторинну пам'ять. Крім того, коротко розглянемо альтернативний метод, відомий під назвою метод тіньового сторінкового обміну.

Метод відновлення з використанням відкладеного оновлення

При використанні цього методу оновлення не заносяться в базу даних до тих пір, поки транзакція не видасть команду фіксації виконаних змін. Якщо виконання транзакції буде припинено до досягнення цієї точки, ніяких змін в базі даних виконано не буде, тому і не потрібно їх скасовувати. Однак в даному випадку може знадобитися виконати накат оновлень зафіксованих транзакцій, оскільки їх результати могли ще не досягти бази даних. При застосуванні даного методу файл журналу використовується з метою захисту від аварій системи наступним чином.

- При запуску транзакції в журналі розміщується запис початку транзакції. При виконанні будь-якої операції записи розміщуються в рядок у файл журналу і містить всі зазначені вище дані (за винятком значень елементів даних, які були до попереднього оновлення). Реальний запис змін в буфера СУБД або в базу даних не відбувається.

- Коли транзакція досягає точки фіксації, в журналі розміщується запис фіксації транзакції. Всі записи журналу, які відповідають даній транзакції, виводяться на диск, після чого фіксуються зміни, внесені транзакцією. Для внесення дійсних змін в базу даних використовується інформація, яка знаходиться в файлі журналу.

- У випадку аварійного завершення транзакції записи журналу по даній транзакції анулюються і не виводяться на диск.

Зверніть увагу на те, що записи журналу по виконуваній транзакції виводяться на диск до того, як її результати будуть зафіксовані. Тому якщо відмова бази даних відбудеться в процесі дійсного внесення оновлень в базу даних, поміщені в журнал відомості збережуться і необхідні оновлення можна буде виконати пізніше. У разі відмови файл журналу аналізується з метою виявлення транзакцій, які перебували в процесі виконання в момент відмови. Починаючи з останнього рядка файл журналу проглядається в зворотному напрямку, аж до запису про останню виконану контрольну точку.

- Для будь-яких транзакцій, для яких у файлі журналу присутні записи початок і фіксацію транзакції, повинен бути виконаний накат. Процедура накату транзакцій виконує всі операції запису в базу даних, використовуючи інформацію про стан елементів даних після оновлення, що міститься в записах журналу по даній транзакції, причому в тому порядку, в якому вони були записані в файл журналу. Якщо ці операції вже були успішно завершені до виникнення відмови, це не матиме ніякого впливу на стан елементів даних, оскільки вони не можуть бути зіпсовані, якщо будуть записані ще раз (це означає, що застосовувана операція запису, яка називається ідемпотентним). Однак такий метод гарантує, що будуть оновлені будь-які елементи даних, які не були коректно оновлені до моменту відмови.

- Будь-яка транзакція, для якої в файлі журналу присутні записи початку і скасування транзакції, просто ігноруються, оскільки ніяких реальних оновлень інформації в базі даних по ній не виконувалося, а значить, не потрібно і реального виконання їх відкату.

Якщо в процесі відновлення виникне інший системний збій, записи файлу журналу можуть бути використані для відновлення бази даних ще раз. В цьому випадку не має значення, скільки разів кожна з рядків журналу була використана для повторного внесення змін до бази даних.

Метод відновлення з використанням негайного поновлення

При використанні протоколу негайного оновлення всі зміни вносяться в базу даних відразу ж після їх виконання в транзакції, ще до досягнення моменту фіксації. Крім необхідності виконання накату змін зафіксованих транзакцій слідом за аварією, в цьому випадку може знадобитися виконати відкат змін, внесених транзакціями, що не були зафіксовані до моменту аварії. При застосуванні даного методу файл журналу використовується для захисту системи від збоїв в такий спосіб.

- При запуску транзакції в журналі розміщується запис початку транзакції.
 - При виконанні будь-якої операції запису, що розміщується в файлі журналу рядок містить всі зазначені вище дані.
 - Як тільки згадана вище запис буде розміщена в файлі журналу, всі виконані оновлення вносяться в буфера бази даних.
 - Оновлення записуються в саму базу даних при кожному черговому скиданні буферів бази даних у вторинну пам'ять.
 - Після фіксації транзакції в файл журналу заноситься запис фіксації транзакції.
- Дуже важливо, щоб всі записи (або хоча б певна їх частина) була в файлі журналу до внесення відповідних змін до бази даних.

Ця вимога відомо як протокол попереднього запису журналу. Якщо зміни спочатку будуть внесені в базу даних і збій в системі виникне до розміщення інформації про це в файл журналу, то диспетчер відновлення не буде мати можливості виконати відкат (або накат) результатів даної операції. При використанні протоколу попереднього запису журналу диспетчер відновлення завжди може діяти, ґрунтуючись на тому, що якщо для певної транзакції в файлі журналу відсутній запис фіксації транзакції, це означає, що транзакція все ще була активна в момент виникнення відмови і тому для неї повинен бути виконаний відкат .

Якщо виконання транзакції закінчилося аварійно, то для скасування внесених нею змін може бути використаний файл журналу, так як в ньому збережені відомості про вихідні значення всіх змінених елементів даних. Оскільки транзакція може виконати кілька змін одного і того ж елемента, скасування операцій записів виконується в зворотному порядку. Незалежно від того, чи були результати виконання транзакції внесені в саму базу даних, наявність в записах журналу вихідних значень полів гарантує, що база даних буде приведена в стан, яке відповідає початку скасованій транзакції. На випадок аварії системи процедурою відновлення передбачено використання файлу журналу для виконання відкату або накату транзакцій. Для будь-якої транзакції T, для якої в файлі журналу присутні записи початку і фіксації транзакції, слід виконати її накат, використовуючи для внесення в базу даних значень після зміни всіх оновлених полів, взяті із записів журналу, як описано вище. Відзначимо, що якщо нові значення вже були реально внесені у файл

бази даних, то повторний їх перезапис, хоча і буде зайвою, не справить на базу даних ніякого негативного впливу. А ті зміни, які ще не досягли бази даних до моменту відмови, будуть в неї внесені в процесі відновлення. Для будь-якої транзакції S, для якої в файлі журналу присутній запис початку транзакції, але немає запису фіксації транзакції, необхідно виконати відкат внесених нею змін. На цей раз із записів файлу журналу витягується інформація про значеннях змінених полів до їх зміни, що дозволяє привести базу даних в стан, який вона мала до початку даної транзакції. Операція відкату виконуються в порядку, зворотному порядку запису їх результатів в файл журналу.

Метод тіньового сторінкового обміну

Альтернативою описаним вище схемами відновлення, побудованим на використанні файлу журналу, є метод тіньового сторінкового обміну. Цей метод передбачає організацію на час виконання транзакції двох таблиць сторінок - поточної і тіньової. Коли транзакція починає роботу, обидві таблиці сторінок є однаковими. Тіньова таблиця сторінок в подальшому не змінюється і може бути використана для відновлення бази даних у разі відмови системи. В ході виконання транзакції поточна таблиця сторінок використовується для реєстрації всіх змін, внесених в базу даних. Після завершення транзакції поточна таблиця сторінок стає тіньовою таблицею. Метод тіньового сторінкового обміну має ряд переваг перед методами використання журналу транзакцій: виключаються витрати, пов'язані з веденням журналу транзакцій, процес відновлення відбувається значно швидше, оскільки немає необхідності виконувати операції наката або відкату. Однак йому властиві і певні недоліки: фрагментація даних і необхідність періодичного виконання процедури збору сміття для повернення в систему невикористовуваних блоків пам'яті.

8.4 Покращені моделі транзакцій

Протоколи обробки транзакцій, які розглядалися вище в цій темі, найбільше підходять для тих типів транзакцій, які мають місце в традиційних додатках, - при проведенні банківських операцій і в системах резервування авіаквитків. Всі ці додатки характеризуються такими особливостями.

- Прості типи даних - цілі числа, десяткові числа, короткі символічні рядки і дати.
- Невелика тривалість транзакцій, зазвичай не перевищує декількох хвилин або навіть секунд.

Існують також більш складні типи додатків баз даних, які отримали останнім часом досить широке поширення. До них відносяться різні засоби автоматизації - комп'ютеризоване проектування (CAD), комп'ютеризоване виробництво (CAM) і комп'ютеризована розробка програмного забезпечення (CASE). Всі вони мають ряд загальних характеристик, що відрізняють їх від традиційних додатків баз даних.

- Створюваний проект може бути дуже великим - цілком можливо, що складається з мільйонів частин, часто об'єднаних в багато взаємозалежних підсистем, що представляють собою окремі проекти.

- Проект не є статичним об'єктом і постійно розвивається.

При внесенні до проекту змін їх наслідки часто поширюються на всі наявні в проекті підсистеми. Динамічний характер створюваних проектів може означати, що деякі дії просто не можуть бути передбачені заздалегідь, на самому початку роботи.

- Оновлення, які виконуються, можуть мати далекоглядні наслідки, викликані існуючими складними структурними взаємозв'язками, функціональними залежностями, допусками, нормативами і т. д.

Одна невелика зміна цілком може вплинути на безліч різних проєктованих елементів.

- Дуже часто для окремих компонентів проєкту розглядається багато різних альтернативних варіантів, причому кожен раз потрібно підготувати нову виправлену версію всіх його компонентів.

Рішення подібних завдань вимагає наявності засобів контролю версій і інструментів управління обраній конфігурацією.

- У роботу можуть бути залучені сотні людей, що працюють над декількома паралельними версіями одного великого проєкту. Проте, кінцевий продукт має бути узгодженим та скоординованим. Іноді такий стиль роботи називають кооперативним проєктуванням. Кооперація вимагає чіткої взаємодії і повної узгодженості всіх паралельно виконуваних дій.

Деякі із зазначених вище особливостей є причиною того, що виконувані транзакції стають надзвичайно складними, які оброблять багато елементів даних і мають дуже велику тривалість - можливо, кілька годин, днів і навіть місяців. Подібні вимоги змушують переглянути традиційні протоколи управління транзакціями з метою усунення таких проблем.

- Оскільки діє фактор часу, тривалі транзакції більш чутливі до відмов. Було б абсолютно неприйнятно скасувати виконання подібної транзакції з потенційною втратою дуже великого обсягу виконаної роботи. Отже, щоб мінімізувати можливі втрати, необхідно мати засоби для відновлення того стану транзакції, яке вона мала незадовго до виникнення відмови.

- З часом тривалі транзакції можуть отримати доступ до дуже великої кількості елементів даних (наприклад, встановити блокування). Щоб уникнути можливого взаємного впливу транзакцій всі ці елементи стають недоступними іншим додаткам аж до завершення даної транзакції. Однак вкрай небажано, щоб істотний обсяг даних залишався недоступним протягом досить тривалого періоду часу, оскільки це впливає на ступінь розпаралелювання обробки в системі.

- Чим довше виконується транзакція, тим імовірніше виникнення ситуації взаємоблокувань, якщо в системі використовується протокол, що допускає подібні помилки. Можна показати, що ймовірність виникнення взаємоблокувань пропорційна значенню часу виконання транзакції, зведеному в четверту ступінь.

- Одним із способів організації кооперативної роботи групи співробітників є надання їм можливості спільно використовувати необхідні елементи даних. Однак традиційні протоколи управління транзакціями істотно обмежують можливості такої кооперації, оскільки в них дотримується вимога ізоляції незавершених транзакцій один від одного.

Модель вкладених транзакцій

Модель вкладених транзакцій - модель, в якій транзакція розглядається як колекція взаємопов'язаних підзавдань або субтранзакцій, кожна з яких також може складатися з будь-якої кількості субтранзакцій.

Модель вкладених транзакцій була запропонована Моссом (Moss) в 1981 році. У цій моделі вся транзакція розглядається як набір пов'язаних підзавдань, які називаються субтранзакціями, кожна з яких також може складатися з довільної кількості субтранзакцій. В даному визначенні повна транзакція є деревоподібною структурою або деяку ієрархію субтранзакцій. У моделі вкладених транзакцій присутні транзакції верхнього рівня, що містить деяку кількість дочірніх транзакцій, кожна з яких, в свою чергу, може включати вкладені транзакції, і т. д. У вихідному варіанті, запропонованому Моссом, виконувати операції в базі даних дозволяється лише транзакціям самого нижнього рівня (субтранзакціями самого нижнього рівня вкладеності). У табл. 8.9 наведено приклад вкладеної транзакції T1, в якій виконується підбір викладачів для проведення занять в групах, здійснення контролю знань студентів і проведення факультативних занять. Вона включає субтранзакцію підбір викладачів для проведення занять (T2), підбір комісії з викладачів для здійснення усного контролю знань (T3), підбір викладачів для організація факультативних занять (T4). Транзакція проведення занять теж є вкладеною і складається з двох субтранзакцій: підбір викладачів для проведення лекційних занять (T5) і підбір викладачів проведення практичних занять (T6).

Завершення роботи транзакцій має відбуватися в напрямку знизу вгору. Отже, транзакції T6 і T5 повинні закінчити свою роботу до завершення роботи їх батьківської транзакції T2, а робота транзакції T2 повинна бути завершена до закінчення роботи її батьківської транзакції T1. Однак скасування транзакції на деякому рівні не впливає на виконання транзакцій більш високих рівнів. Замість цього батьківські транзакції дозволяється виконати свою власну операцію відновлення нормальної роботи, скориставшись одним з наступних допустимих способів:

- Повторити виконання субтранзакцій.
- Проігнорувати дану відмову. В цьому випадку субтранзакція розглядається як несуттєва. В даному випадку такою є транзакція T4.
- Запустити альтернативну транзакцію, яка має назву резервна субтранзакція. У нашому прикладі, якщо не вдасться здійснити підбір комісії для усної перевірки знань студентів, можна запустити субтранзакція пошуку одного викладача для проведення письмової перевірки знань.
- Припинити своє виконання.

Оновлення, які були виконані завершеними субтранзакціями проміжних рівнів, повинні бути видні тільки в транзакціях, які є по відношенню до них батьківськими. Тому, коли транзакція T5 буде завершена, внесені нею зміни будуть видні тільки в транзакції T2. Вони будуть недоступні транзакції T3 або будь-який інший транзакції, зовнішньої по відношенню до транзакції T5. Більш того, завершення субтранзакцій є умовно залежним від нормального завершення або скасування її батьківських транзакцій. При використанні даної моделі транзакція верхнього рівня має всі чотирма властивості (ACID), якими повинні володіти традиційні плоскі (що не допускають вкладеності) транзакції.

Для моделі вкладених транзакцій Моссом був запропонований і протокол управління паралельним доступом, побудований за принципом жорсткого двухфазного блокування. Субтранзакції батьківської транзакції виконуються таким чином, як якщо б вони були незалежними транзакціями. Субтранзакціям дозволяється

встановлювати блокування елемента, якщо деяка інша транзакція, вже встановила блокування, з якої конфліктує дана субтранзакція, є по відношенню до неї батьківською.

Таблиця 8.9. Приклад вкладеної транзакції

Команда	Назва транзакції
Початок транзакції T1	Підбір викладачів
Початок транзакції T2	Проведення занять
Початок транзакції T5	Проведення лекцій
Пошук викладачів для проведення занять (лекції)	
Закінчення транзакції T5	
Початок транзакції T6	Проведення практик
Пошук викладачів для проведення занять (практики)	
Закінчення транзакції T6	
Початок транзакції T3	Проведення усного контролю знань
Пошук викладачів для організації комісії для проведення усного контролю знань	
Закінчення транзакції T3	
Початок транзакції T4	Проведення факультативних занять
Пошук викладачів для проведення факультативних занять	
Закінчення транзакції T4	
Закінчення транзакції T1	

Коли субтранзакція завершує роботу, встановлені нею блокування успадковуються батьківською транзакцією. При спадкуванні блокування батьківська транзакція з більшою ймовірністю задає винятковий режим блокування, якщо і дочірні, і батьківські транзакції встановлювали блокування одного і того ж елемента даних.

Нижче перераховані основні переваги моделі вкладених транзакцій.

- Модульність. Транзакція може бути розкладена на довільну кількість субтранзакцій, що сприяє підвищенню ступеня розпаралелювання обробки і розширює можливості відновлення.

- Створення додаткової міри деталізації в механізмах управління паралельним виконанням і відновленням. Введення нового рівня субтранзакцій, крім рівня основних транзакцій.

- Досягнення розпаралелювання обробки в межах транзакції. Субтранзакції можуть виконуватися в паралельному режимі.

- Можливість управління відновленням в межах транзакції.

Незафіксовані субтранзакції можуть завершуватися аварійно, і для них може виконуватися відкат без будь-яких побічних ефектів для інших субтранзакцій.

Емуляція механізму вкладених транзакцій за допомогою точок збереження

Точка збереження - точкою збереження називається певна точка в плоскій транзакції, що представляє деякий частково узгоджений стан, яка може бути використана як точка проміжного перезапуску для транзакцій в разі виникнення будь-яких проблем в подальшому.

Одним із завдань моделі вкладених транзакцій є надання деякої одиниці відновлення роботи, яка має меншу ступінь деталізації в порівнянні з транзакцією. У процесі виконання транзакції користувач може організувати точку збереження, наприклад, за допомогою оператора *SAVE WORK*. При його виконанні формується певний ідентифікатор, який користувач згодом зможе застосовувати для відкату транзакції до даного, зафіксованого стану, наприклад, за допомогою оператора *ROLLBACK WORK* (ідентифікатор точки збереження). Однак, на відміну від моделі вкладених транзакцій, при використанні точок збереження не забезпечується будь-яка з форм розпаралелювання обробки в межах транзакції.

Хроніки

Хроніка - послідовність (плоских) транзакцій, яка може чергуватися з іншими транзакціями.

Концепція хронік (sagas), яка була введена Гарсія-Моліна (Garcia-Molina) і Салемом (Salem) в 1987 році, побудована на використанні поняття компенсуючих транзакцій. СУБД гарантує, що або всі вхідні в хроніку транзакції будуть успішно завершені, або будуть запущені компенсуючі транзакції, необхідні для усунення досягнутих часткових результатів.

На відміну від методу вкладених транзакцій, що допускає довільний рівень вкладення, метод хронік дозволяє наявність єдиного рівня вкладення. Більш того, для кожної виділеної субтранзакції повинна існувати відповідна компенсуюча транзакція, яка буде семантично правильно анулювати результати, що досягаються за допомогою даної субтранзакції. Таким чином, якщо є хроніка, що складається з послідовності n транзакцій $T_1, T_2, \dots, T_n$ з відповідним набором компенсуючих транзакцій $C_1, C_2, \dots, C_n$, то остаточний результат виконання хроніки буде визначатися однією з наведених нижче послідовностей транзакцій.

1. $T_1, T_2, \dots, T_n$ - якщо вся транзакція була успішно завершена.
2. $T_1, T_2, \dots, T_i, C_i, \dots, C_n$ - якщо виконання субтранзакції T_i було аварійно припинено.

У порівнянні зі звичайними моделями плоских транзакцій в хроніках пом'якшені вимоги до ізолюваності окремих транзакцій, оскільки в них допускається вибірка проміжних результатів інших паралельно виконуваних транзакцій ще до їх завершення. Застосування хронік виявляється досить ефективним в тих випадках, коли входять до неї субтранзакції відносно незалежні і можуть бути підготовлені необхідні компенсуючі транзакції, як в розглянутому прикладі. Але в деяких випадках попереднє визначення компенсуючої транзакції може виявитися скрутним. У цьому випадку може знадобитися, щоб було встановлено взаємодію між користувачем і СУБД для визначення прийнятної ступеня компенсації. В інших випадках можливість визначення компенсуючої транзакції відсутня. Наприклад, може виявитися неможливим підготувати компенсуючу транзакцію для транзакції з одержанням готівки з автоматичного касового апарату.

Модель багаторівневих транзакцій

Модель вкладених транзакцій вимагає, щоб процес фіксації субтранзакцій виконувався знизу вгору, в напрямку транзакції верхнього рівня. Тому дану модель прийнято називати моделлю закритих вкладених транзакцій. Це дозволяє підкреслити, що властивість нерозривності в транзакціях зберігається до самого верхнього рівня. На противагу цьому, існує і модель відкритих вкладених транзакцій, в якій нерозривність порушується і часткові результати виконання субтранзакцій можуть бути доступні поза транзакціями. Прикладом моделі відкритих вкладених транзакцій є модель хронік.

Одним з варіантів моделі відкритих вкладених транзакцій є модель багаторівневих транзакцій, в якій дерево субтранзакцій є збалансованим. Вузли одного і того ж рівня дерева відповідають операціям одного і того ж рівня абстракцій в СУБД. Ребра деревовидного графа моделі багаторівневих транзакцій моделюють реалізацію операції за допомогою послідовності операцій більш низького рівня. Рівні n – рівневої транзакції позначаються як $L_0, \dots, L_j \dots L_n$, де L_0 - найнижчий рівень дерева, а L_n - корінь дерева. Методи обробки звичайних плоских транзакцій гарантують, що на найнижчому рівні (L_0) конфлікти будуть відсутні. Основна концепція моделі багаторівневих транзакцій полягає в тому, що дві операції на рівні L_i можуть не конфліктувати, навіть якщо їх реалізації на наступному, більш низькому рівні $L_i - 1$ конфліктують. Оскільки в ній використовується інформація про конфлікти на конкретному рівні, модель багаторівневих транзакцій дозволяє досягти більш високого ступеня паралельності в порівнянні з моделями обробки плоских транзакцій.

Динамічна реструктуризація

Вище ми обговорювали деякі особливості додатків підтримки, виконання різних проектів, наприклад, невизначена тривалість роботи (від декількох годин до місяців), чергування з іншими видами операцій, невизначеність процесу обробки, що не дозволяє передбачити всі аспекти роботи з самого початку її виконання, і т. д. Для подолання обмежень, що накладаються основними властивостями (ACID) плоских транзакцій, було запропоновано дві нові операції: розбиття транзакції (*split_transaction*) і об'єднання транзакцій (*join_transaction*). Принцип, покладений в основу операції розбиття транзакції, полягає в поділі активної транзакції на дві упорядковані транзакції і розподілі між ними виконуваних дій і використовуваних ресурсів (наприклад, заблокованих елементів даних). З цього моменту новостворені транзакції можуть виконуватися незалежно (можливо, навіть під контролем різних користувачів) і оброблятися таким чином, щоб вони завжди були абсолютно незалежними.

Подібний підхід дозволяє зробити проміжні результати транзакції доступними іншим транзакціям, причому з повним збереженням їх семантики, іншими словами, якщо вихідна транзакція відповідала всім вимогам ACID, то так само поведуться і нові транзакції.

Операція поділу транзакції може застосовуватися тільки в тому випадку, якщо можливо створити дві транзакції, які будуть упорядковані по відношенню один до одного і до всіх інших транзакцій, які виконуються в даний момент. Умови, які

дозволяють поділ транзакції Т на дві транзакції, А і В, можна визначити наступним чином.

1. Якщо обидві транзакції, А і В, виконують запис в один і той же елемент даних, то операція запису транзакції В повинна виконуватися після операції запису транзакції А.

2. Транзакція А не може звертатися до будь-яких результатів виконання транзакції В.

3. Транзакція В може звертатися до результатів виконання транзакції А.

Наведені вище умови гарантують, що транзакція А в упорядкованому графіку буде передуватиме транзакції В. Однак, якщо виконання транзакції А завершиться аварійно, виконання транзакції В також необхідно завершити аварійно, оскільки вона використовує дані, записані транзакцією А.

Операція об'єднання транзакцій виконує зворотні дії по відношенню до операції поділу транзакції, поєднуючи результати роботи двох або декількох незалежних транзакцій таким чином, щоб ці транзакції завжди представляли собою єдину транзакцію.

Використання операцій поділу транзакції, доповнених операціями об'єднання однієї або декількох новостворених транзакцій, дозволяє обмінюватися ресурсами між певними транзакціями, не роблячи ці ресурси доступними іншим транзакціям.

Основні переваги методу динамічної реструктуризації полягають у наступному.

- Адаптивне відновлення. Можливість зафіксувати частину виконаної в транзакції роботи, що виключає її залежність від наступних відмов.

- Зниження рівня ізолюваності. Можливість звільнити частину використовувалися в транзакції ресурсів за допомогою фіксації вже виконаної частини її роботи.

Моделі робочих потоків

Всі обговорювані моделі були розроблені з метою подолання обмежень, що накладаються моделлю плоских транзакцій, тому не прийнятних для транзакцій великої тривалості. Однак було відзначено, що всі ці моделі як і раніше не мають потужність, необхідної для задоволення потреб ділових процесів певних видів. Тому були запропоновані більш складні моделі, що представляють собою комбінації моделей відкритих і вкладених транзакцій. Однак, оскільки ці моделі не в повній мірі відповідають вимогам ACID плоских транзакцій, для них використовують більш відповідну назву - моделі робочих потоків.

Робочий потік - деякий вид діяльності, який передбачає координоване виконання безлічі завдань, здійснюваних різними обробними суб'єктами, які можуть являти собою людей або деякі програмні комплекси (наприклад, СУБД, прикладні програми або служби електронної пошти).

Перед будь-якими системами з робочими потоками стоять дві загальні проблеми: визначення та виконання робочого потоку. Обидві проблеми ускладнюються тим фактом, що в багатьох організаціях одночасно використовується кілька незалежних комп'ютерних систем, призначених для автоматизації різних сторін загального процесу. Наведені нижче визначення виділяють основні завдання, які повинні бути вирішені при визначенні робочого потоку.

Специфікація завдання - структура виконання кожного завдання, що визначається за допомогою надання набору доступних для огляду ззовні станів процесу і набору переходів між цими станами.

Вимоги по координації завдань - вимоги, які задаються за допомогою вказівки залежностей між процесами виконання завдань і залежностей між потоками даних, доповнених умовами завершення робочого потоку.

Вимоги до коректності виконання - обмеження, що накладаються на показники виконання робочого потоку з метою забезпечення його відповідності вимогам коректності в цьому додатку.

До вимог коректності виконання відносяться вимоги по забезпеченню стійкості до відмов, незалежності і паралельності обробки, можливостям відновлення і т. д.

Процесам виконання властива семантика відкритою вкладеності, яка припускає доступність проміжних результатів процесу поза її меж та роздільна різних компонентів фіксувати результати своєї роботи в індивідуальному порядку. Компонентами можуть бути інші суб'єкти обробки, які характеризуються як семантикою відкритою вкладеності, так і семантикою закритих вкладених транзакцій, результати функціонування яких будуть доступні для всієї системи тільки після повного завершення роботи цього компонента. Слід зазначити, що компоненти з семантикою закритих вкладених транзакцій можуть складатися тільки з компонентів з семантикою того ж типу. Деякі з цих компонентів можуть розцінюватися як життєво важливі, і в разі відмови від їх виконання буде потрібно припинити виконання і їх батьківських компонентів. Крім того, можуть бути визначені компенсуючі і резервні транзакції, мова про які вже йшлося вище.

Питання:

8.1. Що мається на увазі під поняттям транзакції? Чому транзакції є найважливішим об'єктом управління в будь-який СУБД?

8.2. Поясніть аспекти узгодженості і надійності транзакцій, які впливають з визначення їх основних (ACID) властивостей. Опишіть кожне з цих властивостей і вкажіть їх зв'язок з механізмами управління паралельним виконанням і відновленням. Доповніть вашу відповідь конкретними прикладами.

8.3. Опишіть на конкретних прикладах ті типи проблем, які можуть мати місце в багатокористувацькому середовищі з паралельним доступом до бази даних.

8.4. Детально опишіть механізм управління паралельним виконанням, який може використовуватися для повного усунення всіх проблем управління паралельним доступом. За рахунок чого запропонований вами механізм дозволить виключити згадані проблеми? Як механізм управління паралельним виконанням взаємодіє з механізмом транзакцій?

8.5. Поясніть концепції послідовного, непослідовного і впорядкованого графіків. Сформулюйте правила еквівалентності графіків.

8.6. Які відмінності між конфліктної упорядкованістю і упорядкованістю за переглядом?

8.7. Які проблеми можуть виникати при використанні механізмів блокувань для управління паралельним виконанням і які дії можуть бути зроблені в СУБД для їх запобігання?

8.8. Чому схема двофазного блокування застосовується в якості схеми управління паралельним виконанням для індексів? Запропонуйте більш прийнятну схему блокування для деревовидних індексів.

8.9. Дайте визначення поняття тимчасової відмітки. У чому полягають відмінності між протоколом управління паралельним виконанням, заснованим на використанні тимчасових відміток, від протоколів, заснованих на використанні блокувань?

8.10. Опишіть основний протокол упорядкувань тимчасових відміток, застосовуваний для управління паралельним виконанням. Сформулюйте правило записи Томаса і вкажіть, який вплив воно робить на організацію цього основного протоколу упорядкування тимчасових відміток.

8.11. Опишіть способи використання численних версій елементів даних для підвищення ступеня розпаралелювання.

8.12. У чому полягає різниця між песимістичними і оптимістичними методами управління паралельним виконанням?

8.13. Визначте типи відмов, які можуть виникати в середовищі бази даних.

8.14. Чому для багатокористувацької СУБД необхідно передбачити механізми відновлення?

8.15. Чому журнал транзакцій є одним з основних засобів будь-якого механізму відновлення? Поясніть, що мається на увазі під прямим і зворотним відновленням, і опишіть, як файл журналу використовується в кожному з цих методів. У чому полягає важливість протоколу попереднім записом файлу журналу? Який вплив робить механізм створення контрольних точок на роботу протоколу відновлення?

8.16. Порівняйте і зіставте характеристики протоколів відновлення з відкладеним оновленням і безпосереднім оновленням.

8.17. Опишіть особливості наступних поліпшених моделей транзакцій:

- а) моделі вкладених транзакцій;
- б) моделі хронік;
- в) моделі багаторівневих транзакцій;
- г) моделі динамічної реструктуризації транзакцій.

Тема 9

Оптимізація запитів

9.1 Загальний огляд методів обробки запитів

Обробка запитів – дії, необхідні для отримання необхідної інформації з бази даних.

Метою виконання процедури обробки запитів є перетворення запиту, оформленого на мові високого рівня (зазвичай це мова SQL), в правильну і ефективну послідовність операцій, виражену на мові низького рівня, що реалізує операції реляційної алгебри. Потім підготовлений план обробки запиту виконується з метою вибірки необхідних даних.

Оптимізація запиту - процедура вибору найбільш ефективного плану виконання запиту, що поступив. Найважливішим моментом в обробці запитів є їх оптимізація.

Завдання оптимізації запитів полягає у визначенні трансформації запиту, яка вимагає мінімального використання системних ресурсів.

Методи оптимізації:

- скорочення загального часу виконання запиту, що представляє собою суму часу виконання всіх його окремих операцій;
- використання ресурсів, є показником, що оптимізуються з метою покращення часу реакції системи, так як фактично в даному випадку мова йде про максимально можливе розпаралелювання виконання операцій.

Оскільки основна проблема пов'язана зі складністю виконання обчислювальних операцій з великою кількістю відношень, стратегія, що обирається найчастіше зводиться до пошуку найближчого з можливих оптимальних рішень.

Обидва методи оптимізації запитів спираються на статистичні показники, що включають відомості про відношення, накопичені в базі даних. Ці показники використовуються для виконання обґрунтованого вибору серед всіх можливих варіантів. Точність і актуальність наявних статистичних даних має вирішальний вплив на загальну ефективність результатів оптимізації виконуваних обчислень.

Приклад 9.1. Порівняння різних методів обробки запитів.

Визначте імена всіх відмінників, які навчаються на потоці ТМ. Необхідний запит SQL може бути записаний в наступному вигляді:

```
SELECT *  
FROM СТУДЕНТ s, ГРУПА g  
WHERE s.НомерЗаліковки = g.НомерЗаліковки AND  
(s.Статус = 'Відмінник' AND g.НазваГрупи = 'ТМ');
```

Виходячи з правил еквівалентності виразів реляційної алгебри, наведений вище запит можна записати в наступних трьох варіантах:

1. $\sigma (\text{Статус} = \text{'Відмінник'}) \wedge (\text{НазваГрупи} = \text{'ТМ'}) \wedge (\text{СТУДЕНТ.НомерЗаліковки} = \text{ГРУПА.НомерЗаліковки}) (\text{СТУДЕНТ} \times \text{ГРУПА})$
2. $\sigma (\text{Статус} = \text{'Відмінник'}) \wedge (\text{НазваГрупи} = \text{'ТМ'}) (\text{СТУДЕНТ} \bowtie \text{СТУДЕНТ.НомерЗаліковки} = \text{ГРУПА.НомерЗаліковки} \text{ ГРУПА})$

3. (σ Статус = ' Відмінники ' (СТУДЕНТ)) $\bowtie$ СТУДЕНТ.НомерЗаліковки = ГРУПА.НомерЗаліковки (σ НазваГрупи = 'ТМ '(ГРУПА))

Щоб уточнити умови застосування даного прикладу, припустимо, що таблиця СТУДЕНТ містить 1000 рядків, а таблиця ГРУПА - 30 (всього груп в інституті). Далі візьмемо, що в інституті 30 відмінників (по 1 в кожній групі) і 10 з 30 груп - ТМ.

Порівняння трьох варіантів запису одного і того ж запиту проведемо з точки зору кількості операцій доступу до диска, виконуваних в кожному випадку. Для спрощення будемо вважати, що жодне з використовуваних відношень не має будь-яких індексів або ключів сортування, а результати будь-яких проміжних операцій завжди записуються на диск.

Операції запису на диск остаточних результатів виконання запиту ігноруються, оскільки їх кількість в кожному з трьох розглянутих варіантів однакова. Додатково припустимо, що в кожній дисковій операції здійснюється доступ тільки до одного рядка (хоча на практиці обмін з дисковими пристроями здійснюється блоками, що містять, як правило, відразу кілька рядків), а оперативна пам'ять комп'ютера досить велика, щоб розмістити всі відношення, що використовуються при виконанні будь-якої операції реляційної алгебри.

При обробці першого варіанту запиту обчислюється декартовий добуток таблиць СТУДЕНТ і ГРУПА, для чого потрібно $(1000 + 30)$ операцій доступу до диска при читанні вихідних відношень і виведення на диск результуючої таблиці, яка містить (1000×30) рядків. Потім потрібно знову зчитати з диску всі ці (1000×30) рядки, щоб перевірити кожен з них на відповідність заданому предикату.

В результаті загальна вартість обробки даного варіанту запиту складе: $(1000 + 30) + 2(1000 \times 30) = 61030$ дискових операцій.

У другому варіанті запиту спочатку здійснюється з'єднання таблиць СТУДЕНТ і ГРУПА по полю НомерЗаліковки, для чого буде потрібно виконати $1000 + 30$ звернень до диска при зчитуванні рядків вихідних відношень. Очевидно, що результуюча таблиця операції з'єднання буде налічувати 1000 рядків - по одній для кожного зі студентів інституту (оскільки кожен студент може бути приписаний тільки до однієї з груп). Подальша операція вибірки буде вимагати виконання 1000 звернень до диску з метою зчитування результатів операції з'єднання, тому загальна вартість виконання запиту складе:

$$2 \times 1000 + (1000 + 30) = 3030 \text{ дискових операцій.}$$

На першому етапі останнього варіанту обробки запиту кожен рядок таблиці СТУДЕНТ перевіряється для визначення того, чи містить вона відомості про відмінників, для цього буде потрібно виконати 1000 операцій читання. Результуючий набір даних складається всього з 30 рядків. На наступному етапі виконується операція вибірки з таблиці ГРУПА відомостей про ті групи, котрі ТМ. Зчитується 30 вихідних рядків, і записується результуюча таблиця з десяти рядків. На наступному етапі виконується з'єднання скорочених відношень СТУДЕНТ і ГРУПА, для чого потрібно $30 + 10$ операцій читання. Отже, загальна вартість обробки запиту складе:

$$1000 + 2 \times 30 + 10 + (30 + 10) = 1110 \text{ дискових операцій.}$$

Очевидно, що найбільш ефективним є третій варіант запису запиту, причому вартість його виконання в 55 разів менша за вартість першого варіанту.

Якщо збільшити кількість рядків в таблиці СТУДЕНТ до 10000, а число груп ГРУПА - до 300, то третій варіант запиту виявиться ефективніше першого в 550 разів.

Основна перевага третього варіанту обробки запиту полягає в тому, що розміри оброблюваних таблиць були істотно скорочені до виконання операції з'єднання.

Звідси можна зробити попередній висновок, що *при оптимізації запитів унарні операції вибірки і проєкції доцільно виконувати якомога раніше, оскільки це дозволяє зменшити розміри наборів даних, що беруть участь в бінарних операціях.*

Основні етапи обробки запитів

Процес обробки запитів може бути розділений на чотири основних етапи:

1. *декомпозиція (включає процедури синтаксичного аналізу і перевірки),*
2. *оптимізація,*
3. *генерація коду,*
4. *виконання.*

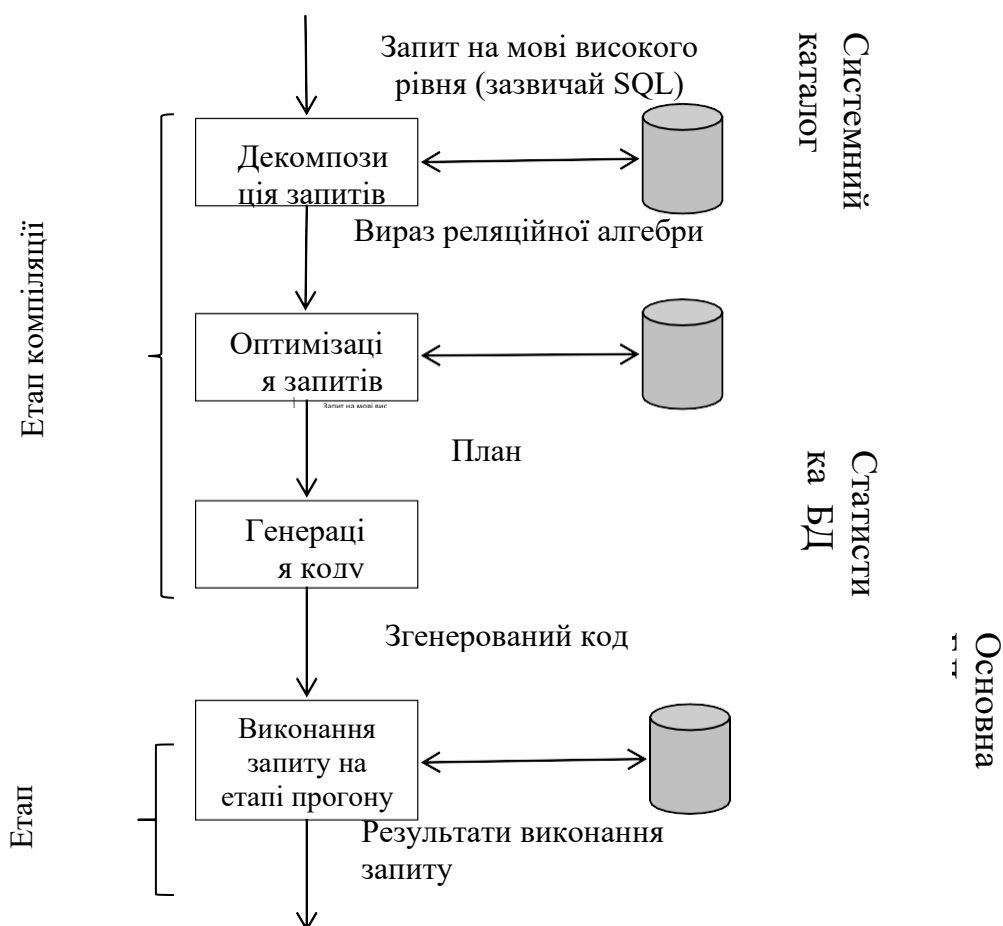


Рис.9.1. Основні етапи процесу обробки запитів

Динамічна та статична оптимізація запитів

Існують два варіанти виконання перших трьох етапів процесу обробки запитів. У першому випадку декомпозиція і оптимізація виконуються динамічно при кожному виклику запиту. Перевага *динамічної оптимізації запитів* полягає в

тому, що вся використовується в процесі вибору оптимальної стратегії інформація є актуальною саме на поточний момент часу. Основний недолік динамічної оптимізації полягає в відносному зниженні ефективності обробки запитів, оскільки етапи синтаксичного аналізу, перевірки та оптимізації повинні кожен раз передувати викликом запиту на виконання. Більш того, може знадобитися виключити з розгляду деякі можливі стратегії виконання для досягнення прийняттого рівня витрат, а це може привести до вибору менш оптимальної стратегії.

Альтернативним варіантом є *статична оптимізація запитів*, що передбачає одноразове виконання етапів синтаксичного аналізу, перевірки та оптимізації. Цей підхід аналогічний використанню компіляторів для обробки програм, написаних на різних мовах програмування.

Основна перевага статичної оптимізації полягає в усуненні додаткового навантаження через проведення процедур оптимізації в процесі виконання запиту. В результаті з'являється можливість більш ретельного підбору оптимальної стратегії за рахунок аналізу більшого числа можливих варіантів, тому ймовірність виявлення більш прийнятною стратегії значно зростає. У разі багаторазово виконуваних запитів додаткові витрати часу на пошук найоптимальніших планів їх виконання можуть дати суттєвий вииграш в продуктивності. Основний недолік цього методу полягає в тому, що обрана стратегія виконання запиту, яка була оптимальною при компіляції запиту, в момент виконання запиту може виявитися вже не оптимальною. Для подолання цього недоліку може застосовуватися комбінований підхід, який передбачає повторну оптимізацію запиту в тому випадку, коли система виявляє, що статистичні показники бази даних істотно змінилися з моменту попередньої оптимізації даного запиту. Альтернативний варіант передбачає автоматичну компіляцію запиту при першому його виконанні в кожному з сеансів з подальшим кешуванням оптимізованого плану запиту протягом всього часу існування даного сеансу. Це дозволяє одноразово виконувати оптимізацію кожного із запитів, які використовуються в окремому сеансі роботи з базою даних.

Декомпозиція запитів

Призначення етапу декомпозиції полягає в перетворенні запиту на мові високого рівня в вираз реляційної алгебри з подальшою перевіркою його синтаксичної і семантичної правильності. Зазвичай декомпозиція включає наступні стадії:

- аналіз,
- нормалізації,
- семантичного аналізу,
- спрощення,
- реструктуризації запиту.

Аналіз

На цій стадії виконується лексичний і синтаксичний аналіз запиту з використанням методів, що застосовуються компіляторами мов програмування високого рівня. Додатково уточнюється, чи присутні в системному каталозі визначення для зазначених у запиті відношень і атрибутів. Крім того, контролюється, чи можуть затребуванні у запиті операції над різними об'єктами бази даних, застосовуватися до об'єктів відповідного типу.

Приклад 9.2.

```
SELECT СтудНомер
FROM СТУДЕНТ
WHERE Статус > 10;
```

Виконання цього запиту буде відхилено за двох причин.

1. У списку вибірки запиту вказаний атрибут СтудНомер, відсутній у визначенні відношення СТУДЕНТ (в дійсності відповідний атрибут має ім'я НомерСтдента).

2. У конструкції WHERE операція порівняння «> 10» несумісна з типом атрибута Статус, який описаний в відношенні як рядок символів змінної довжини.

Після завершення даної стадії декомпозиції запит на мові високого рівня перетворюється на деяке внутрішнє представлення, більш зручне для виконання подальшої обробки. Як правило, для внутрішнього подання запиту вибирається та чи інша форма дерева запиту. Дерево запиту формується таким чином:

- Лист-вузли (що не мають нащадків) створюються для кожного використовуваного в запиті базового відношення.
- Звичайні вузли формуються для кожного проміжного відношення, створюваного в результаті виконання деякої операції реляційної алгебри.
- Корінь дерева являє результуючий набір запиту.
- Операції виконуються в напрямку від лист-вузлів до кореня дерева

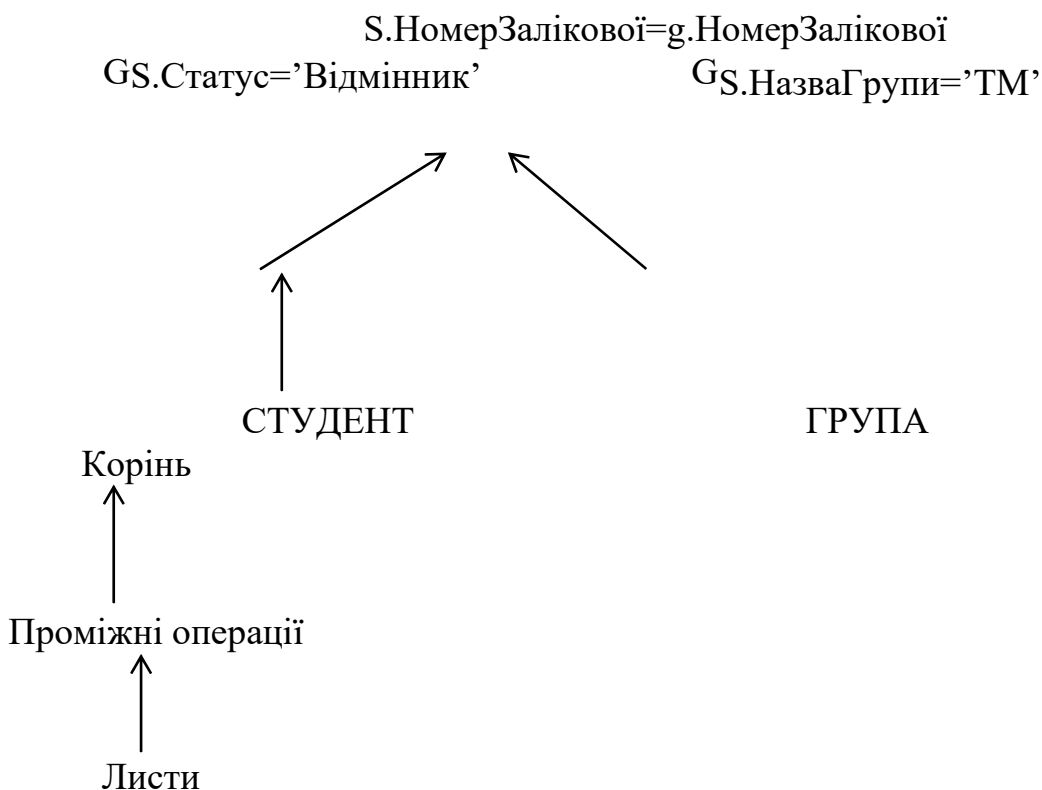


Рис.9.2. Приклад дерева реляційної алгебри

Нормалізація

При оптимізації запитів стадія нормалізації служить для перетворення запиту в нормалізовану форму, що виконується з метою спрощення його подальшої реструктуризації. На цій стадії предикати (в мові SQL вони визначаються як умови конструкції WHERE), які можуть являти собою вираження довільного ступеня складності, перетворюються в одну з двох стандартних форм. Здійснюється ця

операція за допомогою застосування до предикатам декількох перерахованих нижче правил перетворення:

- *Кон'юнктивна нормальна форма.* Являє собою послідовність кон'юнкцій, пов'язаних між собою операторами диз'юнкції $\vee$ (OR). Кожна кон'юнкція містить одне або кілька виразів, з'єднаних операторами кон'юнкції $\wedge$ (AND).

Наприклад:

$(\text{Статус} = \text{'Відмінник'} \vee \text{ПБал} > 4,8) \wedge \text{НазваГрупи} = \text{'ТМ'}$

Вибірка, виконана на основі кон'юнктивної предикати, містить тільки ті рядки, які задовольняють всім, хто входить в предикат кон'юнкції.

- *Диз'юнктивна нормальна форма.* Являє собою послідовність диз'юнкцій, з'єднаних між собою операторами кон'юнкції $\wedge$ (AND). Кожна диз'юнкція містить одне або кілька виразів, з'єднаних операторами кон'юнкції $\vee$ (OR).

Наприклад:

$(\text{Статус} = \text{'Відмінник'} \wedge \text{НазваГрупи} = \text{'ТМ'}) \vee (\text{ПБал} > 4,8 \wedge \text{НазваГрупи} = \text{'ТМ'})$

Вибірка, виконана на основі диз'юнктивного предиката, містить об'єднання всіх рядків, які задовольняють будь-який з вхідних в предикат диз'юнкцій.

Семантичний аналіз

Метою проведення семантичного аналізу є відхилення нормалізованих запитів, які некоректно сформульовані або містять суперечливі вимоги. Запит вважається некоректно сформульованим, якщо його компоненти не беруть участь у формуванні результуючого набору. Найчастіше така ситуація виникає тоді, коли відсутні визначення необхідних операцій з'єднання. Запит вважається суперечливим, якщо його предикату не відповідає ні один рядок. Наприклад, для відношення СТУДЕНТ предикат $(\text{Статус} = \text{'Відмінник'} \wedge \text{Статус} = \text{'Незадовільно'})$ є суперечливим, оскільки ніякої студент не може одночасно бути круглим відмінником і незадовільно. З цієї причини предикат $((\text{Статус} = \text{'Відмінник'} \wedge \text{Статус} = \text{'Незадовільно'}) \wedge \text{ПБал} > 4,8)$ може бути спрощений до варіанту $(\text{ПБал} > 4,8)$, оскільки його суперечливу фразу можна просто замінити логічним значенням FALSE. На жаль, функція виявлення і обробки суперечливих конструкцій підтримується не всіма існуючими СУБД.

Алгоритм визначення правильності існує тільки для підмножини запитів, які містять операції диз'юнкції і заперечення. По відношенню до зазначеного підмножини питань можуть застосовуватися такі перевірки:

1. *Побудова графа з'єднання відносин.* Якщо елементи графа виявляються роз'єднаними, формулювання запиту є невірною. При побудові графа з'єднання відношень спочатку створюється окремий вузол для кожного використовуваного в запиті відносини, а також для результуючої таблиці запиту. Потім між парами вузлів проводяться дуги представляють операції з'єднання, і дуги між окремими вузлами які є джерелом даних для операцій проекції.

2. *Побудова графа з'єднань нормалізованих атрибутів.* Якщо граф включає цикл, для якого сума оцінок є негативною, то запит містить суперечливі умови. При побудові графа з'єднань нормалізованих атрибутів спочатку створюються окремі вузли для кожного посилання на атрибут або константу 0. Потім між вузлами проводяться спрямовані дуги, що представляють операції з'єднання, а також проводяться спрямовані дуги між вузлами атрибутів і констант 0, які представляють

операції вибірки. Далі кожному ребру $a \rightarrow b$ присвоюється вагове значення c , якщо воно являє умова, записана у вигляді нерівності ($a \leq b + c$), а кожному ребру $0 \rightarrow a$ присвоюється значення $-c$, якщо воно являє нерівність типу ($a \geq c$).

Спрощення

Метою виконання процедури спрощення є виявлення надлишкових уточнень, виключення загальних підвиразів і перетворення запиту в семантично еквівалентну форму, що дозволяє спростити необхідні обчислення або підвищити ефективність їх виконання. На цій же стадії зазвичай розглядаються існуючі обмеження доступу, розкриваються визначення використовуваних представлень і враховуються обмеження цілісності даних, тобто виконуються ті процедури, які також можуть стати джерелом надмірності умов запиту. Якщо користувач не має необхідних прав доступу до всіх без винятку компонентів запиту, виконання цього запиту скасовується. Після отримання підтвердження того, що користувач має всі необхідними правами доступу, текст запиту піддається попередній оптимізації на основі широко відомих правил еквівалентності реляційної алгебри:

$p \wedge (p) = p$
 $p \vee (p) = p$
 $p \wedge \text{false} = \text{false}$
 $p \vee \text{false} = p$
 $p \wedge \text{true} = p$
 $p \vee \text{true} = \text{true}$
 $p \wedge (\sim p) = \text{false}$
 $p \vee (\sim p) = \text{true}$
 $p \wedge (p \vee q) = p$
 $p \vee (p \wedge q) = p$

Приклад 9.3.

Розглянемо визначення уявлення і запит, який звертається до даних в цьому поданні:

```
CREATE VIEW Студент3
AS SELECT НомерЗаліковки, НазваГрупи, ПІБ Студента, ПБал
FROM СТУДЕНТ
WHERE НазваГрупи = 'ТМ'
SELECT *
FROM Студент3
WHERE (НазваГрупи = 'ТМ' AND ПБал > 4.8) ;
```

На етапі розкриття представлення цей запит буде перетворений в такий спосіб:

```
SELECT НомерЗаліковки, ПІБ Студента, ПБал
FROM СТУДЕНТ
WHERE (НазваГрупи = 'ТМ'
AND ПБал > 4.8)
AND НазваГрупи = 'ТМ';
```

Текст умови WHERE даного запиту можна скоротити до виду (НазваГрупи = 'ТМ' AND ПБал > 4,8).

Встановлені в системі обмеження цілісності даних також можуть сприяти спрощенню виду запитів. Наприклад, введемо таке обмеження цілісності, яке гарантує, що тільки студенти типу Відмінник мають прохідний бал вище 4.8:

```
CREATE ASSERTION ТількиНайвищаОцінка  
CHECK ((Статус <> 'Відмінник' AND ПБал <4.8)  
OR (Статус = 'Відмінник' AND ПБал > 4.8) );
```

Потім визначимо, до якого результату приводить виконання наступного запиту:

```
SELECT *  
FROM СТУДЕНТ  
WHERE (Статус = 'Відмінник' AND ПБал <3.8);
```

Предикат, заданий в конструкції WHERE даного запиту, в якому виконується пошук даних про студента-відмінника з прохідним балом менше 3.8, суперечить введеному вище обмеженню цілісності, тому йому не відповідає ні один рядок.

Реструктуризація запитів

На фінальній стадії етапу декомпозиції запит перетворюється таким чином, щоб забезпечити найбільш ефективну його реалізацію.

9.2 Евристичний підхід до оптимізації запитів

У цьому розділі розглядаються евристичні методи оптимізації запитів, які передбачають використання правил перетворення для трансформації вираження реляційної алгебри в деяку еквівалентну форму, обробка якої буде свідомо більш ефективною. Більш ефективно виконувати операції вибірки з таблиць до того, як ці таблиці будуть використані в операціях з'єднання, ніж виконувати ці ж дії в зворотному порядку. Далі показано, що існують певні правила перетворення, що дозволяють змінювати порядок виконання операцій з'єднання і вибірки таким чином, щоб операції вибірки виконувалися в першу чергу. Після обговорення того, які види перетворення запитів є допустимими, далі наведено набір евристичних правил, які свідомо дозволяють отримати досить хороші стратегії виконання запитів, хоча і не завжди оптимальні.

Правила перетворення операцій реляційної алгебри

Правила будуть використовуватися для реструктуризації (канонічного) дерева реляційної алгебри, побудованого в процесі декомпозиції запиту. Для представлення цих правил у вигляді формул скористаємося трьома відношеннями - R, S і T, причому відношення R визначено на наборі атрибутів $A = \{A_1, A_2, \dots, A_n\}$, а відношення S - на наборі атрибутів $B = \{B_1, B_2, \dots, B_n\}$. Позначення p, q і r будуть використані для представлення предикатів, а позначення L, L1, L2, M, M1, M2 і N - для представлення наборів атрибутів.

1. *Операція вибірки з кон'юнктивним предикатом може бути перетворена в послідовність операцій вибірки по членам кон'юнкції (і навпаки).*

$$\sigma_{p \wedge q \wedge r}(R) = \sigma_p(\sigma_q(\sigma_r(R)))$$

Цей метод перетворення іноді називають *каскадною вибіркою*.

Наприклад:

$$\sigma_{\text{НазваГрупи} = \text{'ТМ'} \wedge \text{ПБал} > 4.7}(\text{СТУДЕНТ}) = \sigma_{\text{НазваГрупи} = \text{'ТМ'}}(\sigma_{\text{ПБал} > 4.7}(\text{СТУДЕНТ}))$$

2. Правило коммутативності операцій вибірки.

$$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$$

Наприклад:

$$\sigma_{\text{НазваГрупи} = \text{'ТМ'}}(\sigma_{\text{ПБал} > 4.7}(\text{СТУДЕНТ})) = \sigma_{\text{ПБал} > 4.7}(\sigma_{\text{НазваГрупи} = \text{'ТМ'}}(\text{СТУДЕНТ}))$$

3. У послідовності операцій проєкції необхідна тільки остання з операцій.

$$\Pi_L \Pi_M \dots \Pi_N(R) = \Pi_L(R)$$

Наприклад:

$$\Pi_{\text{ПІБ Студента}} \Pi_{\text{НазваГрупи}}(\text{СТУДЕНТ}) = \Pi_{\text{ПІБСтудента}}(\text{СТУДЕНТ})$$

4. Правило комутативності операцій вибірки і проєкції.

Якщо предикат p включає тільки атрибути, що входять в список проєкції, то операції вибірки і проєкції будуть мати властивість комутативності: $\Pi_{A_1} \dots \Pi_{A_n}(\sigma_p(R)) = \sigma_p(\Pi_{A_1} \dots \Pi_{A_n}(R))$ де $p \in \{A_1, A_2, \dots, A_n\}$

Наприклад:

$$\Pi_{\text{ДомТел, МобТел}}(\sigma_{\text{МобТел} = \text{'80504253245'}}(\text{СТУДЕНТ})) = \sigma_{\text{МобТел} = \text{'80504253245'}}(\Pi_{\text{ДомТел, МобТел}}(\text{СТУДЕНТ}))$$

5. Правило комутативності операції тета-з'єднання (і декартова множення).

$$R \bowtie_p S = S \bowtie_p R$$

$$R \times S = S \times R$$

Оскільки операції з'єднання по еквівалентності і природного з'єднання є особливими випадками операції тета-з'єднання, наведене вище правило застосовується і до цих двох операцій з'єднання. Наприклад, для операції з'єднання по еквівалентності відношень **СТУДЕНТ** і **ГРУПА** справедливо наступне твердження:

$$\text{СТУДЕНТ} \bowtie_{\text{СТУДЕНТ.НомерЗаліковки} = \text{ГРУПА.НомерЗаліковки}} \text{ГРУПА} = \text{ГРУПА} \bowtie_{\text{СТУДЕНТ.НомерЗаліковки} = \text{ГРУПА.НомерЗаліковки}} \text{СТУДЕНТ}$$

6. Правило комутативності операцій вибірки і тета-з'єднання (або декартовий добуток).

Якщо предикат операції вибірки включає атрибути тільки одного із з'єднуваних відносин, то операції вибірки і з'єднання (або декартовий; множини) будуть мати властивість комутативності: $\sigma_p(R \times S) = (\sigma_p(R)) \times S$ де $p \in \{A_1, A_2, \dots, A_n\}$ В альтернативному випадку, коли предикат операції вибірки являє собою кон'юнкцію предикатів виду $(p \wedge q)$, де предикат p включає атрибути тільки відносини R , а предикат q - атрибути тільки відносини S , операції вибірки і тета-з'єднання будуть володіти таким варіантом властивості комутативності:

$$\sigma_{p \wedge q}(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r (\sigma_q(S))$$

$$\sigma_{p \wedge q}(R \times S) = (\sigma_p(R)) \times (\sigma_q(S))$$

Наприклад:

$$\sigma_{\text{Статус} = \text{'Відмінник'}} \wedge \text{НазваГрупа} = \text{'ТМ'}$$

$(\text{СТУДЕНТ} \triangleright \triangleleft \text{СТУДЕНТ} . \text{НомерЗаліковки} = \text{ГРУПА} . \text{НомерЗаліковки} \text{ГРУПА})$
 $= (\sigma_{\text{Статус}} = \text{'Відмінник'} (\text{СТУДЕНТ})) \triangleright \triangleleft \text{СТУДЕНТ} . \text{НомерЗаліковки} =$
 $\text{ГРУПА} . \text{НомерЗаліковки} (\sigma_{\text{НазваГрупи}} = \text{'ТМ'} (\text{ГРУПА}))$

7. *Правило коммутативності операцій проекції і тета-з'єднання* (або декартова множини).

Якщо список атрибутів операції проекції має вигляд $L = L_1 L_2$, де в підписок L_1 входять атрибути тільки відносини R , а в підписок L_2 - атрибути тільки відносини S , то в тому випадку, якщо умова з'єднання містить тільки атрибути відносини L , операції проекції і тета - з'єднання будуть володіти таким варіантом властивості комутативності:

$$\Pi_{L_1 \cup L_2}(R \triangleright \triangleleft_r S) = (\Pi_{L_1}(R) \triangleright \triangleleft_r (\Pi_{L_2}(S)))$$

Наприклад:

$\Pi_{\text{Статус, НазваГрупи, НомерЗаліковки}}(\text{СТУДЕНТ} \triangleright \triangleleft \text{СТУДЕНТ} . \text{НомерЗаліковки} = \text{ГРУПА} . \text{НомерЗаліковки} \text{ГРУПА})$
 $=$

$(\Pi_{\text{Статус, НомерЗаліковки}} (\text{СТУДЕНТ}) \triangleright \triangleleft \text{СТУДЕНТ} . \text{НомерЗаліковки} = \text{ГРУПА} . \text{НомерЗаліковки})$
 $(\Pi_{\text{НазваГрупи, НомерЗаліковки}} (\text{ГРУПА}))$

Якщо умова з'єднання містить додаткові атрибути, що не входять в відношення L , наприклад атрибути $M = M_1 \cup M_2$, де список M_1 містить атрибути тільки відносини R , а список M_2 - тільки атрибути відносини S , то додатково буде потрібно виконати завершальну операцію проекції:

$$\Pi_{L_1 \cup L_2}(R \triangleright \triangleleft_r S) = \Pi_{L_1 \cup L_2} ((\Pi_{L_1 \cup M_1}(R)) \triangleright \triangleleft_r (\Pi_{L_2 \cup M_2}(S)))$$

Наприклад:

$\Pi_{\text{Статус, НазваГрупи}} (\text{СТУДЕНТ} \triangleright \triangleleft \text{СТУДЕНТ} . \text{НомерЗаліковки} =$
 $\text{ГРУПА} . \text{НомерЗаліковки})$

$\text{ГРУПА} = \Pi_{\text{Статус, НазваГрупи}} ((\Pi_{\text{Статус, НомерЗаліковки}} (\text{СТУДЕНТ}))$
 $\text{НомерЗаліковки} = \text{ГРУПА} . \text{НомерЗаліковки} (\Pi_{\text{НазваГрупи, НомерЗаліковки}} (\text{ГРУПА})))$

8. *Правило комутативності операцій об'єднання і перетину* (але не різниці множин).

$$R \cup S = S \cup R$$

$$R \cap S = S \cap R$$

9. *Правило комутативності операції вибірки і операція над множинами* (об'єднання, перетин і різницю множин).

$$\sigma_p(R \cup S) = \sigma_p(S) \cup \sigma_p(R)$$

$$\sigma_p(R \cap S) = \sigma_p(S) \cap \sigma_p(R)$$

$$\sigma_p(R - S) = \sigma_p(S) - \sigma_p(R)$$

10. *Правило комутативності операцій проекції та об'єднання* .

$$\Pi_L(R \cup S) = \Pi_L(S) \cup \Pi_L(R)$$

11. *Правило асоціативності операції тета-з'єднання* (і декартового добутку). Операції декартового добутку і природного з'єднання завжди асоціативні:

$$(R \triangleright \triangleleft S) \triangleright \triangleleft T = R \triangleright \triangleleft (S \triangleright \triangleleft T)$$

$$(R \times S) \times T = R \times (S \times T)$$

Якщо умова з'єднання q включає атрибути тільки з відношень S і T , то операція тета-з'єднання має властивість асоціативності такого вигляду :

$$(R \bowtie_p S) \bowtie_{q \wedge r} T = R \bowtie_{p \wedge r} (S \bowtie_q T)$$

12. *Правило асоціативності операцій об'єднання і перетину* (але не операції різниці множин).

$$(R \cup S) \cup T = S \cup (R \cup T)$$

$$(R \cap S) \cap T = S \cap (R \cap T)$$

Стратегії евристичної обробки запитів

У багатьох СУБД для вибору стратегії обробки запитів використовуються ті чи інші евристичні правила. Правила, які найчастіше застосовуються на практиці для оптимізації обробки запитів:

1. *Виконання операцій вибірки на самих початкових етапах обробки.*

Виконання операції вибірки зменшує кардинальність відношення і скорочує час наступних операцій обробки запиту. Тому доцільно почати оптимізацію з використання правила 1 для формування каскадної послідовності операцій вибірки, після чого слід застосовувати правила 2, 4, 6 і 9, щоб поміняти місцями унарні і бінарні операції вибірки. Мета цих маніпуляцій полягає в тому, щоб перемістити операцію вибірки в дереві реляційної алгебри якнайнижче. Крім того, корисно зберігати разом все предикати, що стосуються одного і того ж відношення.

2. Об'єднання в одну операцію з'єднання операції декартового добутку і наступної за нею операції вибірки, предикат якої представляє умова з'єднання.

Вище вже зазначалося, що операція вибірки з предикатом типу тета-з'єднання і операція декартового добутку можуть бути представлені однією операцією тета-з'єднання.

$$\sigma_{R.a \Theta S.b} (R \times S) = R \bowtie_{R.a \Theta S.b} S$$

3. Використання властивості асоціативності бінарних операцій для перевпорядкування лист-вузлів таким чином, що лист-вузли з найбільш обмежувальними операціями вибірки будуть виконуватися в першу чергу.

Слід пам'ятати, що основне емпіричне правило підвищення ефективності полягає в максимально можливому скороченні обсягу результуючого набору перед виконанням бінарних операцій.

Припустимо, що потрібно виконати дві послідовні операції з'єднання:

$$(R \bowtie_{R.a \Theta S.b} S) \bowtie_{S.c \Theta T.d} T$$

У цьому випадку можна скористатися правилами 11 і 12, що описують асоціативні властивості операцій тета-з'єднання (а також об'єднання і перетину). Мета полягає в перевпорядкуванні операцій таким чином, щоб першим виконувалося з'єднання, в результаті якого буде отримано менше за розміром результуючого відношення. Це дозволить виконати друге з'єднання з меншим за розміром вихідним ставленням.

4. Як можна більш раннє виконання операцій проекції.

Слід пам'ятати : операції проєкції зменшують кардинальність відношення, що призводить до скорочення обсягів даних, що підлягають подальшій обробці.

Доцільно спочатку використовувати правило 3 для побудови вихідної послідовності операцій проєкції, після чого застосувати до них правила 4, 7 і 10, що описують властивості комутативності операцій проєкції з бінарними операціями. Метою даної процедури є переміщення операцій проєкції вниз по дереву реляційної алгебри (наскільки це можливо). Крім того, рекомендується об'єднувати операції проєкції, що виконуються над одним і тим же відношенням.

5. Одноразове обчислення загальних виразів.

Якщо один і той же вираз зустрічається в дереві реляційної алгебри кілька разів і результат його обчислення не надто великий, доцільно зберегти цей результат після його першого обчислення з метою повторного використання в міру необхідності. Однак вигащ буде отриманий тільки в тому випадку, якщо розмір результуючої таблиці загального вираження досить малий, щоб зберігати його в оперативній пам'яті, або вартість його вибірки із зовнішньої пам'яті буде менше вартості його повторного обчислення. Цей підхід може виявитися особливо ефективним при роботі з представленнями, оскільки при кожному зверненні до деякого поданням повинні обчислюватися одні й ті самі вирази.

9.3 Оцінка вартості операцій реляційної алгебри

СУБД може підтримувати безліч різних способів реалізації операцій реляційної алгебри. Призначення всього процесу оптимізації запитів полягає у виборі найбільш ефективного способу їх обробки. Для цього система використовує формули, що дозволяють оцінити вартість декількох різних варіантів обробки, після чого вибирає той з них, вартість якого мінімальна.

Статистичні показники бази даних

Достовірність оцінки обсягу і вартості виконання проміжних операцій реляційної алгебри в значній мірі визначається повнотою і актуальністю статистичної інформації, що накопичується в СУБД. Як правило, типова СУБД зберігає в своєму системному каталозі статистичні дані, перераховані нижче.

Характеристики кожного базового відношення R

- nTuples (R). Кількість рядків (записів) у відношенні R (тобто його кардинальність).
- bFactor (R). Показник блокування відношення R (тобто кількість рядків відношення R, що розміщуються в одному дисковому блоці).
- nBlocks (R). Кількість блоків вторинної пам'яті, необхідних для збереження відношення R. Якщо рядки відношення R фізично розміщуються разом, то справедлива наступна формула:
$$nBlocks(R) = [nTuples(R) / bFactor(R)]$$

Характеристики кожного атрибута A базового відношення R

- nDistinctA (R). Кількість унікальних значень атрибута A щодо R

- $\min A(R)$, $\max A(R)$. Мінімальна і максимальна можливі значення атрибута A щодо R .
- $SCA(R)$. Кардинальність вибірки атрибута A щодо R . Ця характеристика являє собою середню кількість рядків, які задовольняють умові рівності, заданому для атрибута A . Якщо припустити, що значення атрибута A рівномірно розподілені по відношенню до R і що існує принаймні одне значення, яке задовольняє заданій умові, то значення $SCA(R)$ може бути визначено із застосуванням формул, наведених в табл. 9.1.

Таблиця 9.1. Формули визначення кардинальності вибірки атрибута A щодо R з урахуванням умови рівності

Формула	Умова
$SCA(R) = 1$	Атрибут A є ключовим атрибутом відношення R
$SC_A(R) = [nTuples(R)/nDistinct_A(R)]$	Атрибут A не є ключовим атрибутом відношення R

Крім того, може бути виконана оцінка кардинальності вибірки для різних умов. У цьому випадку значення $SCA(R)$ може бути визначено за допомогою формул, наведених в табл.2.

Таблиця 9.2. Формули визначення кардинальності вибірки атрибута A щодо R з урахуванням інших умов

Формула	Умова
$SC_A(R) = [nTuples(R) \times ((\max_A(R) - c) / (\max_A(R) - \min_A(R)))]$	Нерівність ($A > c$)
$SC_A(R) = [nTuples(R) \times (c - \max_A(R)) / (\max_A(R) - \min_A(R))]$	Нерівність ($A < c$)
$SC_A(R) = [(nTuples(R)/nDistinct_A(R)) \times n]$	Приналежність до множини ($A \in \{c_1, c_2, \dots, c_n\}$)
$SC_A(R) = SC_A(R) \times SC_B(R)$	Перетин ($A \wedge B$)
$SC_A(R) = SC_A(R) + SC_B(R) - SC_A(R) \times SC_B(R)$	Об'єднання ($A \vee B$)

Характеристики кожного багаторівневого індексу I набору значень атрибута A

- nLevelsA (I). Кількість рівнів індексу I.
- nLfBlocksA (I). Кількість лист- блоків індексу I.

Підтримка всіх цих статистичних показу ялин в актуальному стані може являти собою серйозну проблему. Якщо СУБД буде оновлювати статистичні показники кожного разу при вставці, оновленні або видаленні рядка, то це надасть дуже істотний вплив на загальну продуктивність системи в періоди її пікового навантаження. Тому застосовують два способи:

- процедури поновлення статистичних даних виконуються або з встановленою періодичністю (наприклад, щоночі), або в моменти простою системи;
- відповідальність за визначення того моменту, коли має бути виконано оновлення статистичних даних, покладається на користувача.

Операція вибірки ($S = \sigma_r(R)$)

Нагадування: в реляційній алгебрі операція вибірки виконується над одним відношенням, наприклад R. У результаті її виконання створюється нове відношення S, що містить тільки ті рядки відношення R, які задовольняють вказаному предикату. Предикат може бути простим, що включає єдину операцію порівняння атрибута відношення R з деякою константою, або значенням іншого атрибута. Предикат може також бути складовим, що включає кілька умов, з'єднаних за допомогою логічних операцій $\wedge$ (AND), $\vee$ (OR) и $\neg$ (NOT).

Існує кілька різних способів реалізації операції вибірки, доцільність застосування яких залежить від структури файлу, в якому зберігається відношення, а також від того, якими є використовувані в предикаті атрибута - індексованими або хешованими. Нижче перераховані основні типи стратегій:

- Лінійний пошук (файл не відсортований, індекс відсутній).
- Двійковий пошук (впорядкований файл, індекс відсутній).
- Пошук по рівності ключа хешування.
- Умова рівності, що застосовується до первинного ключа.
- Умова нерівності, що застосовується до первинного ключа.
- Умова рівності, що застосовується до кластеризуючого (вторинного) індексу.
- Умова рівності, що застосовується до некластеризуючого (вторинного) індексу.
- Умова нерівності, що застосовується до вторинного індексу типу B + -Tree.

Формули розрахунку вартості для всіх зазначених типів стратегій наведені в табл. 9.3.

Таблиця 9.3. Зведені дані по оцінці вартості операцій введення-виведення для різних стратегій виконання операцій вибірки

Стратегія	Оцінка вартості
Лінійний пошук (неупорядкований файл без індексу)	$[nBlocks(R)/2]$ – в випадку пошуку за рівністю ключового атрибута;

Двійковий пошук (впорядкований файл без індексу)	$nBlocks(R)$ – в іншому випадку $\lceil \log_2(nBlocks(R)) \rceil$ – у випадку пошуку за рівністю ключового атрибуту; $\lceil \log_2(nBlocks(R)) \rceil + \lceil SC_A(R)/bFactor(R) \rceil - 1$ – в іншому випадку
Пошук за рівністю ключу хешування	1 – при умові переповнення
Пошук за рівністю первинному ключа	$nLevels_A(I) + 1$
Пошук за нерівністю первинного ключа	$nLevels_A(I) + \lceil nBlocks(R)/2 \rceil$
Пошук за рівністю значення кластерного (вторинного) індексу	$nLevels_A(I) + \lceil SC_A(R)/bFactor(R) \rceil$
Пошук за рівністю значення некластерізуючого (вторинного) індексу	$nLevels_A(I) + \lceil SC_A(R) \rceil$
Пошук за нерівністю значення вторинного індексу типу B ⁺ -Tree	$nLevels_A(I) + \lceil nLfBlocks_A(I)/2 + nTuples(R)/2 \rceil$

Оцінка кардинальності операції вибірки

Перш ніж приступати до розгляду особистих стратегій виконання операції вибірки, доцільно познайомитися з методами оцінки очікуваної кількості рядків і унікальних значень атрибута в результуючому відношенні S, отриманому при виконанні операції вибірки щодо R. Отримання точних значень зазвичай є досить складним завданням. Однак якщо взяти стандартне спрощене припущення про те, що значення атрибута рівномірно розподілені в просторі його домену і незалежні один від одного, то можна скористатися наступними формулами:

$\lceil nTuples(S) \rceil = SC_A(R)$ предикат p має вигляд $(A \theta x)$

Для будь-якого атрибуту $v \neq A$ щодо S значення $nDistinctB(S)$ дорівнюватиме:

- $nTuples(S)$, якщо $nTuples(S) < nDistinctB(R) / 2$;
- $\lceil (nTuples(S) + nDistinctB(R)) / 3 \rceil$, якщо $nDistinctB(R) / 2 \leq nTuples(S) \leq nDistinctB(R)$;
- $nDistinctB(R)$, якщо $nTuples(S) > nDistinctB(R)$.

Можна отримати більш точні оцінки, якщо відмовитися від припущення про рівномірний розподіл значень атрибуту, проте в цьому випадку буде потрібно використовувати більш детальну статистичну інформацію, наприклад гістограми і кроки розподілу.

Варіант 1. Лінійний пошук (невпорядкований файл без індексу)

У даному випадку може знадобитися виконати перегляд кожного рядка в кожному дисковому блоці, щоб перевірити її на відповідність заданому предикату. Цей метод ілюструється алгоритмом, представленим в лістингу 9.1.

Такий спосіб пошуку іноді називають *повним переглядом таблиці*. Якщо пошук виконується за умовою рівності ключовому атрибуту відношення і можна припустити, що рядки рівномірно розподілені в файлі, то в середньому до

знаходження необхідної рядки потрібно вважати половину блоків файлу. Тому формула оцінки вартості матиме вигляд:

$$[nBlocks(R) / 2]$$

Для будь-яких інших умов можна обґрунтовано прийняти, що потрібно вважати весь файл, тому формула оцінки вартості матиме такий вигляд: $nBlocks(R)$

Лістинг 9.1

. Алгоритм лінійного пошуку

//

// Лінійний пошук.

// predicate - це ключ пошуку.

//Файл не відсортований.Блоки послідовно пронумеровані,починаючи з 1.

// Повертається результуюча таблиця, що містить ті рядки відношення R,

// які відповідають предикату predicate

//

for i = 1 to nBlocks (R) { // Цикл по всіх блокам файлу

block = read_block (R , i);

for j = 1 to nTuples (block) { // Цикл по всіх рядках в блоці i

if (block.tuple [j] <задовольняє предикату predicate>)

then <помістити рядок в результуючу таблицю>;

}

}

Варіант 2. Двійковий пошук (упорядкований файл без індексу)

Якщо предикат має вигляд ($A = x$) і файл впорядкований за значеннями атрибута A, який одночасно є ключовим атрибутом відносини R, формула для оцінки вартості операції вибірки буде мати наступний вигляд: $[\log_2 (nBlocks(R))]$

Алгоритм для виконання даного типу пошуку представлений в лістингу 9.2. У більш загальному випадку оцінка вартості може бути виконана за іншою формулою:

$$[\log_2 (nBlocks(R))] + [SCA(R) / bFactor(R)] - 1$$

Тут перший вираз відображає вартість пошуку першої з рядка, для чого використовується метод довічного пошуку. Передбачається, що предикату відповідають $SCA(R)$ рядків, розташованих в $[SCA(R) / bFactor(R)]$ блоках файлу, один з яких вже був лічений при виконанні довічного пошуку першого рядка.

Лістинг 9.2. Алгоритм двійкового пошуку в упорядкованому файлі //

// Двійковий пошук.

// predicate - це ключ пошуку.

// Файл впорядкований по зростанню значень ключового поля A.

// Файл складається з nBlocks блоків, послідовно пронумерованих, починаючи

// з 1. Повертається логічна змінна (found), яка вказує, чи була знайдена

// запис, яка відповідає предикату, а також результуюча таблиця, якщо

```

// така існує
//
next = 1; last = nBlocks; found = FALSE; keep_searching = TRUE; while (last >=
1 and (not found) and (keep_searching)) {i = (next + last) / 2; // Поділ обстежуваної
області навпіл
block = read_block (R, i);
if (predicate < ordering_key_field (first_record (block)))
then // Запис в нижній половині обстежуваної області
last = i - 1;
else if (predicate > ordering_key_field (last_record (block)))
then // Запис у верхній половині обстежуваної області
next = i + 1;
else if (check_block_for_predicate (block, predicate, result))
then // Необхідно запис в даному блоці
found = TRUE;
else // Необхідний запис відсутній
keep_searching = FALSE;
}

```

Варіант 3. Пошук по рівності ключу хешування Якщо атрибут A є ключем хешування файлу, то для обчислення адреси призначення рядка використовується алгоритм хешування. Якщо файл не має переповнень блоків, то очікувана вартість вибірки дорівнюватиме 1. Якщо переповнення мають місце, то для доступу до даних можуть знадобитися додаткові дискові операції, число яких залежить від кількості переповнень і обраного методу їх обробки.

Варіант 4. Пошук по рівності первинному ключу

Якщо предикат включає умову рівності значенням в полі первинного ключа ($A = x$), то для вибірки єдиного запису, що задовольняє заданій умові, може використовуватися первинний індекс. У цьому випадку буде потрібно рахувати кількість блоків, на одиницю більше кількості операцій читання індексу, яке буде дорівнює числу його рівнів. Тому формула обчислення оцінки вартості матиме такий вигляд:

$$nLevelsA(I) + 1$$

Варіант 5. Пошук за нерівністю значення первинного ключа

Якщо предикат включає умову нерівності значенням в полі первинного ключа A ($A < x$, $A \leq x$, $A > x$, $A \geq x$), то спочатку потрібно скористатися індексом для пошуку запису, що задовольняє предикату $A = x$. Якщо значення в індексі відсортовані, необхідні записи можуть бути вилучені за допомогою доступу до всіх записів індексу, розташованим вище або нижче знайденої. Якщо прийняти, що записи розподілені в файлі рівномірно, то заданій умові нерівності буде задовольняти половина записів файлу. Тому в даному випадку формула оцінки вартості вибірки буде мати наступний вигляд:

$$nLevelsA(I) + [nBlocks(R) / 2]$$

Варіант 6. Пошук за рівністю значення кластеризуючого (вторинного) індексу

Якщо предикат включає умову рівності значенням атрибута A, який не є первинним ключем, але являє собою кластеризуючий вторинний індекс, то цей

індекс може використовуватися для вибірки необхідних записів. Тому формула оцінки вартості матиме такий вигляд:

$nLevelsA(I) + [SCa(R) / bFactor(R)]$

Другий вираз в цій формулі являє собою оцінку кількості блоків, які будуть потрібні для збереження всіх рядків, що задовольняють умові рівності, загальна кількість яких оцінюється як $SCa(R)$.

Варіант 7. Пошук за рівністю значення некластеризуючого (вторинного) індексу

Якщо предикат включає умову рівності значенням атрибута А, який не є первинним ключем відношення, але є некластеризуючий вторинний індекс, то цей індекс може використовуватися для вибірки необхідних записів. В цьому випадку можна припустити, що всі обрані записи будуть розміщуватися в різних блоках (так як індекс не є кластеризуються). Звідси формула оцінки вартості вибірки має такий вигляд:

$nLevelsA(I) + [SCa(R)]$ Варіант

8. Пошук за нерівністю значення вторинного індексу типу В + -Tree

Якщо предикат включає умову нерівності значенням атрибута А ($A < x$, $A \leq x$, $A > x$, $A \geq x$), який має вторинний індекс типу В + -Tree, то для пошуку за умовою $<$ або $\leq$ можна виконати перегляд лист-вузлів дерева від мінімального значення до значення x , а для пошуку за умовами $>$ або $\geq$ - від значення x до максимального значення. Якщо припустити рівномірний розподіл значень, то в кожному разі буде потрібно обробка половини лист-вузлів індексу. Тому для оцінки вартості вибірки можна використовувати таку формулу: $nLevelsA(I) + [nLfBlocksA(I) / 2 + nTuples(R) / 2]$

Алгоритм пошуку єдиною записи в індексі типу В + -Tree представлений в лістингу 9.3.

Лістинг 9.3. Алгоритм пошуку в дереві В + -Tree єдиного запису, відповідного заданому значенню

```
// Пошук в дереві В + -Tree.
// Структура дерева В + -Tree представлена як пов'язаний список, в якому кожен
// вузол, відмінний від лист-вузла, має наступну структуру: максимум n
// елементів, кожен. з яких складається з наступних компонентів: ключового
// значення (key) і покажчика (p) на дочірній вузол (можливо, NULL).
// Ключі впорядковані: key1 < key2 < key3 < ... < keyn-1.
// Лист-вузли вказують власне на записи даних.
// Rredicate - це ключ пошуку.
// Повертається логічна змінна (found), яка вказує, чи була знайдена
// необхідна запис, а також адреса знайденої запису (return_address), якщо така
// маєтся
//
node = get_root_node ();
while (<вузол не є лист-вузлом>) {
i = 1; // Пошук ключа, який менше значення предиката
while (not (i > n or predicate < node [i].key)) {i = i + 1;
}
Node = get_next_node (node [i].p);
```

```

// node [i] .p вказує на піддерево,
// яке може містити необхідний предикат}
// Лист-вузол знайдений. Перевірити, чи існує запис з необхідним
// предикатом
i = 1;
found = FALSE;
while (not (found or i > n)) {
if (predicate = node [i] .key)
then {
found = TRUE;
return_address = node [i] .p;
}
Else
i = i + 1;
}

```

Складені предикати

До цього моменту мова йшла про прості предикати, що включають умови порівняння для єдиного атрибута. Однак у багатьох випадках використовуються складені предикати, які включають кілька умов, що містять більше одного атрибута. Слід нагадати, що складений предикат може бути представлений в двох формах - кон'юнктивній і диз'юнктивній нормальній формі.

- Кон'юнктивна вибірка включає тільки ті рядки, які відповідають усім заданим кон'юнктам.

- Диз'юнктивна вибірка включає набір рядків, сформований шляхом об'єднання всіх рядків, що задовольняють будь-якого з заданих диз'юнктивів.

Кон'юнктивна вибірка без диз'юнкцій

Якщо складений предикат не містить диз'юнктивних виразів, для реалізації операції вибірки може використовуватися один з наступних підходів:

1. Якщо один з атрибутів в кон'юнкції має індекс або є впорядкованим, то для вибірки рядків, що задовольняють заданій умові, можна використовувати одну з стратегій вибірки 2 - 8, запропоновану вище. Потім для кожного обраного рядка виконується перевірка на відповідність іншим, що входять в предикат умовам.

2. Якщо вибірка включає умову рівності для двох або декількох атрибутів і для даної комбінації атрибутів існує складений індекс (або ключ гешування), то можна виконати прямий пошук в цьому індексі (відповідно до запропонованих вище алгоритмів). Тип індексу визначає, який з обговорюваних вище алгоритмів буде використаний.

3. Якщо існують вторинні індекси, створені для одного або декількох атрибутів, і ці атрибути в предикаті використовуються тільки в умовах пошуку по рівності, то в тому випадку, якщо в індексах використовуються вказівники на записи, а не на блоки даних, можна виконати перегляд кожного з індексів з метою пошуку рядків, які відповідають окремим умовам. Потім будується перетин для всіх отриманих наборів вказівників, який утворює результуючий набір вказівників, які відповідають всім цим умовам. Якщо індекси існують не для всіх використовуваних в предикаті атрибутів, то вибрані рядки додатково перевіряються на відповідність залишившимся умовам.

Вибірки з диз'юнкціями

Якщо один з виразів в умові вибірки містить оператор $\vee$ (OR) і цей вираз вимагає виконання лінійного пошуку, оскільки для нього не існує відповідного індексу або не заданий порядок сортування, то вся операція вибірки повинна виконуватися за допомогою операції лінійного пошуку. Оптимізація операції вибірки можлива тільки в тому випадку, якщо індекс або порядок сортування існує для всіх виразів в умові вибірки і здійснюється за рахунок видалення записів, що задовольняють кожній умові з подальшим об'єднанням отриманих результуючих наборів по методу, запропонованому далі, який дозволяє також виключити повторювані значення. Крім того, в цьому випадку можуть використовуватися вказівники на записи, якщо такі існують. Якщо для організації ефективного доступу не може бути використаний жоден з атрибутів, то вибірка здійснюється за допомогою лінійного пошуку і перевірки кожного рядка на відповідність відразу всім заданим умовам.

Операція з'єднання ($T = (R \bowtie_F S)$)

Слід зазначити, що однією з основних проблем, пов'язаних з використанням перших комерційних СУБД, була властива їм невисока продуктивність обробки запитів. Зокрема, найбільші труднощі викликала обробка операції з'єднання, яка є найдорожчою з усіх операцій реляційної алгебри (не рахуючи операції декартового множення). Тому слід докладати всіх зусиль для забезпечення виконання цієї операції з максимально можливою продуктивністю.

В результаті виконання бінарної операції тета-з'єднання створюється результуюче відношення, яке містить рядки, що задовольняють заданому предикату F , застосовуваному до декартового множення двох вихідних відношень, наприклад, таких як R і S . Предикат F задається в форматі $R.a \Theta S.b$, де Θ символ позначає одну з операцій логічного порівняння. Якщо предикат містить тільки операції рівності ($=$), дане з'єднання називається **з'єднанням по еквівалентності**. Якщо результат з'єднання включає всі загальні атрибути відношень R і S , з'єднання називається **природним**.

Типи стратегій реалізації операцій з'єднання:

- З'єднання блоками з використанням вкладених циклів.
- Індексовані з'єднання блоками з використанням вкладених циклів.
- З'єднання за допомогою сортування-злиття.
- хешоване з'єднання.

Оцінки вартості різних стратегій виконання операцій з'єднання наведені в таблиці 9.4. Обговорення почнемо з визначення оцінок кардинальності операцій з'єднання.

Таблиця 9.4. Зведені дані по оцінці вартості операцій введення-виведення для різних стратегій виконання операцій з'єднання

Стратегія	Вартість	Умова
З'єднання блоками з використанням вкладених циклів	$nBlocks(R) + (nBlocks(R) \times nBlocks(S))$	Буфер містить тільки один блок для R та S
	$nBlocks(R) + [nBlocks(S) \times (nBlocks(R) / (nBuffer - 2))]$	Буфер містить $(nBuffer - 2)$ блоків для R
	$nBlocks(R) + nBlocks(S)$	У разі, якщо все блоки R можуть бути переміщені в буфер
Індексовані з'єднання блоками з використанням вкладених циклів	Залежить від методу індексування, наприклад: $nBlocks(R) + nTuples(R) \times (nLevels_A(I) + 1)$	У разі, якщо з'єднувальний атрибут A є відносно S первинним ключем
	$nBlocks(R) + nTuples(R) \times (nLevels_A(I) + SC_A(R) / bFactor(R))$	Якщо індекс I - кластеризуює індекс атрибута A
З'єднання за допомогою сортування-злиття	$nBlocks(R) \times [\log_2(nBlocks(R))] + nBlocks(S) \times [\log_2(nBlocks(S))]$	В разі сортування
	$nBlocks(R) + nBlocks(S)$	У разі злиття
Хешувальне з'єднання	$3(nBlocks(R) + nBlocks(S))$	В разі, якщо індекс хешування зберігається в пам'яті
	$2(nBlocks(R) + nBlocks(S)) \times [\log_n Buffer - 1(nBlocks(S)) - 1] + nBlocks(R) + nBlocks(S)$	В інших випадках

Оцінка кардинальності операції з'єднання

Кардинальність декартового множення відношень R і S, $R \times S$, визначається дуже просто: $nTuples(R) \times nTuples(S)$. На жаль, визначення кардинальності будь-яких операцій з'єднання являє собою значно більш складне завдання, оскільки воно залежить від розподілу значень з'єднуючих атрибутів. У найгіршому випадку відомо, що кардинальність будь-якого з'єднання не може перевищувати кардинальність декартового множення з'єднуючих відношень, тому

$$nTuples(T) \leq nTuples(R) \times nTuples(S)$$

У деяких системах використовується саме ця верхня межа, однак подібну оцінку слід вважати надмірно песимістичною. Якщо знову припустити, що значення атрибутів в обох відношеннях розподілені рівномірно, то можна таким чином поліпшити оцінку кардинальності з'єднання по еквівалентності з використанням предиката $(R.A = S.B)$:

1. Якщо атрибут A є ключовим атрибутом відношення R , то кожен рядок відношення S може бути з'єднаним тільки з одним рядком відношення R . Отже, кардинальність з'єднання по еквівалентності не може перевищити кардинальності відношення S :

$$nTuples(T) \leq nTuples(S)$$

2. Аналогічно, якщо атрибут B є ключовим атрибутом відношення S , то $nTuples(T) \leq nTuples(R)$

3. Якщо ні атрибут A , ні атрибут B не є ключовими, то кардинальність з'єднання можна оцінити таким чином:

$$nTuples(T) = SC_A(R) \times nTuples(S) \text{ или}$$

$$nTuples(T) = SC_B(S) \times nTuples(R)$$

Для отримання першої оцінки використовується той факт, що для будь-якого рядка s щодо S очікується в середньому отримання $SC_A(R)$ рядків із заданим значенням атрибута A і саме ця кількість рядків відношення R буде брати участь в з'єднанні. Помноживши це значення на кількість рядків у відношенні S , отримуємо першу з наведених вище оцінок.

Варіант 1. З'єднання блоками з використанням вкладених циклів

Найпростішим алгоритмом виконання операції з'єднання є метод вкладених циклів, при якому з'єднання двох відношень виконується кожен раз по одному рядку. У зовнішньому циклі виконується послідовний перебір рядків першого відношення R , а у внутрішньому циклі послідовно проглядаються всі рядки другого відношення S . Але, оскільки відомо, що основною одиницею обміну при читанні даних на дискових пристроях є блок, можна поліпшити базовий алгоритм, створивши ще два додаткових цикли, призначених для обробки лічених з диска блоків, що і зроблено в алгоритмі, наведеному в переліку 9.4.

Перелік 9.4. Алгоритм з'єднання блоками з використанням вкладених циклів

// З'єднання блоками з використанням вкладених циклів.

// Блоки в обох файлах послідовно пронумеровані, починаючи з 1.

// Повертається результуюча таблиця з'єднання відношень R і S

// for iblock = 1 to nBlocks (R) { // Зовнішній цикл

Rblock = read_block (R, iblock);

for jblock = 1 to nBlocks (S) { // Внутрішній цикл

Sblock = read_block (S, jblock);

for i = 1 to nTuples (Rblock) { for j = 1 to nTuples (Sblock) {

if (Rblock.tuple [i] /Sblock.tuple [j] <відповідає умові з'єднання>)

then <помістити його в результуючу таблицю>;

}

}

}

}

Оскільки кожен блок відношення R повинен бути лічений хоча б один раз, а кожен блок відношення S лічений для кожного блоку відношенні R , оцінка вартості цього методу може бути виконана за формулою

$$nBlocks(R) + (nBlocks(R) \times nBlocks(S))$$

Зауважимо, що значення другого виразу цієї формули постійно, тоді як значення першого виразу залежить від того, яке з відношень вибрано для зчитування в зовнішньому циклі. Очевидно, що в зовнішньому циклі повинно зчитуватися те з'єднувальне відношення, яке займає меншу кількість дискових блоків.

Ще одним поліпшенням даної стратегії може стати зчитування в буфер бази даних максимально можливої кількості блоків меншого з відношень (скажімо, R) зі збереженням місця тільки для одного блоку внутрішнього відношення і одного блоку результуючого відношення. Якщо в буфер бази даних може бути вміщено nBuffer блоків, то слід відразу ж вважати nBuffer-2 блоки відношення R і один блок відношення S. Загальна кількість зчитувальних блоків відношення R як і раніше залишається рівним nBlocks (R), проте загальна кількість зчитувальних блоків відношення S зменшується приблизно до $[nBlocks (S) \times (nBlocks (R) / (nBuffer - 2))]$. При використанні даного підходу формула обчислення оцінки вартості операції з'єднання приймає наступний вигляд:

$$nBlocks (R) + [nBlocks (S) \times (nBlocks (R) / (nBuffer - 2))]$$

Якщо виявиться можливим вважати всі блоки відношення R, то формула буде мати такий вигляд:

$$nBlocks (R) + nBlocks (S)$$

Якщо з'єднуються атрибути в з'єднанні по еквівалентності (або природному з'єднанні) вони утворюють ключ внутрішнього відношення, то внутрішній цикл може завершуватися, як тільки буде знайдено перша відповідність.

Варіант 2. Індексовані з'єднання з використанням вкладених циклів

Якщо для з'єднуються атрибутів внутрішнього відношення існує індекс (або хеш-функція), то можна замінити неефективну операцію перегляду файлу більш ефективним пошуком за індексом. Для кожного рядка відношення R вибірка відповідних рядків відношення S здійснюється за допомогою індексу.

Це значно ефективніший алгоритм з'єднання, що дозволяє уникнути створення в якості проміжного результату декартове множення відношень R і S. Вартість перегляду відношення R становить nBlocks (R), як і в попередньому варіанті. Однак вартість вибірки відповідних рядків відношення S тепер залежить від типу наявного індексу і кількості рядків, відповідних умові. Наприклад, якщо з'єднувальний атрибут A є відносно S первинним ключем, то вартість операції з'єднання може бути обчислена за формулою

$nBlocks (R) + nTuples (R) \times (nLevels_A (I) + 1)$ Якщо сполучається атрибут A щодо S є кластеризуються індексом, то вартість з'єднання можна обчислити за такою формулою:

$$nBlocks(R) + nTuples(R) \times (nLevels_A(I) + [SC_A(R)/bFactor(R)])$$

Варіант 3. З'єднання за допомогою сортування-злиття

Для з'єднань по еквівалентності найбільш ефективний алгоритм реалізується в тому випадку, якщо обидва відношення відсортовані по з'єднувальним атрибутам. В цьому випадку можна знайти відповідні рядки відношень R і S, виконавши злиття двох відношень. Якщо вихідні відношення не відсортовані, можна попередньо виконати їх сортування. Оскільки рядки відношень відсортовані, ті з них, які містять однакові значення з'єднувального атрибута, гарантовано будуть розміщуватися поруч. Якщо припустити, що з'єднання

виконується для зв'язку типу "багато до багатьох", тобто в обох відношеннях може існувати кілька рядків з одним і тим же з'єднувальним значенням, а також прийняти, що будь-який набір рядків з одним і тим же з'єднувальним значенням може бути повністю поміщений в буфер бази даних, то кожен блок кожного з з'єднувальних відношень досить буде оцінити тільки один раз. Тому вартість з'єднання методом сортування-злиття може бути обчислена за формулою

$$n\text{Blocks}(R) + n\text{Blocks}(S)$$

Якщо одне з відношень (наприклад, R) має бути попередньо відсортовано, то в формулу слід додати оцінку вартості операції сортування, яка приблизно може бути виконана за такою формулою: $n\text{Blocks}(R) \times [\log_2(n\text{Blocks}(R))]$

Варіант 4. З'єднання за допомогою хешування

Алгоритм з'єднання за допомогою хешування може використовуватися в разі природного з'єднання (або з'єднання по еквівалентності) відношень R і S по атрибуту A . Суть цього алгоритму: На першому етапі - етап хешування, відбувається розбиття відношення R і S на розділи відповідно до значень деякої хеш-функції, результати обчислення якої дають рівномірно розподілені випадкові значення. Кожен отриманий еквівалентний розділ відношень R і S повинен мати одні й ті ж значення, що з'єднувальних атрибутів, хоча він може містити більше одного їх значення. Алгоритм передбачає перевірку еквівалентних розділів на наявність одного і того ж значення.

Наприклад, якщо відношення R за допомогою деякої хеш-функції $h()$ розбите на розділи $R_1, R_2, \dots, R_m$, а відношення S - на розділи $S_1, S_2, \dots, S_m$, то якщо B і C є атрибутами відношень R і S відповідно, а $h(R.B) \neq h(S.C)$, то $R.B \neq S.C$. Але якщо $h(R.B) = h(S.C)$, це не обов'язково означає, що $R.B = S.C$, оскільки одне і те ж значення хеш-функції може бути отримано для різних вихідних значень.

На другому етапі - етап перевірки, послідовно зчитується кожен з розділів R і робиться спроба поєднати вміст в ньому рядки з рядками еквівалентного розділу S . Якщо другий етап виконується за допомогою з'єднання вкладеними циклами, то в зовнішньому циклі застосовується розділ меншого розміру, припустимо R_i . Весь розділ R_i зчитується в пам'ять, після чого з диска зчитується кожен блок еквівалентного йому розділу S_i і в ньому кожен рядок використовується для перевірки наявності відповідних рядків в розділі R_i . Для підвищення ефективності зазвичай в оперативній пам'яті формується хеш-таблиця для кожного розділу R_i із застосуванням другої хеш-функції, відмінної від тієї, за допомогою якої виконується розбиття відношень на розділи.

Вартість операції з'єднання по методу хешування може бути обчислена за такою формулою:

$$3(n\text{Blocks}(R) + n\text{Blocks}(S))$$

У цій формулі враховується необхідність зчитування відношень R і S при їх розбитті на розділи, записи кожного з цих розділів на диск і повторного зчитування кожного з розділів відношень R і S для пошуку відповідних рядків. Ця оцінка є приблизною, і в ній не враховується можливість переповнення розділу. Крім того, передбачається, що є можливість зберігати весь хеш-індекс у оперативній пам'яті. Якщо ці умови не дотримуються, то розбиття відношень на розділи неможливо виконати за один прохід і тому повинен застосовуватися рекурсивний алгоритм розбиття на розділи.

В цьому випадку оцінка вартості може бути виражена за допомогою формули $2(n\text{Blocks (R)} + n\text{Blocks (S)}) \times [\log_{\text{buffer-1}}(n\text{Blocks (S)}) - 1] + n\text{Blocks (R)} + n\text{Blocks (S)}$

Операція проєкції ($S = \Pi_{A_1, A_2, \dots, A_m}(R)$)

Операція проєкції також є унарною операцією, в результаті виконання якої створюється відношення S , що містить вертикальна підмножина відношення R , яке включає значення зазначених атрибутів з виключенням повторюваних рядків. Таким чином, для реалізації операції проєкції необхідно виконати наступну послідовність дій.

1. Видалення із відношення непотрібних атрибутів.

2. Виключення з результуючої таблиці будь-яких повторюваних рядків, що з'явилися в результаті виконання попереднього етапу.

Другий етап виконати складніше, ніж перший, але він потрібен тільки в тому випадку, якщо атрибути проєкції не містять ключ відношення. Для виключення дублікатів використовуються два основні підходи: сортування та хешування.

Перш ніж приступити до їх розгляду, слід оцінити кардинальність результуючого відношення операції проєкції.

Оцінка кардинальності операції проєкції

Якщо створювана проєкція містить ключовий атрибут, то етап видалення повторюваних рядків не буде потрібен, а кардинальність результуючої таблиці операції проєкції буде дорівнювати:

$$n\text{Tuples (S)} = n\text{Tuples (R)}$$

Якщо проєкція складається з єдиного неключових атрибута ($S = \Pi_A(R)$), то кардинальність результуючої таблиці може бути оцінена за формулою

$$n\text{Tuples(S)} = SC_A(R)$$

В іншому випадку, якщо припустити, що відношення є декартовим множенням значень всіх його атрибутів (що, взагалі кажучи, мало ймовірно), кардинальність результуючої таблиці операції проєкції можна буде обчислити за такою формулою:

$$n\text{Tuples(S)} \leq \min(n\text{Tuples(R)}, \prod_{i=1}^i n\text{Distinct}_{A_i}(R))$$

Варіант 1. Виняток дублікатів за допомогою сортування

Суть даного підходу полягає в сортуванні рядків скороченого варіанту відношення з використанням всіх, залишившихся атрибутів як ключ сортування. Це призведе до переупорядкування рядків таким чином, що рядки що повторюються виявляться поруч і можуть бути легко видалені. Для видалення непотрібних атрибутів потрібно сканувати всі рядки відношення R і скопіювати всі необхідні атрибути у тимчасове відношення. Вартість цієї операції складе $n\text{Blocks (R)}$. Оцінка вартості операції сортування може бути обчислена за формулою $n\text{Blocks(R)} \times [\log_2(n\text{Blocks(R)})]$, тому загальна вартість першого етапу виконання операції проєкції складе:

$$n\text{Blocks(R)} + n\text{Blocks(R)} \times [\log_2(n\text{Blocks(R)})]$$

Варіант 2. Видалення дублікатів за допомогою хешування

Підхід з використанням хешування може виявитися корисним в тих випадках, якщо доступне кількість буферів дискових блоків в порівнянні із загальною кількістю блоків відношення R досить велике. хешування виконується в два етапи: спочатку відбувається розбиття на розділи, а потім видалення дублікатів.

На першому етапі один буферний блок виділяється для читання вихідного відношення R , а що залишилися $(nBuffer-1)$ блоки - для розміщення вихідних даних. З кожного рядка відношення R будуть видалені всі непотрібні атрибути, після чого до комбінації інших атрибутів застосовується хеш-функція $h()$ і скорочений рядок записується в буфер відповідно до обчисленим значенням. Хеш-функція $h()$ вибирається таким чином, щоб по її значенню всі рядки рівномірно розподілялися по $(nBuffer-1)$ розділах.

Два рядки, що належать різним розділам, гарантовано не будуть дублікатами, оскільки обчислені для них значення хеш-функції виявилися різними. В результаті область пошуку дублікатів звужується до розміру окремих розділів.

На другому етапі виконуються наступні дії:

- почергове зчитування вмісту кожного зі створених $(nBuffer-1)$ розділів;
- застосування до кожного зчитаного рядку другої (відмінної від першої) хеш-функції $h_2()$;
- вставка рядка і обчисленого для неї значення в хеш-таблицю, розміщену в оперативній пам'яті;
- якщо значення хеш-функції для деякого рядка виявилось таким же, як і для однієї з попередніх рядків, виконується перевірка, чи є ці рядки однаковими, і видалення всіх повторюваних рядків, крім першої;
- після обробки розділу всі його рядки з хеш-таблиці в оперативній пам'яті виводяться в результуючий файл.

Якщо кількість блоків, необхідних для розміщення тимчасової таблиці, яка містить результати виконання операції проєкції для відношення R до видалення дублікатів, прийняти рівним nb , то вартість операції видалення дублікатів можна оцінити за формулою

$$nBlocks(R) + nb$$

У цій формулі не враховуються операції запису вихідного набору даних, а також передбачається, що при хешуванні не відбувається переповнення розділів.

Операції реляційної алгебри над множинами $T=R \cup S$, $T=R \cap S$, $T=R-S$

Бінарні операції над множинами, до яких відносяться об'єднання $(R \cup S)$, перетин $(R \cap S)$ і різницю множин $(R-S)$, застосовні тільки до тих відношень, які є сумісними з об'єднанням (тобто мають ідентичні структури). Всі ці операції можуть бути реалізовані за допомогою попереднього сортування вихідних відношень по одним і тим же атрибутам з наступним однократним переглядом відсортованих даних з метою отримання бажаного результату. У разі об'єднання в результуючу таблицю поміщаються всі рядки, присутні в кожному з вихідних відношень, але з видаленням дублікатів, якщо це буде потрібно. У разі перетину в результуючу таблицю поміщаються тільки ті рядки, які присутні в обох відношеннях. У разі

різниці множин аналізується кожен рядок відношення R, причому вона виводиться в результуючий набір тільки в тому випадку, якщо для цього рядка немає відповідного рядка щодо S. Для будь-якої з цих операцій може бути розроблений алгоритм, заснований на структурі алгоритму сортування-злиття.

Формула обчислення вартості кожної з цих операцій досить проста:

$$n\text{Blocks}(R) + n\text{Blocks}(S) + n\text{Blocks}(R) \times [\log_2(n\text{Blocks}(R))] + n\text{Blocks}(S) \times [\log_2(n\text{Blocks}(S))]$$

Крім того, для реалізації цих операцій над множинами можна застосовувати і алгоритм хешування. Наприклад, об'єднання може виконуватися за допомогою побудови в пам'яті індексу хешування для відношення R з наступним додаванням в цей індекс тільки тих рядків відношення S, які в ньому відсутні. По завершенні цього процесу всі рядки з індексу хешування виводяться в результуючий набір даних.

Оцінка кардинальності операцій над множинами

Оскільки при виконанні операції об'єднання виключаються повторювані рядки, в загальному випадку досить складно визначити кардинальність її результуючого набору, однак можна вказати верхню і нижню межі наступним чином:

$$\max(n\text{Tuples}(R), n\text{Tuples}(S)) \leq n\text{Tuples}(T) \leq n\text{Tuples}(R) + n\text{Tuples}(S)$$

У разі операції різниці множин верхня і нижня межі можуть бути визначені за допомогою формули

$$0 \leq n\text{Tuples}(T) \leq n\text{Tuples}(R)$$

Приклад 9.4. Розглянемо наступний запит SQL, за допомогою якого визначається середній прохідний бал студента інституту:

```
SELECT AVG (ПБал)
FROM СТУДЕНТ;
```

У цьому запиті використовується агрегуюча функція AVG (обчислення середнього значення). Для реалізації подібного запиту потрібно переглянути всі відношення СТУДЕНТ з підрахунком кількості рядків що в ньому знаходяться і обчисленням суми значень атрибута ПБал. Після завершення перегляду на підставі двох обчислених сум визначається потрібну установку. Тепер розглянемо інший запит SQL, за допомогою якого визначається середнє значення прохідного балу студента кожної з груп:

```
SELECT AVG (ПБал)
FROM СТУДЕНТ
GROUP BY НомерЗаліковки;
```

В цьому запиті також застосовується агрегована функція AVG, але на цей раз в поєднанні з конструкцією групування. Для обробки групуються запитів може використовуватися алгоритм сортування або хешування - аналогічно тому, як і при видаленні дублікатів. У разі групуючих запитів формула оцінки кардинальності результуючої таблиці може бути виведена з формул, запропонованих для операції вибірки.

Конвеєрна обробка даних

До цього моменту передбачалося, що результати всіх проміжних операцій реляційної алгебри тимчасово записуються на диск. Цей процес носить назву

матеріалізації - результати однієї операції до початку наступної зберігаються у тимчасових відношеннях для подальшої обробки.

Альтернативний підхід полягає в конвеєрній передачі результатів однієї операції на обробку іншої операції без створення тимчасових відношень, призначених для зберігання проміжних результатів. Очевидно, що при використанні конвеєрного механізму виключаються витрати на створення тимчасових відношень і повторне їх зчитування.

$\sigma \text{Статус} = \text{'Відмінник'} \wedge \text{ПБал} > 4.8 \text{ (СТУДЕНТ)}$

Якщо припустити, що для атрибута ПБал існує індекс, то можна використовувати правило каскадної вибірки для перетворення цієї операції в дві операції з простими предикатами:

$\sigma \text{Статус} = \text{'Відмінник'} \text{ (} \sigma \text{ ПБал} > 4.8 \text{ (СТУДЕНТ))}$

Тепер наявний індекс можна використовувати для ефективної обробки першої операції вибірки за значенням атрибута ПБал; отриманий результат потрібно помістити у тимчасове відношення, після чого застосувати другу операцію вибірки, але вже до тимчасового відношення з проміжними результатами. Механізм конвеєрної обробки виключає створення тимчасового відношення і передбачає застосування другої операції вибірки до кожного окремого рядку, одержуваної в результаті виконання першої операції в міру її створення. Рядки, що задовольняють предикату другої операції вибірки, будуть записуватися безпосередньо в результуючий набір даних.

У загальному випадку механізм конвеєрної обробки реалізується за допомогою групи окремих процесів або потоків, які виконуються в рамках обчислювального середовища СУБД. Кожен процес механізму конвеєрної обробки отримує на вході потік рядків і створює інший потік рядків на виході. Для кожної пари суміжних операцій створюється окремий буфер в пам'яті, призначений для розміщення рядків, що передаються з першої операції в другу. Один з основних недоліків методу конвеєрної обробки полягає в тому, що вхідні дані не можуть бути доступні операціям одночасно у всьому об'ємі. Це обмежує можливості вибору алгоритмів обробки. Наприклад, якщо при реалізації операції з'єднання, що передаються конвеєрним способом рядки будуть не відсортовані по з'єднувальним атрибутам, то це не дозволить застосувати стандартний алгоритм з'єднання методом сортування-злиття. Проте, існує безліч інших можливостей застосування механізму конвеєрної обробки при реалізації різних стратегій.

Питання:

9.1. В чому полягає призначення процедури обробки запитів?

9.2. Чим відрізняється обробка запитів в реляційних системах від обробки запитів на низькорівневих мовах запитів, підтримуваних мережевими і ієрархічними системами?

9.3. Які основні етапи процесу обробки запиту?

9.4. Назвіть основні стадії етапу декомпозиції запиту.

9.5. У чому полягає відмінність між кон'юнктивною і диз'юнктивною нормальними формами?

9.6. Назвіть правила перетворення запитів, які можуть застосовуватися в наступних випадках:

а) для операцій вибірки;

- б) для операцій проекції;
- в) для операцій тета-з'єднання.

9.7. Назвіть евристичні правила, які повинні застосовуватися для підвищення продуктивності обробки запитів.

9.8. Які типи статистичних відомостей повинна зберігати СУБД, щоб мати можливість оцінювати вартість операцій реляційної алгебри?

9.9. За яких обставин система буде змушена для виконання операції вибірки вдатися до стратегії лінійного пошуку?

9.10. Які основні типи стратегій реалізації операцій з'єднання вам відомі?

9.11. Поясніть суть конвеєрної обробки даних?

ТЕМА 10

Захист баз даних

Дані є цінним ресурсом, доступ до якого необхідно строго контролювати і регламентувати, так само як і до будь-яких інших корпоративних ресурсів. Частина або навіть всі корпоративні дані можуть мати стратегічне значення для організації і тому повинні зберігатися в секреті, під надійною охороною.

Захист бази даних - забезпечення захищеності бази даних проти будь-яких навмисних або ненавмисних загроз за допомогою різних заходів.

Поняття захисту можна застосувати не тільки до даних, що зберігаються в базі даних. Проломи в системі захисту можуть виникати і в інших частинах системи, що, в свою чергу, наражає на небезпеку і саму базу даних. Отже, захист бази даних повинен охоплювати використання обладнання, програмне забезпечення, персонал і, власне, дані. Для ефективної реалізації захисту необхідні відповідні засоби контролю, які визначаються конкретними вимогами, що впливають з особливостей експлуатованої системи.

Потенційні небезпеки:

- викрадення і фальсифікація даних;
- втрата конфіденційності (порушення таємниці)
- порушення недоторканності особистих даних;
- втрата цілісності;
- втрата доступності.

Зазначені ситуації відмічають основні напрямки, в яких керівництво повинно вживати заходи, що знижують ступінь ризику, тобто потенційну можливість втрати або пошкодження даних. У деяких ситуаціях всі зазначені аспекти пошкодження даних тісно пов'язані між собою, так що дії, спрямовані на порушення захищеності системи в одному напрямку, часто призводять до зниження її захищеності у всіх інших.

Викрадення і фальсифікація даних може відбуватися єе тільки в середовищі бази даних - вся організація, так чи інакше, схильна до цього ризику. Однак дії з викрадення або фальсифікації інформації завжди відбуваються людьми, тому основна увага має бути зосереджена на скороченні загальної кількості зручних ситуацій для виконання подібних дій.

Поняття *конфіденційності* означає необхідність збереження даних в таємниці. Як правило, конфіденційними вважаються тільки ті дані, які є важливими для всієї організації, тоді як поняття *недоторканності* даних стосується вимогам захисту інформації про окремих співробітників. Наслідком порушення в системі захисту, викликаного втрату конфіденційності даних, може бути втрата надійних позицій в конкурентній боротьбі.

Втрата цілісності даних призвело до перекручення чи руйнування даних, що може мати найсерйозніші наслідки для подальшої роботи організації. В даний час безліч організацій функціонує в безперервному режимі, надаючи свої послуги клієнтам 24 години на добу і 7 днів на тиждень.

Втрата доступності даних означатиме, що або дані, або система, або і те й інше одночасно виявляться недоступними користувачам, а це може поставити під загрозу фінансове становище організації.

Мета захисту бази даних - мінімізувати втрати, викликані заздалегідь передбаченими подіями. Прийняті рішення повинні забезпечувати ефективне використання понесених витрат і виключати зайве обмеження можливостей, які надаються користувачам.

10.1 Основні типи загроз

Загроза - будь-яка ситуація (або подія), викликана навмисно або ненавмисно, яка здатна несприятливо вплинути на систему. Загроза може бути викликана ситуацією (або подією), яка здатна завдати шкоди організації, причиною якої може служити людина, подія чи збіг обставин. Збиток може бути матеріальним (наприклад, втрата обладнання, програмного забезпечення або даних) або нематеріальним (наприклад, втрата довіри партнерів або клієнтів).

Будь-яка загроза повинна розглядатися як потенційна можливість порушення системи захисту, яка в разі успішної реалізації може надати той чи інший негативний вплив. У таблиці 10.1 наведені приклади різних типів загроз із зазначенням тих областей, в яких вони можуть впливати на систему.

Таблиця 10.1. Приклади можливих загроз

Небезпека	Викрадення і фальсифікація даних	Втрата конфіденційності	Порушення недоторканості особистих даних	Втрата цілісності	Втрата доступності
Використання прав доступу іншої особи	+	+	+		
Несанкціонована зміна або копіювання даних	+			+	
Зміна програм	+			+	+
Непродумані методики і процедури, що допускають змішування конфіденційних і звичайних даних в одному документі	+	+	+		
Підключення до кабельних мереж	+	+	+		
Введення хакерами некоректних даних	+	+	+		

Шантаж	+	+	+		
Створення "лазівок" в системі	+	+	+		
Викрадення даних, програм і обладнання	+	+	+		+
Відмова систем захисту, що викликала перевищення допустимого рівня доступу		+	+	+	
Брак персоналу і страйки				+	+
Недостатня навченість персоналу		+	+	+	+
Перегляд і розкриття засекречених даних	+	+	+		
Електронні перешкоди і радіація				+	+
Руйнування даних в результаті відключення або стресу в мережі				+	+
Пожежі (через короткі замикання, удари блискавок, підпалів), повені, диверсії				+	+
Фізичне пошкодження обладнання				+	+
Обрив або від'єднання кабелів				+	+
Впровадження комп'ютерних вірусів				+	+

Будь-яка організація повинна встановити типи можливих загроз, яким може піддатися її комп'ютерна система, після чого розробити відповідні плани і необхідні контрзаходи, з оцінкою рівня витрат, необхідних для їх реалізації.

Навіть малоймовірні обставини повинні бути прийняті до уваги, особливо якщо їх вплив може виявитися досить істотним.

Контрзаходи – комп'ютерні засоби контролю

Відносно загроз, які можуть негативно вплинути на роботу комп'ютерних систем, повинні бути прийняті контрзаходи самих різних типів, починаючи від фізичного контролю і закінчуючи адміністративно-організаційними процедурами.

Типи засобів контролю:

- авторизація користувачів,
- застосування уявлень,
- резервне копіювання і відновлення,
- підтримка цілісності,
- шифрування,
- застосування RAID-масивів.

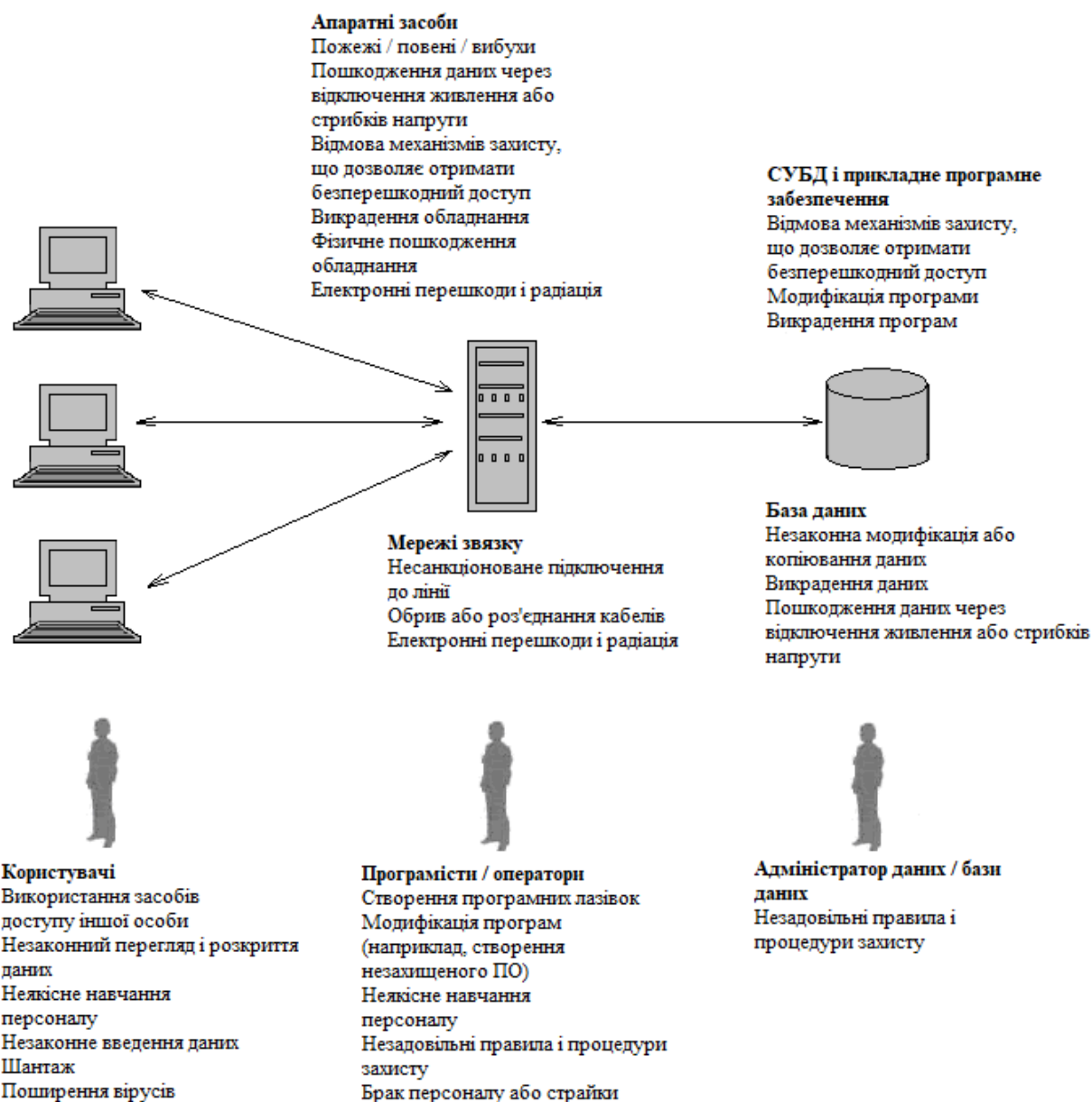


Рис. 10.1. Узагальнена схема потенційних загроз, до яких схильні комп'ютерні системи

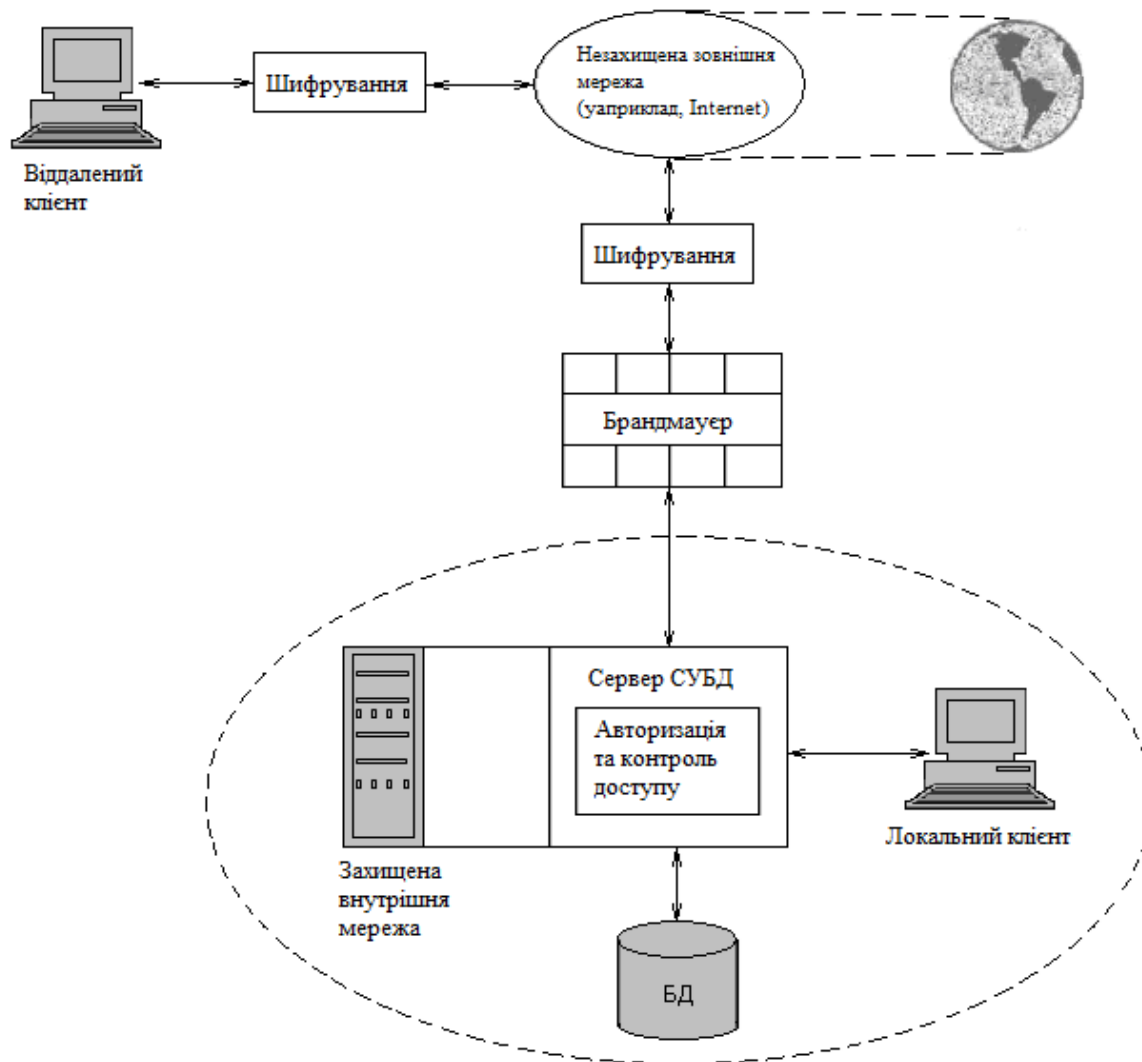


Рис. 10.2. Загальна схема типової багатокористувацької комп'ютерної системи

Авторизація користувачів

Авторизація - надання прав (або привілеїв), що дозволяють їх власнику мати законний доступ до системи або до її об'єктів. Засоби авторизації користувачів можуть бути вбудовані безпосередньо в програмне забезпечення і можуть управляти не тільки наданими користувачам правами доступу до системи або об'єктами, а й набором операцій, які користувачі можуть виконувати з кожним доступним йому об'єктом. З цієї причини механізм авторизації часто називають засобами управління доступом. Термін "власник" в наведеному вище визначенні може позначати користувача - людину або програму. Термін "об'єкт" може позначати таблицю даних, уявлення, додаток, процедуру, тригер або будь-який інший об'єкт, який може бути створений в рамках системи.

Аутентифікація - механізм визначення того, чи є користувач тим, за кого себе видає. За надання користувачу доступу до комп'ютерної системи зазвичай відповідає системний адміністратор, і в обов'язки якого входить створення облікових записів

користувачів. Кожному користувачеві присвоюється унікальний ідентифікатор, який використовується операційною системою для визначення того, хто є хто. З кожним ідентифікатором зв'язується певний пароль, який обирається користувачем і відомий операційній системі. При реєстрації користувач повинен надавати системі свій пароль для виконання перевірки (аутентифікації) того, чи є він тим, за кого себе видає. У деяких СУБД ведеться список ідентифікаторів користувачів і пов'язаних з ними паролів, що відрізняється від аналогічного списку, підтримуваного операційною системою. В інших типах СУБД ведеться список, записи якого зв'язуються з записами списку користувачів операційної системи з урахуванням поточного реєстраційного ідентифікатора користувача. Це запобігає спробам користувачів зареєструватися в середовищі СУБД під ідентифікатором, відмінним від того, який вони використовували при реєстрації в системі.

9.2 Привілеї

Як тільки користувач отримає право доступу до СУБД, йому можуть автоматично надаватися різні привілеї, пов'язані з його ідентифікатором. Зокрема, ці привілеї можуть включати дозвіл на доступ до певних баз даних, таблиць, уявлень і індексам, а також на створення цих об'єктів або ж право викликати на виконання різні утиліти СУБД. Привілеї надаються користувачам, щоб вони могли виконувати завдання, які належать до кола їх посадових обов'язків. Надання зайвих або непотрібних привілеїв може привести до порушення захисту, тому користувач повинен отримувати тільки такі привілеї, без яких він не має можливості виконувати свою роботу.

Деякі типи СУБД функціонують як *закриті системи*, тому користувачам крім дозволу на доступ до самої СУБД потрібно мати окремі дозволи і на доступ до конкретних її об'єктів. Ці дозволи видаються або АБД, або власниками певних об'єктів системи. На противагу цьому, *відкриті системи* за умовчанням надають користувачам, які пройшли перевірку їх достовірності, повний доступ до всіх об'єктів бази даних.

Права володіння та привілеї

Деякі об'єкти в середовищі СУБД належать самій СУБД. Зазвичай ця приналежність оформлена за допомогою спеціального ідентифікатора суперкористувача, наприклад адміністратора бази даних. Як правило, володіння деяким об'єктом надає його власнику весь можливий набір привілеїв щодо цього об'єкта. Це правило застосовується до всіх авторизованих користувачів, які отримують права володіння певними об'єктами.

Будь-який новостворений об'єкт автоматично передається у володіння його творцеві, який і отримує весь можливий набір привілеїв для даного об'єкта. Хоча при цьому користувач може бути власником деякого представлення, єдиною привілегією, яка буде надана йому в відношенні цього об'єкта, може виявитися право вибірки даних з даного подання. Причина подібних обмежень полягає в тому, що вказаний користувач має обмежений набір прав щодо базових таблиць створеного ним представлення. Належні власнику привілеї можуть бути передані іншим авторизованим користувачам. Наприклад, власник кількох таблиць бази даних може надати іншим

користувачам право вибірки інформації з цих таблиць, але не дозволити їм вносити в таблиці будь-які зміни. У мові SQL передбачено, що якщо користувач передає будь-які привілеї, він може вказати, чи набуває одержувач цих привілеїв право передавати ці привілеї іншим користувачам.

Представлення (підсхеми)

Представлення - динамічний результат однієї або декількох реляційних операцій з базовими відносинами з метою створення деякого іншого ставлення. Представлення є віртуальним відношенням, яке реально в базі даних не існує, але створюється на вимогу окремого користувача в момент надходження цієї вимоги. Механізм уявлень служить потужним і гнучким інструментом організації захисту даних, що дозволяє приховувати від певних користувачів деякі частини бази даних. В результаті користувачі не матимуть ніяких відомостей про існування будь-яких атрибутів або рядків даних, які не доступні через представлення, що знаходяться в їх розпорядженні. Представлення може бути визначено на базі декількох таблиць, після чого користувачеві будуть надані необхідні привілеї доступу до цього представлення, але не до базових таблиць. В даному випадку використання представлення встановлює більш жорсткий механізм контролю доступу, ніж звичайне надання користувачеві тих чи інших прав доступу до базових таблиць.

Резервне копіювання і відновлення

Резервне копіювання - періодично виконувана процедура отримання копії бази даних і її файлу журналу (а також, можливо, програм) на носії, що зберігається окремо від системи. Будь-яка сучасна СУБД повинна надавати методи резервного копіювання, що дозволяють відновлювати базу даних в разі її руйнування. Крім того, рекомендується створювати резервні копії бази даних і її файлу журналу з деякою встановленою періодичністю, а також організовувати зберігання створених копій в місцях, забезпечених необхідним захистом. У разі аварійної відмови, в результаті якої база даних стає непридатною для подальшої експлуатації, резервна копія і зафіксована в файлі журналу оперативна інформація використовуються для відновлення бази даних до останнього узгодженого стану.

Ведення журналу - процедура створення і обслуговування файлу журналу, що містить відомості про всі зміни, внесені в базу даних з моменту створення останньої резервної копії, і призначеного для забезпечення ефективного відновлення системи в разі її відмови. СУБД повинна надавати способи ведення системного журналу, в якому будуть фіксуватися відомості про всі зміни стану бази даних і про хід виконання поточних транзакцій, що необхідно для ефективного відновлення бази даних у разі відмови. Переваги використання подібного журналу полягають в тому, що в разі порушення роботи або відмови СУБД базу даних можна буде відновити до останнього відомого узгодженого стану, скориставшись останньою створеною

резервною копією бази даних і оперативною інформацією, що міститься у файлі журналу.

Підтримка цілісності

Засоби підтримки цілісності даних також вносять певний вклад в загальну захищеність бази даних, оскільки вони повинні запобігти переходу даних в неузгоджений стан, а значить, виключити загрозу отримання помилкових або неправильних результатів розрахунків.

Шифрування

Шифрування - перетворення даних з використанням спеціального алгоритму, в результаті чого дані стають недоступними для читання будь-якою програмою, яка не має ключа дешифрування. Якщо в системі з базою даних міститься дуже важлива конфіденційна інформація, то має сенс зашифрувати її з метою попередження можливої загрози несанкціонованого доступу з зовнішнього боку (по відношенню до СУБД). Деякі СУБД включають засоби шифрування, призначені для використання в подібних цілях. Підпрограми таких СУБД забезпечують санкціонований доступ до даних (після їх дешифрування), хоча це буде пов'язано з деяким зниженням продуктивності, викликаним необхідністю подвійного перетворення. Існує безліч різних технологій шифрування даних з метою приховування інформації, що передається, причому одні з них називають необоротними, а інші - оборотними. Необоротні методи, як і впливає з їх назви, не дозволяють встановити вихідні дані, хоча останні можуть використовуватися для збору достовірної статистичної інформації. Оборотні технології використовуються частіше. Для організації захищеної передачі даних по незахищених мережах повинні використовуватися системи шифрування, що включають такі компоненти:

- ключ шифрування, призначений для шифрування вихідних даних (звичайного тексту);
- алгоритм шифрування, який описує, як за допомогою ключа шифрування перетворити звичайний текст в зашифрований;
- ключ дешифрування, призначений для дешифрування зашифрованого тексту;
- алгоритм дешифрування, який описує, як за допомогою ключа дешифрування перетворити зашифрований текст в звичайний вихідний.

Деякі системи шифрування, звані симетричними, використовують один і той же ключ, як для шифрування, так і для дешифрування, при цьому передбачається наявність захищених ліній зв'язку, призначених для обміну ключами. Однак більшість користувачів не мають доступу до захищених ліній зв'язку, тому для отримання надійного захисту довжина ключа повинна бути не менше довжини самого повідомлення. Як правило, симетричні алгоритми є більш швидкодіючими, ніж асиметричні, однак на практиці обидві схеми часто застосовуються спільно, коли алгоритм з відкритим ключем використовується для шифрування випадковим чином

згенерованого ключа шифрування, а випадковий ключ служить для шифрування самого повідомлення із застосуванням деякого симетричного алгоритму.

RAID (масив незалежний дискових накопичувачів с надлишковістю)

Апаратне забезпечення, на якому експлуатується СУБД, має бути відмовостійким. Це означає, що СУБД повинна продовжувати працювати навіть при відмові апаратних компонентів. Для цього необхідно мати надлишкові компоненти, які можуть бути об'єднані в систему, що зберігає свою працездатність при відмові одного або декількох компонент. До числа основних апаратних компонентів, які повинні бути відмовостійкими, відносяться дискові накопичувачі, дискові контролери, процесори, джерела живлення і вентилятори охолодження. Дискові накопичувачі є найбільш уразливими серед всіх апаратних компонентів і характеризуються найнижчими показниками безперервної роботи. Одним з рішень цієї проблеми є застосування технології RAID.

Спочатку ця аббревіатура розшифровувалась як Redundant Array of Inexpensive Disks (Масив недорогих дискових накопичувачів з надмірністю), але в подальшому букву "I" в цій аббревіатурі стали розглядати як скорочення від "independent" (незалежний).

RAID-масив являє собою масив дискових накопичувачів великого об'єму, що складається з декількох незалежних дисків, спільне функціонування яких організовано таким чином, що при цьому підвищується надійність і разом з тим збільшується продуктивність.

Продуктивність збільшується завдяки смуговому розподілу даних. Дані на дисках розподіляються за сегментами, які представляють собою розділи дисків рівного розміру (цей розмір називається *одиноцею смугового розподілу*), які розподіляються по декількох дисках і забезпечують прозорий доступ. В результаті такий масив стає аналогічним одному великому швидкодіючому диску, але фактично дані в ньому розподілені по декількох дисках меншого обсягу. Смуговий розподіл забезпечує підвищення продуктивності введення-виведення, оскільки дозволяє одночасно виконувати кілька операцій введення-виведення (на різних дисках).

Поряд з цим смуговий розподіл даних дозволяє рівномірно розподіляти навантаження між дисками.

Підвищена надійність RAID-масиву забезпечується завдяки дублюванню даних (такі дублікати називаються *дзеркальними копіями*) і зберігання на дисках надлишкової інформації, сформованої з використанням схем контролю парності або схем виправлення помилок, таких як код Ріда-Соломона (Reed-Solomon). У схемі контролю парності кожен байт повинен мати пов'язаний з ним біт парності, який приймає значення 0 або 1, залежно від того, чи є парним або непарним кількість бітів 1 в байті, якому відповідав би цей біт контролю парності. Якщо в контрольованому байті деякі біти будуть спотворені, значення біта парності не співпаде із значенням, що відповідає новому складу бітів 1 в цьому байті. Аналогічним чином, при спотворенні зберігаємого біта контролю парності він не буде відповідати даним в байті, що дозволить виявити помилку. З іншого боку, схеми коригування помилок передбачають зберігання двох або більше додаткових бітів і дозволяють відновлювати початкові дані, якщо один з бітів буде спотворений. Схеми контролю парності і коригування помилок можуть застосовуватися при смуговому розподілі даних по дисках.

У RAID-масивах використовуються різні поєднання описаних вище методів підвищення продуктивності і надійності, що отримали назву рівнів RAID. Ці рівні перераховані нижче.

- RAID 0 (не надлишковий масив). На цьому рівні не застосовується дублювання даних і тому забезпечується найвища продуктивність запису, оскільки не доводиться копіювати по дискам оновлювані дані. Смоговий розподіл даних здійснюється на рівні дискових блоків.
- RAID 1 (масив з дзеркальним відображенням). На цьому рівні ведуться дві ідентичні (дзеркальні) копії даних на різних дисках. Для забезпечення збереження даних на випадок відмови диска запис на різні диски в деяких варіантах реалізації такого масиву не виконується одночасно. Цей варіант організації зберігання даних у зовнішній пам'яті є найбільш дорогим.
- RAID 0 + 1 (не надлишковий масив з дзеркальним відображенням). На цьому рівні застосовується поєднання методів смогового розподілу і дзеркального відображення даних.
- RAID 2 (масив із застосуванням кодів коригування помилок, що зберігаються у зовнішній пам'яті). На цьому рівні одиницею смогового розподілу є один біт і для реалізації схеми надмірності застосовуються коди Хеммінга.
- RAID 3 (масив, що забезпечує контроль парності з чергуванням бітів). На цьому рівні передбачається зберігання надлишкових даних, що представляють собою інформацію контролю парності на окремому диску масиву. Ця інформація може застосовуватися для відновлення даних, що зберігаються на інших дисках, в разі відмови цих дисків. На цьому рівні використовується менший додатковий обсяг простору зовнішньої пам'яті в порівнянні з рівнем RAID 1, але доступ до диска з інформацією контролю парності може стати проблемним місцем, що обмежує продуктивність.
- RAID 4 (масив, що забезпечує контроль парності з чергуванням блоків). На цьому рівні одиницею смогового розподілу є блок диска; блоки з інформацією контролю парності зберігаються на іншому диску, ніж відповідні блоки даних з декількох інших дисків. При відмові одного з дисків з даними блок контролю парності може застосовуватися в поєднанні з відповідними блоками з інших дисків для відновлення даних, які зберігалися на диску який відмовив.
- RAID 5 (масив, що забезпечує контроль парності з чергуванням блоків і розподілом інформації контролю парності). На цьому рівні інформація контролю парності застосовується в якості надлишкової і забезпечує відновлення початкових даних за таким же принципом, як в масиві RAID 3, але дані контролю парності розподіляються за допомогою методу смогового розподілу по всіх дисках таким же чином, як відбувається розподіл вихідних даних. Це дозволяє усунути проблемне місце, що виникає, якщо вся інформація контролю парності зберігається на одному диску.
- RAID 6 (масив з надмірністю $P + Q$). Цей рівень аналогічний рівню RAID 5, але передбачає зберігання додаткових надлишкових даних для захисту від відмови

відразу декількох дисків. При цьому замість інформації контролю парності використовуються коди виправлення помилок.

Наприклад, корпорація Oracle рекомендує використовувати рівень RAID 1 для файлів журналу відновлення. А для файлів бази даних рекомендується застосування рівня RAID 5, якщо він забезпечує прийнятні затримки при запису, а в іншому випадку рекомендується рівень RAID 1 або RAID 0+1.

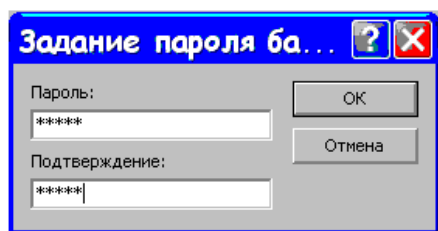
Заходи захисту СУБД Microsoft Access

Microsoft Access 2000 надає наступні два методи захисту бази даних:

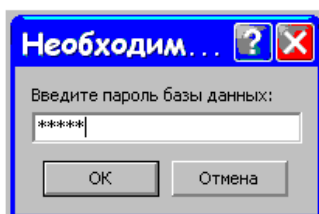
- встановлення пароля, який застосовується при відкритті бази даних (цей метод в термінології Microsoft Access називається захистом системи);
- застосування засобів захисту на рівні користувача, які можуть застосовуватися для визначення тих частин бази даних, в яких користувач може виконувати операції читання або оновлення (це засіб в термінології Microsoft Access називається захистом даних).

Встановлення пароля

Найпростішим методом захисту є встановлення пароля, що застосовується для відкриття бази даних. Після встановлення пароля (в меню Tools - Security) при будь-якій спробі відкрити базу даних на екрані з'являється діалогове вікно із запрошенням ввести пароль. Дозвіл відкрити базу даних отримують тільки ті користувачі, які вводять правильний пароль. Цей метод є надійним, оскільки СУБД Microsoft Access шифрує пароль таким чином, щоб його не можна було визначити, безпосередньо зчитуючи файл бази даних, але після відкриття бази даних всі об'єкти, що містяться в ній, стають доступними для користувача. На рис. 10.3а показано діалогове вікно для встановлення пароля, а на рис. 10.3б - діалогове вікно із запрошенням ввести пароль, який з'являється на екрані при кожній спробі відкрити базу даних, захищену паролем.



а



б

Рис.10.3. Захист бази даних із застосуванням паролю:

- а) діалогове вікно Set Database Password (Задання паролю бази даних);
б) діалогове вікно Password Required (Необхідно ввести пароль),
яке відкривається під час відкриття програми.

Захист на рівні користувача

Засоби захисту на рівні користувача в СУБД Microsoft Access аналогічні засобам, які застосовуються в більшості мережесистем. При запуску програми

Microsoft Access користувачі повинні вказати свій ідентифікатор і ввести пароль. У файлі з інформацією про робочі групи програми Microsoft Access користувачі позначаються як члени певної групи. У СУБД Access передбачені за замовчуванням дві групи: адміністратори (група Admins) і користувачі (група Users), мають місце бути і додаткові групи.

Групам і користувачам надаються права доступу, які дозволяють регламентувати перелік допустимих для них операцій з кожним об'єктом бази даних. Для цього застосовується діалогове вікно Дозволи (User and Group Permissions). У табл. 10.2 наведено перелік прав доступу, які можуть бути встановлені в СУБД Microsoft Access.

Таблиця 10.2. Права доступу в СУБД Microsoft Access

Права доступу	Опис допустимих операцій
Open/Run	Відкривати базу даних, форму, звіт або викликати макрос для виконання
Open Exclusive	Відкривати базу даних з ексклюзивними правами доступу
Read Design	Переглядати об'єкти в представленні Design
Modify Design	Стосовно до баз даних: встановлювати пароль бази даних, копіювати бази даних і модифікувати сценарії запуску
Administer	Стосовно об'єктів бази даних: повний доступ, в тому числі можливість призначати права доступу
Read Data	Переглядати дані
Update Data	Переглядати і модифікувати дані (але не вставляти і видаляти)
Insert Data	Переглядати і вставляти дані (але не модифікувати і видаляти)
Delete Data	Переглядати і видаляти дані (але не вставляти і модифікувати)

Питання:

10.1. Яке призначення і область застосування поняття "захисту бази даних"?

10.2. Перерахуйте основні типи загроз, яким можуть піддаватися системи з базами даних, і вкажіть для кожної з них можливі засоби контролю і протидії.

10.3. Поясніть сенс наступних понять з точки зору захисту бази даних:

- авторизація користувачів;
- представлення;
- резервне копіювання і відновлення;
- цілісність даних;
- шифрування;
- технологія RAID.

10.4. Які методи захисту СУБД Microsoft Access існують?

ТЕМА 11

Напрямки розвитку баз даних і СУБД

Управління інформацією завжди було основною сферою застосування комп'ютерів і буде відігравати велике значення у майбутньому.

Системи управління базами даних СУБД протягом всього шляху розвитку комп'ютерної техніки удосконалювалися, підтримуючи все більш складні рівні абстрактних даних, заданих користувачем, і забезпечуючи взаємодію компонентів, розподілених в глобальних мережах і поступово інтегруючись з телекомунікаційними системами.

Серед основних класифікаторів СУБД можна виділити наступні за ознаками:

- тип моделі даних: ієрархічні, мережеві і реляційні;
- місце зберігання бази даних: централізоване і розподілене;
- об'єктні технології: об'єктно-орієнтовані і об'єктно-реляційні.

11.1 Розподілені системи управління базами даних (СУБД)

Зростання баз даних, використання їх в територіально рознесених організаціях призвело до того, що централізовані СУБД стали погано справлятися з ростом числа оброблюваних транзакцій. Це призводило до зниження загальної надійності і продуктивності системи при обробці запитів користувачів.

В якості можливого вирішення цих проблем була здійснена децентралізація даних.

Якщо з БД працюють одночасно декілька користувачів, то в цьому випадку СУБД повинна забезпечувати коректну паралельну роботу всіх користувачів над одними і тими ж даними.

Розрізняють розподілену обробку і розподілені БД.

Розосереджена або, якщо правильно називати, **розподілена база даних** - це набір логічно пов'язаних між собою сукупностей поділених даних і їхніх описів, які фізично розподілені в деякій комп'ютерній мережі, це така БД, до складу якої входить кілька комп'ютерів, пов'язаних мережею, на кожному з яких працює локальна база даних.

Сукупність усіх цих програмно-апаратних засобів створює **розподілену СУБД** - програмний комплекс, призначений для управління розподіленими базами даних і забезпечення прозорого доступу користувачів до розподіленої інформації.

Розподілені бази даних ззовні виглядають як звичайні локальні бази, їх апаратна рознесеність не помітна користувачам. Розподілена система управління контролює всі вузли БД і забезпечує зв'язність даних.

З виникненням проблеми нездатності системи з одним процесором задовольняти зростаючі вимоги до масштабованості, надійності і продуктивності обробки даних. ефективною і економічно обґрунтованою альтернативою однопроцесорних СУБД стали паралельні СУБД, функціонуючі одночасно на декількох процесорах.

Слід розрізняти відмінності, існуючі між розподіленою і паралельною СУБД.

Розподілена обробка - обробка з використанням централізованої бази даних, доступ до якої може здійснюватися з різних вузлів мережі.

Паралельна СУБД - система управління базою даних, яка функціонує з використанням декількох процесорів і жорстких дисків, що дозволяє їй (якщо це можливо).

Застосування паралельної СУБД дозволяє об'єднати декілька малопотужних машин для отримання такого ж рівня продуктивності, як і у разі однієї, але більш могутньої машини, з додатковим вигоном в масштабованість і надійності системи в порівнянні з однопроцесорною СУБД.

Для надання декільком процесорам спільного доступу до однієї і тієї ж бази даних паралельна СУБД повинна забезпечувати управління спільним доступом до ресурсів. Те, які саме ресурси розділяються і як це розділення реалізовано на практиці, безпосередньо впливає на показники продуктивності і масштабованість системи, що створюється, що, в свою чергу, визначає придатність конкретної СУБД до умов заданої обчислювальної середовища і вимог додатків.

Головними перевагами розподілених СУБД можна визначити:

- Відображення структури організації
- Висока ступінь роздільності та локальної автономності
- Підвищення доступності даних
- Підвищення надійності
- Підвищення продуктивності
- Економічні вигоди
- Модульність системи

Серед недоліків розподілених СУБД визначають:

- Підвищення складності
- Збільшення вартості
- Проблеми захисту
- Ускладнення контролю за цілісністю даних
- Відсутність стандартів
- Брак досвіду
- Ускладнення процедури розробки бази даних

Принципи організації і роботи глобальних мереж

Комп'ютерна мережа – сукупність комп'ютерів та інших пристроїв, з'єднаних між собою для обміну даними і спільного використання пристроїв.

Глобальні мережі (WideArea Network, WAN) – це телекомунікаційні структури, що об'єднують локальні комп'ютерні мережі, які мають загальний протокол зв'язку, методи підключення і протоколи обміну даними. Кожна з глобальних мереж (INTERNET, BITNET, DECNET і ін.) організовувалася для певних цілей, а надалі розширювалася завдяки підключенню локальних мереж, що використовують її послуги і ресурси.

Мережевий протокол - набір правил, що визначає способи передачі, інтерпретації та обробки повідомлень між комп'ютерами

У переважній більшості комп'ютерних мереж обмін даними здійснюється на основі сімейства (стеку) протоколів **TCP/IP** (англ. **Transmission Control Protocol** —



протокол керування передаванням; IP — Internet Protocol — між-мережевий протокол).

Протокол TCP забезпечує зв'язок між комп'ютерами та керує передаванням даних, визначає правила розбиття даних на пакети, доставку адресату та об'єднання пакетів в єдине ціле.

На основі протоколу IP здійснюється пересилання пакетів між комп'ютерами. Всі протоколи будуються за багаторівневим принципом. Протокол певного рівня визначає одне з технічних правил зв'язку.

У сімействі протоколів TCP/IP є чотири рівні

(рис.2)

Крім TCP/IP, в глобальній мережі часто використовують протоколи:

HTTP (Hyper Text Transfer Protocol – передача гіпертексту) використовується при передачі Web-сторінок з одного ПК на інший. **FTP (File Transfer Protocol** – передача

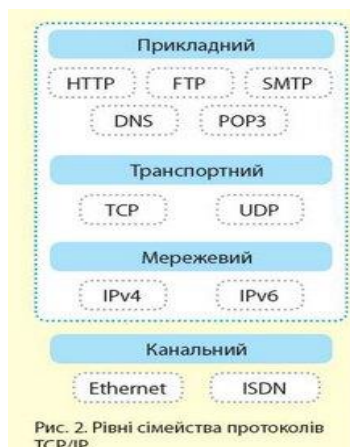


Рис. 2. Рівні сімейства протоколів TCP/IP

файлів з файлового сервера на ПК користувача) надає можливість користувачеві обмінюватися файлами з будь-якими комп'ютерами в мережі.

POP (Post Office Protocol) – протокол поштового з'єднання) дозволяє обробляти вхідну пошту.

SMTP (Simple Mail Transfer Protocol) створює звід правил для передачі пошти; **UUCP (Unix to Unix Copy Protocol)** протокол передачі інформації, та для електронної пошти, у тому числі.

TELNET (протокол віддаленого доступу) надає можливість користувачам мережі працювати на будь-якому комп'ютері

мережі. Практично ця можливість регламентується рівнем доступу, заданим адміністратором кожної віддаленої машини.

DTN призначений для наддалекого космічного зв'язку.

Мережеві протоколи за рівнями призначення розподіляються:

Фізичного рівня: RS-485, ISDN, RS-232, EIA-422.

Канального рівня: Token ring, Ethernet, FDDI, GVRP, HDLC, PPP, L2TP, PPTP, xDSL, ATM.

Мережного рівня: IPv4, IPv6, ICMP, ARP, IPX. Транспортного рівня: SPX, TCP, DHCP (в моделі OSI), SCTP, UDP (Unreliable/User Datagram Protocol), RTCP, RUDP (Reliable User Datagram Protocol), RDP (Reliable Data Protocol).

Сеансового рівня: SSL.

Представлення даних: XML-RPC, ASN.1, XDR, TDI, FTP, SNMP, Telnet, NCP, SMTP. Прикладного рівня: binkp, Finger, FTP, Gnutella, DNS, HTTP, Gopher, IMAP, HTTPS, XMPP, IRC, NTP, LDAP, POP3, NNTP, SSH, RDP (Remote Desktop Protocol), Telnet, SMTP, SIP, SNMP.

Функції та архітектура СУБД

Перелік служб, які повинні бути реалізовані в будь-якій повномасштабній СУБД:

- Зберігання, видалення та оновлення даних
- Каталог, доступний кінцевим користувачам
- Підтримка транзакцій
- Служби управління паралельною роботою
- Служби відновлення
- Служби контролю доступу до даних
- Підтримка обміну даними
- Служби підтримки цілісності даних

Крім того, розподілена СУБД повинна мати наступний набір функціональних можливостей:

- Розширені служби установки з'єднань повинні забезпечувати доступ до віддалених вузлів і дозволяти передавати запити і дані між вузлами, що входять до мережі.
- Розширені засоби ведення каталогу, що дозволяють зберігати відомості про розподіл даних у мережі.
- Засоби обробки розподілених запитів, включаючи механізми оптимізації запитів і організації віддаленого доступу до даних.
- Розширені функції управління захистом, що дозволяють забезпечити дотримання правил авторизації прав доступу до розподілених даних.
- Розширені функції управління паралельним виконанням, що дозволяють підтримувати цілісність копійованих даних.
- Розширені функції відновлення, враховують ймовірність відмов у роботі окремих вузлів і ліній зв'язку.

Рекомендована архітектура

Розподілені СУБД мають безліч відмінностей, які досить складно відобразити в деякому еквівалентному архітектурному рішенні, прийнятному для більшості випадків. Але було б корисно знайти які-небудь рекомендовані рішення, що враховують особливості роботи з розподіленими даними.

Рекомендовані наступні компоненти:

- набір глобальних зовнішніх схем;
- глобальна концептуальна схема;
- схема фрагментації і розподілу;
- набір схем для кожної локальної СУБД, що відповідає вимогам трирівневої архітектури ANSI-SPARC додати в презентацію и додати NoSQL
- Відповідність NoSQL— база даних , яка забезпечує механізм зберігання та видобування даних відмінний від підходу таблиць-відношень в реляційних базах даних.

Компонентна архітектура

Компонентна архітектура розподіленої СУБД, включає наступні основні компоненти: локальна СУБД, компонент передачі даних. Глобальний системний каталог, розподілена СУБД.

Архітектура СУРБД, повинна включати чотири наступних найважливіших компонента:

- локальну СУБД;
- компонент передачі даних;
- глобальний системний каталог;
- розподілену СУБД (СУРБД).

Загальний вигляд компонентної архітектури розподіленої СУБД представлений на малюнку.



Компонент локальної СУБД являє собою стандартну СУБД, призначену для управління локальними даними на кожному з сайтів, вхідних до складу розподіленої бази даних. Локальна СУБД має свій власний системний каталог, в якому міститься інформація про дані, що зберігаються на цьому сайті.

Компонент передачі даних являє собою програмне забезпечення, що дозволяє всім сайтам взаємодіяти між собою. Він містить відомості об існуючому сайтах і лініях зв'язку між ними.

Глобальний системний каталог має те ж саме функціональне призначення, що і системний каталог в централізованих базах даних. Глобальний каталог містить

інформацію, специфічну для розподіленої природи системи, наприклад схеми фрагментації і розподілу. Цей каталог сам по собі може бути розподіленою базою даних і тому зазнавати фрагментації і розподілу, бути повністю репліцируемым або централізованим, як і будь-яке інше відношення.

Компонент розподіленої СУБД є керуючим по відношенню до всієї системи елементом.

Розробка розподілених реляційних баз даних

Існують чотири альтернативні стратегії розміщення даних у системі:

- Централізоване;
- Роздільне (фрагментоване);
- З повною реплікацією
- З виборчою реплікацією.

Фрагментація - база даних розбивається на непересічні фрагменти, кожен з яких розміщується на одному з вузлів системи. Якщо елемент даних буде розміщений на тому сайті, на якому він найчастіше використовується, отриманий рівень локалізації посилань високий.

Існують два основних типи фрагментації: горизонтальна і вертикальна.

Горизонтальні фрагменти являють собою підмножини рядків відносин, а вертикальні — підмножини атрибутів відносин.

Крім того, існують ще два типи фрагментації: змішана і похідна (представляє собою варіант горизонтальної фрагментації).

Забезпечення прозорості в розподіленої СУБД

Прозорість розміщення бази даних дозволяє кінцевим користувачам сприймати базу даних як єдине логічне ціле. Якщо розподілена СУБД забезпечує прозорість розміщення, то користувачеві не потрібно враховувати фрагментацію даних (прозорість фрагментації) або їх місцезнаходження (прозорість місцезнаходження).

Прозорість транзакцій в середовищі розподілених СУБД означає, що при виконанні будь-яких розподілених транзакцій гарантується збереження цілісності та узгодженості розподіленої бази даних.

Прозорість виконання вимагає, щоб робота в середовищі розподіленої СУБД виконувалася точно так само, як і в середовищі централізованої СУБД. В розподіленому середовищі робота системи не повинна демонструвати зниження продуктивності, пов'язаного з її розподіленою архітектурою, наприклад з наявністю повільних мережеских з'єднань.

Прозорість використання СУБД дозволяє приховати від користувача розподіленої СУБД той факт, що на різних вузлах можуть функціонувати різні локальні СУБД. Тому даний тип прозорості застосовуємо тільки у випадку гетерогенних розподілених систем.

11.2 Об'єктні системи управління базами даних (СУБД)

Об'єктно-орієнтована технологія побудови СУБД радикально змінила сферу розробки програмного забезпечення вже в 1990-х роках.

Основна перевага об'єктно-орієнтованого підходу полягає в тому, що задача вирішується новим способом. Замість формування набору процедур, які призначені для вирішення конкретної задачі, формується набір об'єктів, які властиві даній предметній області. Якщо такий набір складається грамотно, то не тільки конкретна задача може бути вирішена, але й потенційно закладається фундамент для вирішення всіх задач в даній предметній області. Звичайно при структурному програмуванні такий підхід стихійно складався при розробці програм у виді бібліотек підпрограм за темами. Але об'єктно-орієнтований підхід дає нові механізми. З основні властивості об'єктно-орієнтованого підходу дозволяють створювати дійсно незалежний від задачі опис предметної області в виді набору об'єктів.

Основними властивостями об'єктно-орієнтованого підходу є:

- **Інкапсуляція** - така властивість, при якій об'єкти містять опис атрибутів і дій одночасно;
- **Успадкування** - такий метод визначення об'єктів, при якому похідні об'єкти (нащадки) успадковують властивості (атрибути і дії) від своїх батьків;
- **Поліморфізм** - така властивість об'єктів, при якій дії з однаковими іменами викликають різну поведінку для різних об'єктів.



Об'єктно-орієнтовані системи управління базами даних СУБД

Об'єктно-орієнтована модель є подальшим розвитком технології баз даних ГІС. У цьому випадку вся сукупність даних, що буде зберігатися й оброблятися в базі даних, подана не у вигляді набору окремих картографічних шарів і таблиць, а у вигляді об'єктів певного класу. Об'єктно-орієнтована модель поряд з геометричною й атрибутивною інформацією зберігає програмний код, що визначає поведінку об'єктів того чи іншого класу при введенні і редагуванні, аналізі або поданні даних. Класи об'єктів являють собою ієрархічну структуру — під ними розуміють загальний батьківський клас (наприклад, робочий простір), на підставі властивостей якого визначаються й описуються похідні класи (векторні, растрові, TIN-просторові дані).

Об'єктно-орієнтована модель даних (ООМД) — логічна модель даних, яка враховує семантику об'єктів, застосовувану в ООП.

Об'єктно-орієнтована база даних (ООБД) — перманентний, спільно використовуваний набір об'єктів, визначений засобами ООМД.

Виникнення напрямку ООБД визначилося насамперед потребами практики: необхідністю розробки складних інформаційних прикладних систем, для яких технологія попередніх систем БД не була цілком задовільною. Звичайно, ООБД виникли не на порожньому місці. Відповідний базис забезпечують як попередні роботи в області БД, так і давно розвиваються напрямки мов програмування з абстрактними типами даних і об'єктно-орієнтованих мов програмування.

Об'єктно-орієнтована система управління базою даних (ООСУБД) – система управління ООБД.

Вимогами до ООСУБД є:

- Надання функціональних засобів баз даних
- Підтримання ідентичності об'єктів
- Забезпечення інкапсуляції
- Підтримання об'єктів зі складною будовою.

Для розробки ООСУБД визначають наступні підходи:

- **Введення засобів роботи з базою даних в існуючу об'єктно-орієнтовану мову програмування** – це підход у якому традиційні функції бази даних вбудовуються в існуючі об'єктно-орієнтовані мови програмування;
- **Доповнення існуючої мови бази даних об'єктно-орієнтованими функціями** -завдяки широкому поширенню мови SQL деякі компанії-розробники програмного забезпечення намагаються розширити його з метою підтримки об'єктно-орієнтованих конструкцій.
- **Розробка нової мови бази даних або моделі даних** - найбільш радикальний підхід, який, починаючи з нуля, призводить до створення абсолютно нової мови баз даних і СУБД, що володіють об'єктно-орієнтованими можливостями.

Сучасні системи баз даних мають бути забезпечені наступними компонентами і функціональними засобами:

- Модель даних – особливий спосіб опису даних і зв'язків між ними.
- Перманентність даних – дані повинні існувати поза періоду виконання програми.
- Спільне використання даних – доступність даних одночасно декільком додаткам.

Основними напрямками розвитку ООСУБД визначають:

- Надійність – гарантія захисту даних від збоїв.
- Масштабованість – прості засоби для роботи з різними обсягами даних.
- Безпека і цілісність – захист від несанкціонованого доступу, а також від суперечливості.
- Розгалуженість – можливість фізичного розподілу логічно пов'язаних даних по мережі.

11.3 Об'єктно-реляційні системи управління базами даних СУБД

Одним з важливих етапів розвитку в області технологій баз даних є **об'єктно-реляційна модель**. Об'єктно-реляційна база даних (ОРБД) поєднує в собі реляційну модель (реляційна база даних це база даних, побудована на

основі реляційної моделі, тобто БД, що має табличний спосіб вистави даних, а на зовнішньому рівні, що задається набором однорідних таблиць. Кожний об'єкт записується рядком у таблиці. Рядок називається записом. Запис складається з полів різного типу) і об'єктну модель. Та сама об'єктно-реляційна система керування базами даних (СУОРБД) здатна обробляти реляційні структури даних (реляційні таблиці) і об'єктні структури даних (об'єктні таблиці).

Об'єктно-реляційна БД – розширена реляційна БД, функціональність якої також включена робота з об'єктами.

Перевагами ОРСУБД є:

- **Повторне та спільне використання компонентів** - ґрунтується на можливості доповнення сервера СУБД стандартними функціями, які виконуються централізовано і не входять у вигляді виконуваного коду до складу кожного додатку.
- **Розширений реляційний підхід** дозволяє скористатися великим обсягом накопичених знань і досвіду, пов'язаних з розробкою реляційних додатків.
- **Складність і пов'язані з нею додаткові витрати** – деякі прихильники реляційного підходу впевнені, що простота і ясність, притаманна реляційній моделі, при використанні розширень подібних типів втрачається.

Серед недоліків ОРСУБД визначають:

- **В об'єктно-реляційних системах спотворюється об'єктна термінологія** – замість моделі об'єктно-реляційної технології використовується поняття користувацьких типів даних, тоді як об'єктно-орієнтований підхід має справу з абстрактними типами, ієрархією класів та об'єктною моделлю.

11.4 Документо-орієнтовані СУБД

Документо-орієнтовані бази даних (ДОБД)- набір перманентних даних, що використовуються прикладними програмними системами.

Документо-орієнтована система управління базами даних (ДОСУБД) —система управління базами даних, спеціально призначена для зберігання ієрархічних структур даних (документів) і зазвичай реалізована за допомогою підходу NoSQL.

Тенденції використання ДОБД

Документо-орієнтовані бази даних використовуються для: аналітичної обробки великого обсягу даних і на сайтах з дуже високим навантаженням, оскільки легко масштабуються горизонтально. Часто ДОБД використовуються спільно з традиційними рішеннями на реляційних базах даних. ДОБД не є заміною реляційних баз даних, це скоріше альтернатива. Це інструмент, який може робити те ж, що можуть робити безліч інших. Дещо - краще, дещо - ні.

Переваги ДОБД

- У порівнянні з реляційними базами даних найкраща продуктивність при індексуванні великих обсягів даних і великої кількості запитів на читання.
- Легше масштабуються у порівнянні з SQL рішеннями.
- Децентралізовані.

- Легко міняти "схему" даних: не потрібно виконувати ніяких операцій оновлення для додавання нових полів
- Немає проблем із зберіганням неструктурованих даних.
- Єдине місце зберігання всієї інформації про об'єкт.
- Простий інтерфейс спілкування з БД (ключ → значення, немає SQL).

Недоліки ДОБД

- Відсутність транзакційної логіки і контролю цілісності більшості реалізацій: необхідно реалізовувати її в логіці програми. З іншого боку, спеціалізована логіка може виявитися ефективніше загальних алгоритмів реляційних ДБ.
- Для обробки даних необхідно використання додаткової мови програмування. Для програмістів, які цю мову знають, мінус перетворюється в плюс. Крім того, іноді для цих цілей можна використовувати різні мови.

Приклади ДОСУБД

Прикладами документо-орієнтованих СУБД є:

- MongoDB;
- CouchDB;
- Oracle NoSQL;
- IBM Lotus Notes;
- Berkeley DB;
- Mnesia;
- Redis та ін.

11.5 Тенденції розвитку СУБД

В даний час в світі використовується досить велика кількість універсальних промислових СУБД. Серед них можна виділити трьох безперечних лідерів (як за рівнем розвитку технологій, так і за обсягом ринку – вони разом займають більше 90% світового ринку СУБД). Це СУБД першого ешелону – Oracle, Microsoft SQL Server, IBM DB2. Список СУБД другого ешелону досить великий, сюди відносять такі СУБД, як Sybase, Informix, Ingress, Adabas, Interbase, Progress, Postgres, Cache, Linter, Firebird, Teradata і т. д. Існують також невеликі СУБД для нішевих (спеціалізованих) рішень і постійно з'являються прототипи нових спеціалізованих СУБД (об'єктно-орієнтовані СУБД, XML СУБД, СУБД для обробки потокових даних, СУБД для роботи з текстами і т. д.).

Хоча продуктивність СУБД важлива, але користувачі вибирають СУБД, орієнтуючись на сукупність характеристик. Зазвичай враховуються такі характеристики, як забезпечення високої надійності та безперервності роботи, масштабованість, безпека, простота управління, простота розробки, можливість роботи з великими БД, підтримка спеціальних схем, алгоритмів і типів даних (DW, OLTP, XML, GIS, тексти і т. д.), підтримка стандартів, підтримка національної мови. Крім того, дуже важлива поширеність даної СУБД в країні, наявність і можливість навчання фахівців (адміністраторів, розробників), наявність великого числа вдалих впроваджень СУБД для додатків схожого типу (references) і т. д. Тут спеціалізовані СУБД конкурувати з лідерами ринку не можуть.

Найбільш важливими тенденціями розвитку сучасних універсальних комерційних СУБД на найближчий час будуть:

- Віртуалізація ресурсів та GRID-технології;
- Вбудовування Information Life Cycle Management (ILM) в СУБД;
- Самоврядування, самодіагностика, самолікування;
- Real Application Testing – механізми промислового тестування версій і змін
- Вдосконалення архітектур максимальної доступності (різні режими Standby, Active standby, Snapshot standby)
- Підтримка нових типів даних (XML, RFID, Semantic Web, геном, медицина, швидкі LOB і т.д)
- Розумні механізми стиснення і дедублювання
- Хмарні обчислення (Cloud computing)

Віртуалізація ресурсів та GRID-технології

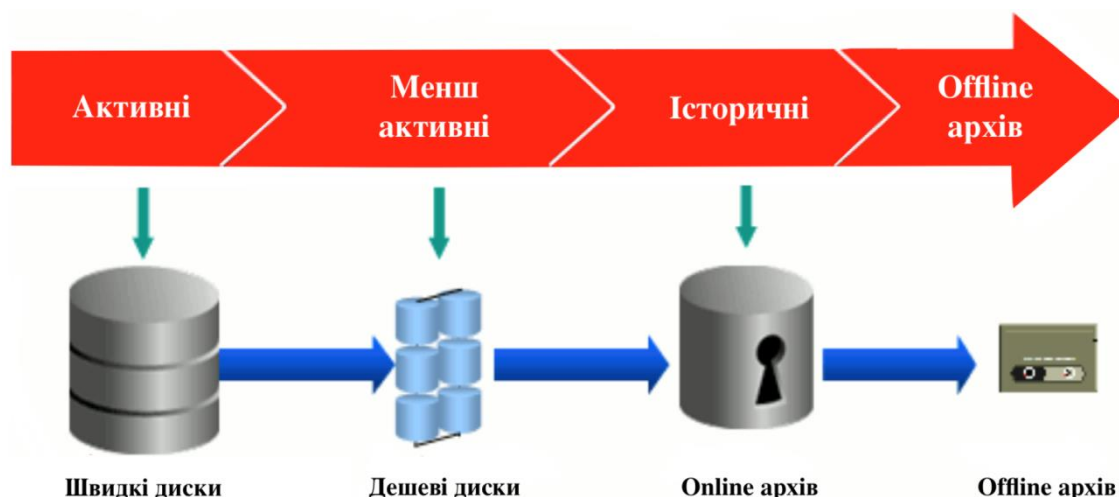
В даний час у більшості організацій для кожного додатка виділяється свій комп'ютер (група комп'ютерів) і свій набір дисків для зберігання БД. Як правило, ці комп'ютери та диски різними додатками спільно не використовуються. Оскільки в реальному житті навантаження на програми постійно змінюється, ми часто маємо ситуацію, коли одні комп'ютери сильно перевантажені, в той час як інші недовантажені або простоюють. В одних системах є запас дисків, а в інших дискового простору катастрофічно не вистачає. Однак, на жаль, ми не можемо оперативно перекидати дискові і процесорні ресурси туди, куди треба. Наприклад, якщо протягом кварталу йде великий потік транзакцій, а в кінці кварталу він спадає, але починає зростати

навантаження на систему створення корпоративної звітності, то ми змушені на обидві системи тримати максимальні обчислювальні потужності. Система виходить дорогою, надлишковою, негнучкою. Виходом є віртуалізація ресурсів. При цьому всі диски центру обробки даних (ЦОД) об'єднуються в один великий віртуальний диск, на якому містяться всі дані всіх додатків, і ми можемо гнучко на ходу збільшувати цей віртуальний диск, додаючи до нього нові фізичні диски, або легко віддавати місце для зберігання тим додаткам, яким воно зараз необхідно. Та ж картина і з процесорним ресурсом. Об'єднання безлічі комп'ютерів в один віртуальний комп'ютер, на якому одночасно працює безліч додатків, потужність якого можна легко збільшувати, на льоту додаючи нові комп'ютери, і можливість перерозподілу обчислювальних ресурсів цього суперкомп'ютера між додатками по мірі необхідності дозволяють різко підвищити гнучкість обчислювальної системи, знизити її вартість і збільшити ефективність використання обладнання. Такий підхід дуже привабливий для організацій, він дозволяє створювати обчислювальний ресурс необмеженої потужності і не піклуватися про те, на яких комп'ютерах реально працює ваша програма. Головне – запросити й одержати той обчислювальний ресурс, який вам потрібен. Система сама виконає балансування навантаження для вашого додатку, створить, якщо треба, БД з урахуванням забезпечення дзеркалювання даних і зниження навантаження щодо введення/висновку, задіє стільки елементів GRID, скільки необхідно для забезпечення необхідної вам надійності і продуктивності. І, по мірі зміни зовнішніх умов і вимог, вона буде підлаштовуватися під ці вимоги. Сьогодні вже реалізовані такі Enterprise GRID першого покоління на Oracle (Amazon, e-Bay Латинської Америки, EDS ABNAMRO і т. д.), компанія IBM активно проводить дослідження в області GRID-технологій для наукових обчислень, існує спеціальний інструментарій Globus Toolkit для використання Grid-технологій при розробці додатків. Однак підхід Oracle, який не вимагає використання спеціальних інструментів і підходів для розробки корпоративних додатків, які будуть працювати в середовищі GRID, здається більш перспективним. Основою Database GRID у Oracle є архітектура Real Application Clusters, що реалізує підхід shared disk (всі вузли кластера одночасно працюють з єдиною БД). Ще одним підтвердженням перспективності підходу послужила інформація про розробку компанією Sybase продукту Sybase ASE Cluster Edition, який дуже схожий на Oracle RAC, схожу архітектуру реалізує і IBM DB2 для Mainframe.

Вбудовування Information Life Cycle Management (ILM) в СУБД:

Як відомо, дані та інформація мають свій цикл життя (рис.11.1). Спочатку вони надходять в систему (БД) і є активними, тобто використовуються дуже часто. Потрібно забезпечити швидкий доступ до інформації та зберігати її на швидких

дорогих дисках. Потім інформація починає застарівати, переходить у розряд менш



активної. Вона все ще повинна бути під рукою, але її вже можна перемістити на більш дешеві і повільні диски.

Рис. 11.1 Цикл життя даних та інформації

Потім інформація переходить в розряд історичної. За законом ми повинні зберігати її певну кількість років (або вічно). Після закінчення терміну зберігання інформацію треба видаляти, щоб звільнити місце в БД. Потрібно мати можливість легко відтворювати старі звіти на базі цієї інформації. Крім того, дані історичної інформації не можна змінювати, і треба гарантувати їх незмінність. Треба також гарантувати їх збереження протягом заданого терміну. Не всім користувачам і додаткам дозволяється працювати з історичною інформацією, потрібен контроль за тим, хто і як її використовує і т. д. і т. п. Оскільки запити до історичних даних виконуються рідко, а їх обсяг досить великий, часто вдаються до стиснення історичних даних. Ну, і в кінці свого життя дані часто переносяться у офлайнові архіви на стрічках. Підтримка життєвого циклу інформації та даних різного типу, як правило, сьогодні реалізується на рівні логіки додатків. Це складно, негнучко, ускладнює програми і знижує якість управління інформацією. У результаті важливі дані можуть бути втрачені або займати великий обсяг на дорогих дисках. Часто управління життєвим циклом даних реалізують виробники систем зберігання. Але, оскільки вони працюють саме з даними (а не з інформацією) і не знають бізнес-логіки та бізнес-характеристик інформації, що реалізують підтримку саме життєвого циклу даних (DLM), а не інформації (ILM). Тому сьогодні намітилася тенденція реалізації механізму ILM на рівні самої СУБД. Адміністратори або розробники описують різні області зберігання даних з різними характеристиками (швидкі диски, повільні диски, стрічки, флеш-пам'ять, CD тощо) і правила поділу даних одного типу на групи (наприклад, по мірі старіння, за частотою використання тощо). Групи прив'язуються до областей зберігання, і при зміні характеристик даних вони автоматично переміщуються в іншу групу, а отже, і в іншу область зберігання. При цьому змінюються характеристики цих даних, дані можуть стискатися, переводитися в режим "тільки для читання" і т. д. Система буде стежити, щоб дані групи зберігалися необхідний час і потім знищувалися.

Самоврядування, самодіагностика, самолікування:

СУБД стають все складніше і складніше, і для їхньої швидкої і надійної роботи потрібно все більш досвідчені адміністратори. Але навіть вони не можуть оперативно реагувати на постійно мінливе навантаження СУБД, зміна режиму роботи додатків і т. д. Крім того, зростає число СУБД в організації, які треба адмініструвати. Єдиний вихід вирішити ці проблеми – зробити СУБД самоврядними, самодіагностованими, самоліковними. Більшість виробників СУБД вже йдуть цим шляхом, але можливостей розвитку залишається ще багато. Основна ідея тут проста. Під час роботи СУБД постійно збирає інформацію про свою роботу, аналізує її (оперативно і у фоновому режимі), приймає рішення, і автоматично їх реалізує (наприклад, змінює розміри областей пам'яті, виконує процедури, запитує в ОС додаткові ресурси, змінює плани виконання запитів), або інформує адміністратора про проблеми, рекомендує йому послідовність дій для рішення проблеми і після схвалення виконує ці дії. Системи контролю і самоналаштування можуть налаштовувати розміри областей пам'яті, введення/висновки, давати рекомендації по оптимізації запитів, виявляти проблеми безпеки (наприклад, незакриті порти, незмінені паролі і т. д.) і продуктивності, нестачі місця і ресурсів, відхилення від заданих або типових стандартів поведінки і т. д. У фоновому режимі СУБД постійно перевіряє свій стан, цілісність даних, шукає і виправляє биті блоки і файли, звіряє контрольні суми і т.д. Ведеться робота по влаштуванню алгоритмів data mining для виявлення та контролю залежностей між даними. Багато СУБД навчилися сьогодні автоматично (без участі DBA) виконувати резервне копіювання і відновлення, збирати інформацію про збої і відправляти її в службу технічної підтримки, відстежувати появу нових важливих патчів. Триває вдосконалення оптимізаторів SQL-запитів, вони враховують все більше важливих факторів (наприклад, взаємовплив кардинальності стовпців, параметри обладнання і навколишнього середовища експлуатації тощо).

Real Application Testing – механізми промислового тестування версій і змін

В процесі роботи доводиться постійно змінювати параметри роботи СУБД і структури даних, створювати нові об'єкти БД (індекси, матеріалізовані уявлення тощо), змінювати архітектуру обчислювальної системи (наприклад, переходити від одного комп'ютера до кластера або GRID), виконувати рекомендації служби технічної підтримки, досвідчених колег, керівництва і т.д. На жаль, будь-які з цих необхідних дій можуть призвести не до поліпшення роботи системи, а до погіршення або зупинку роботи. Єдина можливість вирішити цю проблему – дозволити заздалегідь протестувати будь-які з перерахованих змін (ОС, обладнання, структур даних, архітектури) на тестовій системі (до їх виконання на промисловій системі) і оцінити наслідки. Для цього використовується тестова копія БД; у деяких виробників в якості тестової можна тимчасово використовувати резервну БД (Oracle Snapshot Standby). Існує багато пакетів, що дозволяють імітувати навантаження на тестовій системі для виконання функціонального і навантажувального тестування. Проте всі ці пакети не створюють реальну, а синтетичне навантаження, все це вимагає тривалого часу для аналізу роботи системи і створення тестових скриптів. І в результаті багато проблем так і залишаються не виявленими. З'явився у Oracle продукт RAT (Real Application Testing), очевидно, незабаром буде реалізований у більшості СУБД, оскільки він

дозволяє реально працює під навантаженням виробничій системі захопити (з мінімальними накладними витратами) реальне навантаження (з урахуванням часу виконання) і відтворити це навантаження на тестовій БД, не встановлюючи там додаток. Захоплення і програвання навантаження легко виконати, вони дозволяють виявити з'явилися/зниклі/змінювалися, після виконання змін, помилки, зміни в планах і результати виконання реальних SQL, проблеми зниження продуктивності як всієї системи, так і окремих запитів. Після аналізу результатів система дає рекомендації щодо поліпшення роботи SQL, дозволяє спробувати різні варіанти оптимізації роботи. Відкочувати тестову БД, тому можна тестувати різні зміни і їх сукупність. І тільки після того, як всі проблеми виявлені і вирішені, можете зробити зміни у виробничому середовищі. Оскільки захоплення, програвання, аналіз, оптимізація проводяться в автоматичному чи автоматизованому режимі, вони не вимагають багато часу від адміністратора БД. Вперше з'явилася можливість значно знизити ризики переходу від старої версії СУБД до нової, більш досконалої. Регулярний апгрейд СУБД стає звичайною рутинною справою.



Рис. 11.2 Механізми промислового тестування версій і зм

Вдосконалення архітектур максимальної доступності (різні режими Standby, Active standby, Snapshot standby)

Вимоги до безперервності роботи СУБД при будь-яких видах збоїв і регламентних робіт постійно підвищуються. Існує цілий набір рішень, які всі разом реалізують архітектуру максимальної доступності. Це і кластери, і можливість автоматичного перезапуску СУБД на іншому сервері, і створення резервних обчислювальних центрів (Standby). Всі ці рішення будуть розвиватися і вдосконалюватися, забезпечуючи і швидку синхронізацію вузлів, і мінімальний час перемикання, і можливість синхронної/асинхронної синхронізації вузлів, і можливість встановлення часу затримки синхронізації (щоб встигнути перехопити помилки), і можливість відкочу баз транзакцій, операцій DDL для боротьби з людськими помилками. Особливо хочеться відзначити тенденцію до більш повного використання можливостей резервної БД. Сьогодні у більшості виробників вона постійно наздоганяє основну БД і використовується тільки тоді, коли основна БД вийде з ладу. Це заморожує інвестиції в обладнання і не дає гарантії того, що, коли

збій нарешті станеться, перемикання буде успішним. Тому виробники починають реалізовувати механізм швидкого перемикання „основна і резервна“ і „резервна–основна“ БД, щоб цей механізм можна було легко і часто використовувати при виконанні регламентних робіт та для поетапного апгрейда СУБД без зупинки її роботи. Крім того, з'являються механізми відкриття резервної БД на читання в той час, як вона наздоганяє основну. Це дозволяє розвантажити виробничі системи, виносячи на резервну БД операції побудови звітності, аналізу, backup. Також резервну БД можна переводити в режим тестування і використовувати для тестування змін. Після закінчення тестування вона повертається в режим резервної БД і продовжує наздоганяти основну БД. Це більш зручно тому, що можна одночасно мати кілька standby баз, іноді навіть в різних містах. Сьогодні більшість складних, ресурсомістких операцій по зміні структури БД або параметрів роботи СУБД може проводитися online, у фоновому режимі. Тобто зміна структури таблиць і індексів, переміщення об'єктів не призводить до уповільнення роботи з цими об'єктами. Online патчі дозволяють модифікувати без зупинки роботи системи.

Підтримка нових типів даних (XML, RFID, Semantic Web, геном, медицина, швидкі LOB і т.д.)

У міру розвитку інформаційних технологій з'являються програми, що працюють з новими типами даних. Сьогодні більшість СУБД вміє працювати не тільки з алфавітно-цифровою інформацією, але і з текстами, аудіо, відео, геоінформацією, XML. Під словом "працювати" ми маємо на увазі не тільки швидке завантаження, ефективне зберігання і швидке вилучення цих даних, але і перетворення даних в різні формати, спеціальні способи індексації та оптимізації, набір типових для даного типу даних функцій і т. д. Останнім часом розпочалася реалізація підтримки в універсальних СУБД таких специфічних типів даних, як RFID, дані семантичних мереж (Semantic Web), дані геному, спеціальні типи даних і алгоритмів для медицини, біології, імунології, генетики. Практично всі універсальні СУБД сьогодні удосконалюють роботу з XML. Також намітилася тенденція до переміщення неструктурованих даних, раніше збережених поруч з СУБД (у файловій системі), всередину СУБД. Користь від використання єдиного механізму доступу, безпеки, надійності і т. д. для структурованих і неструктурованих даних була очевидна давно. Однак СУБД працювали з неструктурованими даними повільніше, ніж файлова система. Сьогодні нові механізми СУБД (такі як SecureFiles Oracle) дозволяють працювати з такими неструктурованими даними не повільніше, а іноді і швидше (за рахунок стиснення, дедублювання, роботи по частинах і т. д.), ніж при їх зберіганні у файловій системі. Зберігання документів, зображень, відео, аудіо і т. д. в БД дозволить спростити розробку додатків і підвищити їх якість.

Розумні механізми стиснення і дедублювання

Обсяги даних сьогодні зростають лавиноподібно. Багато програм зберігають дані за великий період часу, в БД зберігаються об'єкти, документи, відео і т. д. А вартість дискового простору досі досить висока (до 1000\$ за терабайт). Тому більшість виробників СУБД реалізувало механізми стиснення даних в БД. Мова йде про те, наскільки розумні ці механізми стиснення. Справа в тому, що за стиск, розтиснення і зміна стислих даних доводиться платити продуктивністю роботи системи. Тому 455

простежується тенденція до вдосконалення механізмів стиснення. По-перше, для різних даних повинні автоматично застосовуватися різні алгоритми стиснення, а там, де стиснення не дає великого виграшу, воно проводиться не повинно. По-друге, адміністратор повинен мати можливість вибирати різні рівні стиснення, розуміючи, що за економію місця доведеться платити процесорним часом. По-третє, стиснення повинно бути прозоро для програми, тобто додаток не знає про те, стиснуті його дані чи ні. В-четверте, іноді корисний механізм дедублювання, коли система сама виявляє дублі полів і зберігає їх тільки в одному примірнику. По-п'яте, стискати треба вміти все. Не тільки реляційні, але і неструктуровані дані, backup дані мережевого трафіку, дані експорту, індекси. А іноді треба, навпаки, стискати не всі дані таблиці, а тільки рідко використовувану частину таблиці. І СУБД повинна працювати зі стиснутими даними так само ефективно, як і з не стиснутими даними, не витрачаючи час на перебудову таблиць, на розтиснення непотрібних блоків і т. д. Традиційні алгоритми стиснення дозволяють стиснути дані в 2-3 рази. Деякі специфічні види стиснення дозволяють збільшити цю ступінь стиснення на порядок, але при цьому сильно сповільнюється виконання операцій DML з цими даними. Наприклад, відомо, що стиск поколоночно зберігається і заздалегідь відсортованні дані забезпечить дуже високий рівень стиснення (іноді в десятки разів). Однак спроби зміни таких таблиць Oracle 11.2 показали, що час відновлення даних значно збільшується. Але для сховищ даних та історичних даних такий підхід цілком прийнятний.

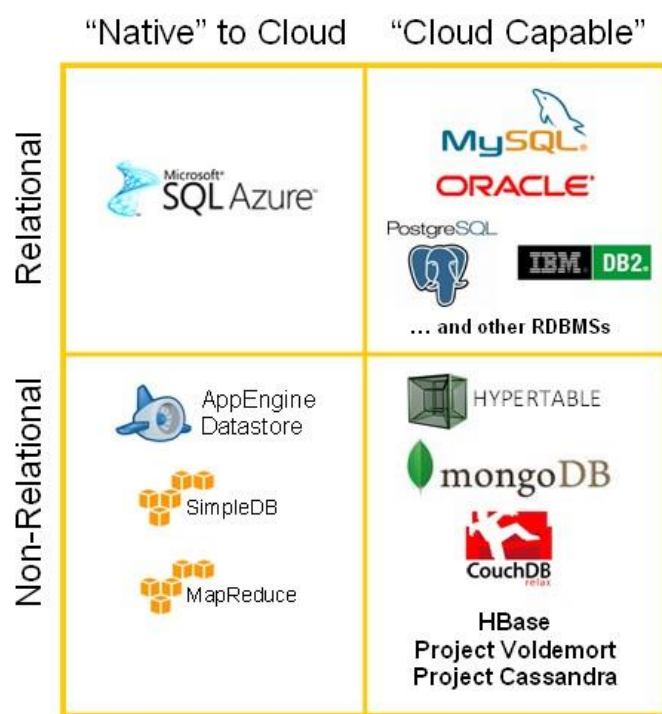


Рис. 11.3 Приклад хмарних обчислювань

Сьогодні починають бурхливо розвиватися так звані хмарні обчислення (Cloud computing). Ця технологія дуже приваблива для користувачів, оскільки вони легко і недорого можуть отримати через Інтернет і отримати у тимчасове користування деякий сервіс для зберігання і обробки даних. Наприклад, можна замовити

комп'ютер з заданим об'ємом пам'яті, дисків, кількістю процесорів, з попередньо встановленими ОС і СУБД або іншими пакетами і працювати з ним зі свого офісу, хоча фізично і обладнання, і розташовуються і обслуговуються у віддаленому чужому ЦОД. Можна замовити потрібний дисковий ресурс і зберігати на ньому свої файли. СУБД повинні вміти працювати в такому середовищі і використовувати цю середу (наприклад, її систему зберігання) для створення і зберігання файлів та копіювання. Робота в середовищі cloud computing пред'являє підвищені вимоги до СУБД з точки зору захисту даних і розмежування доступу (адже дані розміщуються на чужих комп'ютерах), з погляду масштабованості (декларується, що користувач сервісу може замовити машину великої потужності і з великою кількістю процесорів). Оскільки управління системою йде через Інтернет, та й взагалі сам підхід передбачає мінімальні зусилля по управлінню як інфраструктурою, так і самої СУБД, то, очевидно, знадобиться ще більше вдосконалювати засоби адміністрування і підвищити самоврядність СУБД.

Машини баз даних

Зараз почався рух універсальних комерційних СУБД в бік машин баз даних. Перший комерційний приклад – спільна машина баз даних Oracle і HP з осередками зберігання Exadata. Вона забезпечує для звичайних додатків сховищ даних Oracle прискорення обробки даних в десятки разів (до 70 разів) без переписування додатків. Oracle–HP Database Machine являє з себе єдиний преналаштований комп'ютер (з осередками Exadata і кластером БД). Дуже важлива для таких архітектур збалансованість компонент, тобто розміру пам'яті, дисків, числа процесорів, швидкості каналів введення/виведення і т. д. У тому ж напрямку зараз рухається Microsoft, яка купила DATAlegro і інтегрує її з MS SQL Server. Машина баз даних не тільки прискорює обробку, але і спрощує розробку сховищ даних, так як багатьом питань налаштування СУБД, які вимагали створення індексів, матеріалізованих представлень, секціонування таблиць і індексів, кластеризації даних і налаштування областей пам'яті, введення/виведення і т. д., тепер можна приділяти менше уваги. За рахунок дуже високої швидкості і розпаралелювання виконання найважчих операцій СУБД – проекція і join, навіть не дуже добре спроектована БД працює дуже швидко. Так що, очевидно, незабаром всі комерційні

СУБД, що працюють в області сховищ даних, стануть працювати в архітектурі машин баз даних.

	HP Oracle Database Machine	Teradata 2550	Netezza 10100
Announced	Sept 24, 2008	Sept 15, 2008	Dec 4, 2006
Total Storage	168 TB	43 TB	43 TB
System Price	650,000 \$	1,500,000 \$	1,250,000 \$
Price per TB	4000 \$ per TB	35,000 \$ per TB	29,000 per TB
Software License	1,680,000 \$	Included	Included
License Model	Perpetual	Partially Transferrable	Non-Transferrable
Price H/W & S/W	< 14,000 \$ per TB	35,000 per TB	29,000 per TB

	HP Oracle Database Machine	Teradata 2550	Netezza 10100
Footprint	1 rack	1 rack	1 rack
Total Storage	168 TB	43 TB	43 TB
Disks	168 x 1000 GB disks	133 x 300 GB disks	108 x 400 GB disks
Database cores	64 DB Cores	32 DB Cores	4 DB Cores
Storage cores	112 Storage Cores	0 Storage Cores	108 Storage Cores
Total cores	176 Cores	32 Cores	112 Cores
Interconnect	20 GB/sec Infiniband	1 GB/sec BYNET	1 GB/sec Ethernet
Memory	368 GB	128 GB	108 GB
HW Architecture	Exdata	Proprietary	Proprietary

Рис. 11.4 Машини баз даних

Питання:

- 11.1 Які тенденції розвитку?
- 11.2 Що таке віртуалізація ресурсів та GRID-технології?
- 11.3 Поясніть підтримку нових типів даних
- 11.4 Що таке механізми стиснення і дедублювання?

ТЕМА 12

Нереляційні СУБД

12.1 Що таке бази даних NoSQL?

Бази даних *NoSQL* також називають *нереляційними* або «не *SQL*», щоб підкреслити той факт, що вони відрізняються від реляційних баз даних (*SQL*) зі строками та таблицями, які можуть обробляти великі обсяги швидко змінюваної неструктурованої інформації.

Технології *NoSQL* з'явилися ще у 1960-х роках і мали різні назви, але зараз їхня популярність пов'язана з тим, що дані змінюються, а розробникам необхідно адаптуватись до великих обсягів інформації, яка надходить із хмари, соціальних мереж, системи обробки великих обсягів даних та з мобільних пристроїв.

Нові дані та їх типи — від вірусних твітів знаменитостей до життєво важливої інформації в електронних медичних записах — створюються з шаленою швидкістю. Бази даних *NoSQL* призначені допомагати розробникам швидко створювати системи баз даних для зберігання нової інформації, а також зробити їх доступними для пошуку, консолідації та аналізу.

Переваги баз даних NoSQL

Бази даних *NoSQL* допомагають IT-спеціалістам і розробникам впоратися з новими завданнями в умовах зростаючого різноманіття моделей і типів даних. Крім того, вони забезпечують ефективну обробку непередбачуваних даних, часто з дуже високою швидкістю. Вони також дозволяють легко переносити бази даних у хмару для існуючих робочих навантажень *NoSQL*.

Гнучка розробка: Завдяки можливості реагувати на непередбачені ситуації бази даних *NoSQL* підходять для інтенсивних циклів випуску програмного забезпечення, а також для швидкої та гнучкої розробки додатків.

Гнучка обробка даних: *NoSQL* надає розробникам більше свободи, прискорює їх роботу, а також дозволяє змінювати схеми і запити відповідно до вимог до даних. Зберігання даних у вигляді агрегатів допомагає швидко вносити ітеративні покращення, не проектуючи схему заздалегідь.

Робота в будь-яких масштабах: Бази даних *NoSQL* надають привабливі експлуатаційні переваги і допомагають економити завдяки можливості горизонтального масштабування, що дозволяє зменшити кількість дорогих серверів без необхідності в оновленні. Ви можете масштабувати їх для обробки великих обсягів даних або зберігати єдину велику базу даних в легко розподіленому кластері серверів.

12.2 Типи моделей даних з базами даних NoSQL

Більшість нереляційних баз даних високої продуктивності (іноді називаються "не тільки *SQL*") також можуть обробляти дані зі складною структурою. При цьому вони не прив'язані до фіксованих моделей даних, як реляційні бази даних (*SQL*).

Ось чотири найбільш поширені типи баз даних NoSQL:

- **Ключ-значення:** В сховищах типу "ключ-значення" пари ключів і значень формуються за допомогою хеш-таблиці. Бази даних типу "ключ-значення"

найкраще підходять для випадків, коли ключ відомий, а пов'язане з ним значення — ні.

- **Документарні бази даних:** Документарні бази даних — це розширена версія сховищ типу "ключ-значення", в яких цілі документи впорядковані за групами, що називаються колекціями. Вони підтримують вкладені пари "ключ-значення" та запити до будь-яких атрибутів документів.
- **Стовпцеві бази даних:** Бази даних на основі стовпців, широких стовпців або сімейств стовпців забезпечують ефективне зберігання даних і відправку запитів до рядків розріджених даних. Також їх зручно використовувати для запитів до певних стовпців бази даних.
- **Графові бази даних:** У графових базах даних використовується модель на основі вузлів і ребер, що представляють взаємозв'язані дані (наприклад, відносини між людьми в соціальній мережі). Крім того, вони спрощують зберігання та навігацію по складним відносинам.

12.3 Нереляційні дані і бази даних NoSQL

Нереляційна база даних — це база даних, у якій на відміну від більшості традиційних систем баз даних не використовується таблична схема рядків та стовпців.

У цих базах даних використовується модель зберігання, оптимізована під конкретні вимоги типу зберігання даних. Наприклад, дані можуть зберігатися як прості пари «ключ-значення», документи у форматі JSON або граф, який складається з ребер і вершин.

Всі ці сховища даних не використовують реляційну модель. Крім того, вони, як правило, підтримують певні типи даних. Процес запиту даних також специфічний. Наприклад, сховища даних часових рядів розраховані на запити до послідовностей даних, впорядкованих за часом. З іншого боку, сховища графів призначені для аналізу зв'язаних зв'язків між сутностями. Жоден з цих форматів не є ідеальним для виконання завдань управління даними про транзакції.

Термін NoSQL використовується для сховищ даних, які не використовують мову запитів SQL. Замість цього вони запитують дані за допомогою інших мов програмування та конструкцій. На практиці NoSQL означає «нереляційна база даних», навіть незважаючи на те, що багато з цих баз даних підтримують запити, сумісні з SQL. Однак основна стратегія виконання запитів SQL зазвичай відрізняється значними відмінностями в порівнянні з реляційною базою даних.

Нижче наведено основні категорії нереляційних баз даних або баз даних NoSQL.

Сховища даних документів

Сховище даних документів керує набором іменних значень рядкових полів і даних об'єкту в сутності, яка називається документом. Зазвичай дані в цих сховищах зберігаються у виді документів JSON.

Кожне значення поля може представляти собою скалярний елемент, наприклад число або складний об'єкт, наприклад список або колекцію типу «батько-нащадок». Дані в полях документу можна закодувати різним способом, наприклад у форматі XML, YAML, JSON, BSON або зберігати у формі звичайного тексту. Поля документів

доступні системі управління сховищем, що дозволяє застосувати запити та фільтри засновані на значеннях цих полів.

Як правило, документ містить всі дані для сутності. Елементи, які складають сутність, залежать від конкретного додатка. Наприклад, сутність може містити інформацію про клієнта, замовлення або їх поєднання. Один документ може включати інформацію, яка в реляційній базі даних зазвичай розподіляється по різних реляційних таблицях. Сховище документів не вимагає, щоб всі документи мали однакову структуру. Такий вільний підхід до форми надає велику гнучкість.

Наприклад, додатки можуть зберігати в документах різні дані згідно з поточними вимогами компанії.

Key	Document
1001	<pre>{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }</pre>
1002	<pre>{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }</pre>

Програма може отримувати документи по ключу документа.

Ключ – це унікальний ідентифікатор документу. Часто для нього використовується хешування для рівномірного розподілу даних. Деякі бази даних документів автоматично створюють ключ документа. Інші дозволяють вказати атрибут документа, який буде використовуватись як ключ.

Додаток також може запитувати документи на основі значень одного або декількох полів. Деякі бази даних документів підтримують індексацію для полегшення швидкості пошуку документів за одним або кількома індексованими полями.

Багато баз даних документів підтримують оновлення «на місці», тобто дозволяє програмі змінювати значення окремих полів без перезапису всього документа. Операція читання і запису в декількох полях одного документа зазвичай являються атомарними.

Стовпчасті сховища даних

Стовпчасті сховища даних, також відомі як сховища стовпців або сховища родини стовпців, організовують дані за допомогою стовпців і рядків. Стовпчасте сховище даних в його найпростішій формі майже не відрізняється від реляційної бази даних, принаймні організаційно. Основна перевага стовпчастого сховища даних полягає в здатності денормалізовано структурувати розріджені дані, що пов'язано зі стовпцево-орієнтованим методом зберігання даних.

Стовпчасте сховище даних можна уявити як набір табличних даних з рядками та стовпцями, де стовпці поділяються на певні групи або родини стовпців. Кожна родина стовпців включає набір логічно пов'язаних стовпців, які зазвичай витягуються або керуються як єдине ціле. Інші дані, які використовуються в інших процесах, зберігаються окремо в інших родині стовпців. В родину стовпців можна динамічно додавати нові стовпці, а рядки можуть бути розрідженими (тобто рядки не зобов'язані мати значення для кожного стовпця).

На наступній діаграмі представлено приклад таблиці з двома родині стовпців: *Identity* та *Contact Info*. Дані одної сутності мають однакові ключі рядків у всіх родині стовпців. Така структура, де рядки будь-якої сутності в родині стовпців можуть динамічно змінюватися, визначає важливу перевагу цієї категорії сховищ. Родини стовпців дуже добре підходять для зберігання даних з різними схемами.

CustomerID	Column Family: Identity
001	First name: Mu Bae Last name: Min
002	First name: Francisco Last name: Vila Nova Suffix: Jr.
003	First name: Lena Last name: Adamczyk Title: Dr.

CustomerID	Column Family: Contact Info
001	Phone number: 555-0100 Email: someone@example.com
002	Email: vilanova@contoso.com
003	Phone number: 555-0120

На відміну від сховища пар "ключ-значення" та баз даних документів, більшість стовпчастих баз даних упорядковують збережені дані за самими значеннями ключів, а не хеш-кодами від них. Ключ рядка розглядається як первинний індекс і забезпечує доступ на основі конкретного ключа або їх діапазону. Деякі реалізації дозволяють створювати вторинні індекси за певними стовбцями в родині стовпців. Вторинні індекси дозволяють отримувати дані за значеннями стовбців, а не ключами рядка.

Усі стовпці одної родини зберігаються на диску в одному файлі. Кожен файл містить певну кількість рядків. З використанням великих обсягів даних такий підхід дозволяє підвищити продуктивність за рахунок зменшення обсягу даних, які потрібно зчитувати з диска, коли відправляється запит на отримання кількох стовбців одночасно.

Читання та запис рядка з одної родини стовпців - це зазвичай атомарні операції. Однак деякі реалізації підтримують атомарність всього рядка, розподіленого по декількох родині стовпців.

Сховище пар "ключ-значення" по суті представляє собою велику хеш-таблицю. Кожне значення відповідає унікальному ключу, і сховище ключів використовує цей ключ для зберігання даних, застосовуючи до нього певну функцію хешування. Вибір функції хешування повинен забезпечити рівномірний розподіл хешованих ключів по сховищу даних.

Більшість сховищ пар "ключ-значення" підтримують лише найпростіші операції запиту, вставки та видалення. Щоб частково чи повністю змінити значення, додаток завжди перезаписує існуюче значення повністю. У більшості реалізацій атомарною

операцією вважається читання або запис одного значення. Запис великих значень займає відносно довгий час.

Додаток може зберігати в наборі значень довільні дані, але деякі сховища пар "ключ-значення" накладають обмеження на максимальний розмір значень. Програмне забезпечення сховища нічого не знає про значення, які в ньому зберігаються. Всі відомості про схему підтримуються і використовуються на рівні додатка. Ці значення по суті є великими двійковими об'єктами, які сховище витягує та зберігає за відповідним ключем.

Key	Value
AAAAA	1101001111010100110101111...
AABAB	1001100001011001101011110...
DFA766	0000000000101010110101010...
FABCC4	1110110110101010100101101...

Opaque to data store

Сховища пар "ключ-значення" спроектовані для додатків, які виконують прості операції пошуку на основі значення ключа або діапазону ключів, але вони не дуже підходять для систем, які повинні запитувати дані з кількох таблиць сховищ пар "ключ-значення", наприклад, даних, з'єднаних в декількох таблицях.

Крім того, сховища пар "ключ-значення" не зручні в сценаріях, де можуть виконуватися запити або фільтрація за значенням, а не лише за ключами. Наприклад, у реляційній базі даних ви можете знайти певний запис за допомогою умови WHERE та відфільтрувати його за неключовими стовбцями, але сховища пар "ключ-значення" зазвичай не підтримують таку можливість пошуку значень, або цей процес виконується повільно.

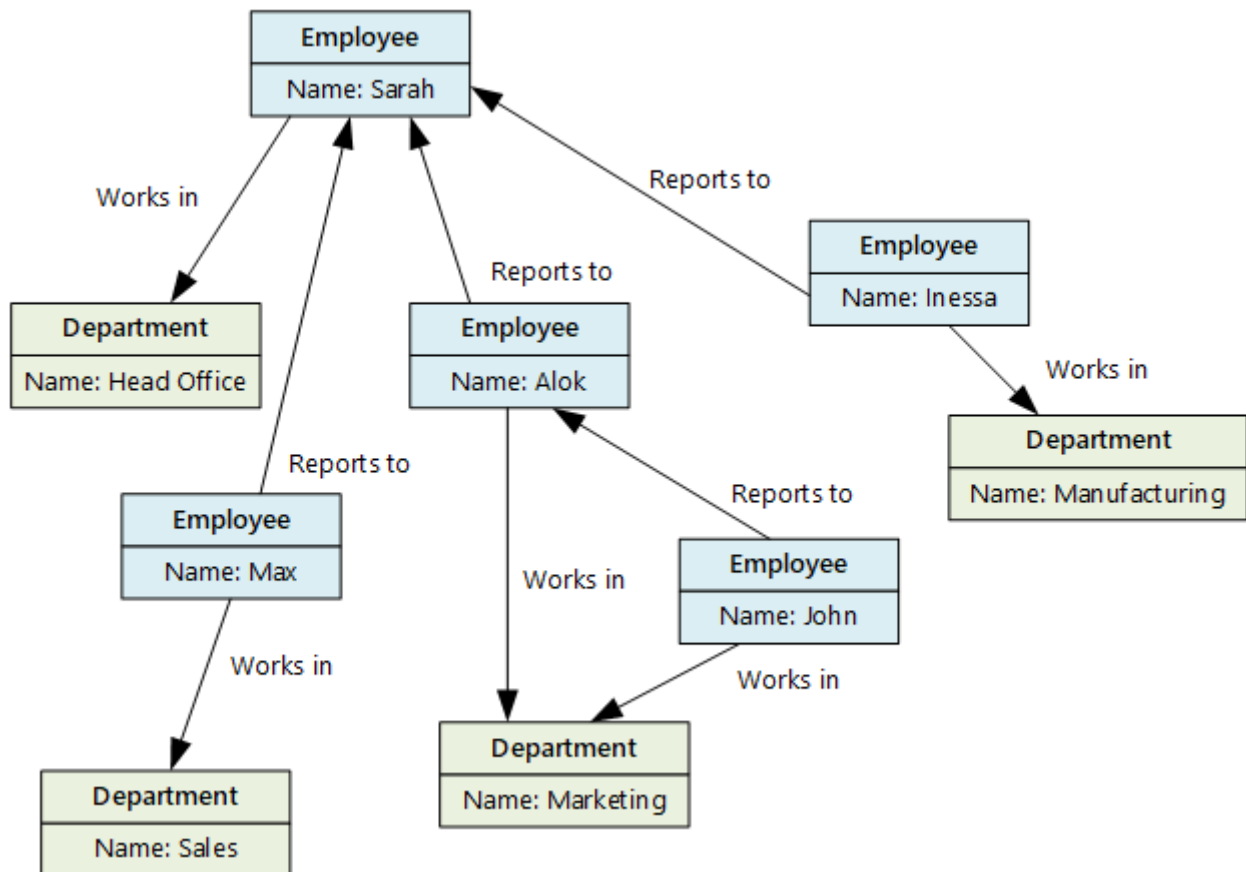
Одне сховище пар "ключ-значення" дуже легко масштабується, оскільки дозволяє зручно розподіляти дані серед кількох вузлів на різних комп'ютерах.

Сховища даних графів

Сховища даних графів керують інформацією двох типів: вузлами та ребрами. Вузли у цьому випадку представляють сутності, а ребра визначають зв'язки між ними. Вузли та ребра мають властивості, які надають інформацію про конкретний вузол чи ребро, приблизно як стовпці в реляційній таблиці. Ребра можуть мати напрямок, що вказує на характер зв'язку.

Сховища даних графів дозволяють додаткам ефективно виконувати запити, які проходять через мережу вузлів та ребер, а також аналізувати зв'язки між сутностями.

На схемі нижче представлені дані персоналу організації, структуровані у вигляді графа. Сутностями тут є співробітники та відділи, а ребра визначають відносини підпорядкування та відділ, в якому працює кожен співробітник. Стрілки на ребрах цього графа показують напрямок зв'язків.



Така структура дозволяє легко виконувати такі запити, як "знайти всіх співробітників, які безпосередньо чи опосередковано підпорядковуються Світлані" або "знайти всіх, хто працює в одному відділі з Дмитром". Процеси складного аналізу виконуються швидко навіть на великих графах з великою кількістю сутностей та зв'язків. Багато баз даних графів надають мову запитів, яку можна використовувати для ефективного обходу мережі зв'язків.

Сховища даних тимчасових рядів

Сховища даних тимчасових рядів - це набори значень, які впорядковані за часом. Відповідно, сховища даних часових рядів оптимізовані для зберігання даних саме такого типу. Ці сховища даних тимчасових рядів повинні підтримувати дуже велику кількість операцій запису, оскільки зазвичай вони в реальному часі збирають великий обсяг даних з великої кількості джерел. Ці сховища також добре підходять для зберігання даних телеметрії, наприклад, для збору даних від датчиків Інтернету речей або лічильників в застосунках чи системах. Оновлення в таких базах даних виконуються рідко, а видалення, як правило, є масовою операцією.

timestamp	deviceid	value
2017-01-05T08:00:00.123	1	90.0
2017-01-05T08:00:01.225	2	75.0
2017-01-05T08:01:01.525	2	78.0

Розмір окремих записів у базі даних тимчасових рядів, як правило, невеликий, але їх дуже багато, що означає, що загальний обсяг даних швидко збільшується. Сховища даних тимчасових рядів також обробляють дані, отримані в поза очікуваний час або несвоєчасно, автоматично індексують точки даних та оптимізують запити, отримані протягом певного проміжку часу. Ця остання можливість дозволяє швидко виконувати запити до мільйонів точок даних та кількох потоків даних, що, в свою чергу, забезпечує підтримку візуалізації тимчасових рядів (стандартний спосіб використання даних тимчасових рядів).

Сховище даних об'єктів

Сховища даних об'єктів оптимізовані для зберігання та отримання великих бінарних об'єктів, таких як зображення, текстові файли, відео- та аудіо потоки, об'єкти даних та документи великого розміру, образи дисків віртуальних машин. Об'єкт складається з збережених даних, метаданих та унікального ідентифікатора доступу до об'єкта. Сховища об'єктів підтримують окремі великі файли, а також дозволяють керувати всіма файлами завдяки значному об'єму сховища.

path	blob	metadata
/delays/2017/06/01/flights.csv	OXAABBCCDDEEF...	{created: 2017-06-02}
/delays/2017/06/02/flights.csv	OXAADDCCDDEEF...	{created: 2017-06-03}
/delays/2017/06/03/flights.csv	OXAEBBDEDDDEEF...	{created: 2017-06-03}

Деякі сховища даних об'єктів реплікують певний великий двійковий об'єкт між декількома вузлами кластера, що забезпечує швидке паралельне читання. Цей процес, в свою чергу, дозволяє реалізувати масштабовану архітектуру запиту даних, збережених у великих файлах, оскільки кілька процесів, які зазвичай виконуються на різних серверах, можуть одночасно запитувати великі файли даних.

Часто сховища даних об'єктів використовують як мережеві спільні теки. Доступ до файлів, які зберігаються в цих теках, можна отримати через комп'ютерну мережу за допомогою стандартних мережевих протоколів, таких як SMB. Якщо створені необхідні механізми підтримки безпеки та одночасного доступу, таке спільне використання даних дозволяє розподіленим службам з високим рівнем масштабованості надавати доступ до даних для базових низькорівневих операцій, тобто для простих запитів на читання та запис.

Сховища даних зовнішніх індексів

Сховища даних зовнішніх індексів дозволяють шукати інформацію, що міститься в інших сховищах даних та сервісах. Зовнішній індекс виступає у ролі вторинного індексу будь-якого сховища даних. Крім того, за його допомогою можна індексувати великі обсяги даних та надавати доступ до цих індексів практично в реальному часі.

Наприклад, у файловій системі можуть зберігатися текстові файли. За шляхом файл можна швидко знайти, але пошук на основі вмісту виконується повільно, оскільки скануються всі файли. Зовнішній індекс дозволяє створювати вторинні індекси, а потім швидко знаходити шлях до файлів, які відповідають заданим умовам. Розглянемо ще один приклад використання зовнішнього індексу. Припустимо, що сховища пар "ключ — значення" підтримують індексацію лише за ключем. Ви можете створити вторинний індекс на основі значень даних та швидко знайти ключ, який однозначно визначає кожен відповідний елемент.

id	search-document
233358	{"name": "Pacific Crest National Scenic Trail", "county": "San Diego", "elevation": 1294, "location": {"type": "Point", "coordinates": [-120.802102, 49.00021]}}
801970	{"name": "Lewis and Clark National Historic Trail", "county": "Richland", "elevation": 584, "location": {"type": "Point", "coordinates": [-104.8546903, 48.1264084]}}
1144102	{"name": "Intake Trail", "county": "Umatilla", "elevation": 1076, "location": {"type": "Point", "coordinates": [-118.0468873, 45.9981939]}}

Індекси створюються в процесі індексації, який може виконуватися за моделлю витягування, тобто за вимогою сховища даних, або за моделлю передачі, тобто за командою з коду додатка. У деяких системах підтримуються багатовимірні індекси та повнотекстовий пошук за великими обсягами текстових даних.

Часто сховища даних зовнішніх індексів використовуються для реалізації повнотекстового пошуку та пошуку в Інтернеті. У цих випадках підтримується точний чи неповний пошук. Неповний пошук знаходить документи, які відповідають набору умов, і обчислює для них коефіцієнт відповідності цьому набору. Деякі зовнішні індекси також підтримують лінгвістичний аналіз, який повертає відповідності з урахуванням синонімів, категорій (наприклад, при пошуку за запитом "собаки" вважається відповідність "вихованці") та морфології (наприклад, при пошуку за запитом "біг" вважається відповідність "біжить").

Стандартні вимоги

Часто архітектура нереляційних сховищ даних відрізняється від архітектури реляційних баз даних. Зокрема, ці сховища, як правило, не мають фіксованої схеми, а також не підтримують транзакції або обмежують їх область. З міркувань масштабованості вони, як правило, не включають вторинні індекси.

У таблиці нижче подано порівняння вимог для кожного нереляційного сховища даних.

Вимоги	Сховище даних документів	Стовбчасте сховище даних	Сховище даних пар «ключ-значення»	Дані графів

Нормалізація	Денормалізовані дані	Денормалізовані дані	Денормалізовані дані	Нормалізовані дані
Схема	Схема при читанні	Родини стовбців, визначення під час запису, схема стовбця при читанні	Схема при читанні	Схема при читанні
Узгодженість (між паралельними транзакціями)	Налаштований рівень узгодженості, гарантії на рівні документа	Гарантії на рівні родини стовбців	Гарантії на рівні ключів	Гарантії на рівні графа
Атомарність (область транзакції)	Колекція	Таблиця	Таблиця	Графік
Стратегія блокування	оптимістична (без блокування)	Песимістична (без блокування рядків)	Оптимістична (ETag)	-
Шаблон доступу	Прямий доступ	Статистичні вирази на основі даних великого формату	Прямий доступ	Прямий доступ
Індексація	Первинний та вторинний індекси	Первинний та вторинний індекси	Тільки первинний індекс	Первинний та вторинний індекси
Форма представлення даних	Документ	Таблиця з родинними стовбцями	Ключ і значення	Граф з ребрами та вершинами
Розріджений	Так	Так	Так	Ні
Масштабність (велика кількість стовбців атрибутів)	Так	Так	Ні	Ні
Розмір даних	Від малого (КБ) до середнього (декілька МБ)	Від середнього (МБ) до великого (декілька ГБ)	Невеликий (КБ)	Невеликий (КБ)
Загальний максимальний масштаб	Дуже великий (ПБ)	Дуже великий (ПБ)	Дуже великий (ПБ)	Великий (ТБ)

Вимоги	Дані тимчасових рядків	Сховище даних об'єктів	Сховище даних зовнішніх індексів
Нормалізація	Нормалізовані дані	Денормалізовані дані	Денормалізовані дані
Схема	Схема при читанні	Схема при читанні	Схема при записі
Узгодженість (мі паралельними транзакціями)	Не застосовується	Н/Д	Не застосовується
Атомарність (область транзакції)	Н/П	Об'єкт	Н/П
Стратегія блокування	Н/П	Песимістична (блокування великих двійкових об'єктів)	Н/П
Шаблон доступу	Прямий доступ та агрегування	Послідовний доступ	Прямий доступ
Індексація	Первинний та вторинний індекси	Тільки первинний індекс	Н/П
Форма представлення даних	Табличний	Великий двійковий об'єкт та метадані	Документ
Розріджений	Ні	Н/П	Ні
Масштабність (велика кількість стовбців атрибутів)	Ні	Так	Так
Розмір даних	Невеликий (КБ)	Від великого(ГБ) до дуже великого (ТБ)	Невеликий (ГБ)
Загальний максимальний масштаб	Великий (декілька ТБ)	Дуже великий (ПБ)	Великий (декілька ТБ)

Вибір між реляційними (SQL) та нереляційними (NoSQL) базами даних

	NOSQL (НЕРЕЛЯЦІЙНІ БАЗИ ДАНИХ)	SQL (РЕЛЯЦІЙНІ БАЗИ ДАНИХ)
ПРИЗНАЧЕННЯ:	Обробка великих, не пов'язаних між собою,	Обробка реляційних даних з логічним і

	невизначених або швидко змінюючих даних. Не залежні від схеми дані або схема, яка задається програмою. Програми для яких продуктивність і доступність важливіша, ніж сувора узгодженість. Неперервно працюючі програми, які обслуговують користувачів по всьому світу.	відокремленими вимогами, котрі можна визначити заздалегідь. Схема, яка потребує зберігати, підтримуючи синхронізацію між програмою і базою даних. Застарілі системи, призначенні для реляційних структур. Програми, які потребують використання складних запитів або багаторядкових транзакцій.
СЦЕНАРІЇ:	Мобільні додатки. Аналітика в реальному часі. Керування вмістом. Персоналізація . Додаток Інтернет речей. Перенесення баз даних.	Бухгалтерські, фінансові та банківські системи. Системи керування записами. Системи керування транзакціями.
МАСШТАБУВАННЯ:	Горизонтальне масштабування даних шляхом сегментування між серверами.	Вертикальне масштабування даних шляхом підвищення навантаження на сервери.
МОДЕЛЬ ДАНИХ:	Типи баз даних: ключ-значення, бази даних документів, стовбцеві і графові бази даних. Зберігання даних в залежності від типу баз даних.	Тип баз даних: таблиці рядків, згруповані по зв'язкам. Використання мови SQL. Дані зберігаються у вигляді рядків таблиці. Зв'язані дані зберігаються окремо і об'єднуються для складних запитів.

12.4 Порівняння SQL та NoSQL баз даних



SQL і NoSQL — дві популярні моделі баз даних, які використовуються для вирішення різних завдань. Щоб зрозуміти, яка з них підійде у вашому випадку, слід розібратися в їх відмінностях, перевагах та недоліках.

У цій статті я розгляну основні характеристики SQL- та NoSQL-баз даних і порівняю їх, щоб допомогти вибрати найкращий варіант для вашого проекту.

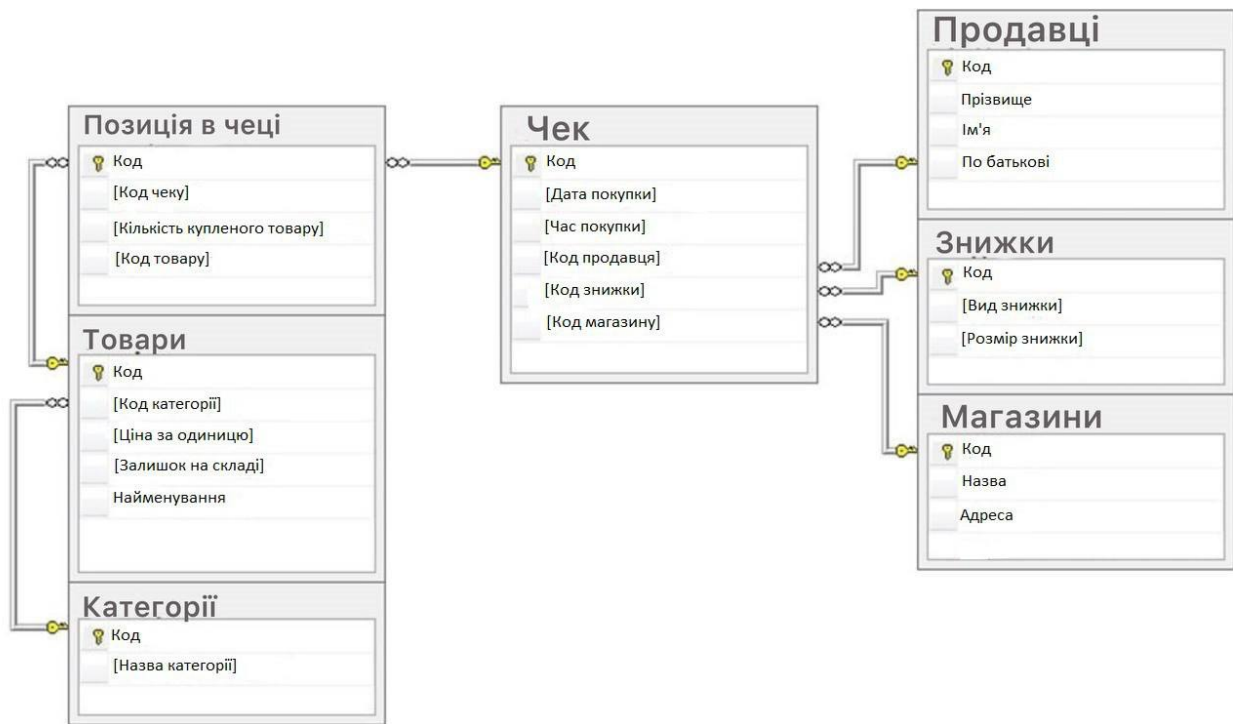
Почнемо з визначень SQL- та NoSQL-баз даних

SQL (Structured Query Language) — це мова структурованих запитань, яку використовують для управління та маніпулювання реляційними базами даних. SQL-бази даних використовуються там, де необхідно зберігати та керувати даними структурованої природи, наприклад, інформацією про продукти, покупців та оформлені замовлення в магазині.

NoSQL (Not Only SQL) — це широкий термін, який відноситься до нереляційних моделей баз даних, які використовують різні структури для зберігання даних: документи, ключ-значення, стовпцеві та графові бази даних. NoSQL-бази даних використовуються, коли необхідно зберігати дані неструктурованої природи, наприклад, великі обсяги текстових даних, зображення та відео.

Я люблю метафори, тому для кожного розглянутого типу баз даних приведу приклади:

SQL-бази даних можна порівняти з великим сховищем файлів, які розміщені в ящиках з мітками та зв'язками між ними. Кожна коробка містить окремий файл із визначеною інформацією. Щоб отримати доступ до потрібного файлу, слід знати мітку коробки, де він зберігається, і зв'язок з іншими ящиками, щоб скласти повну картину інформації. Це схоже на роботу з таблицями та зв'язками в SQL-базах даних, де кожна таблиця представляє собою окрему коробку із визначеною інформацією, яку можна легко знайти та об'єднати за допомогою спеціальних запитів.



NoSQL-бази даних можна порівняти із великим і складним пазлом, де кожна частина представляє окремий фрагмент інформації. Інформація може бути подана не лише у вигляді тексту, але й у вигляді зображень, звукових файлів, відеоматеріалів і т. д. Кожен фрагмент може мати різний розмір і форму, і для того, щоб отримати повну картину, необхідно зібрати і об'єднати всі фрагменти. Це схоже на роботу з NoSQL-базами даних, де кожен елемент може мати різний тип і формат, і для того, щоб отримати всю необхідну інформацію, слід використовувати різні методи та інструменти запитів. Як і в пазлі, кожен елемент даних має своє місце і збирається в єдину картину по мірі необхідності.

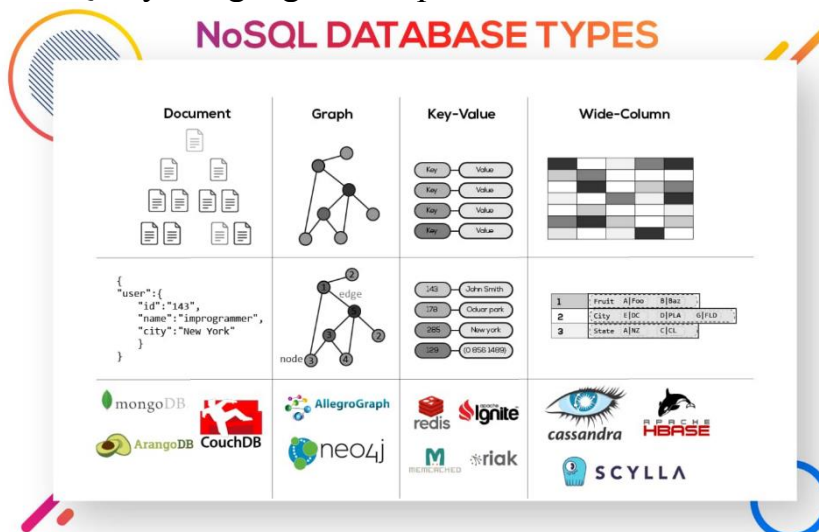
Типи та структури даних

У SQL та NoSQL два різних підходи до зберігання та організації інформації в базах даних. Кожен підхід має свій унікальний набір типів даних та структур, які використовуються для зберігання та обробки даних.

SQL-бази даних використовують реляційну модель даних, де інформація зберігається в таблицях, які мають певний набір стовпців, кожен з яких відповідає певному типу даних. Наприклад, цілі числа, рядки, дати, час, нульові значення, булеві значення тощо. Це означає, що набір даних може бути розділений на окремі поля для кожного виду інформації. У SQL-базах даних використовується мова запитів SQL, яка дозволяє користувачам створювати таблиці, додавати, змінювати і видаляти дані, а також отримувати інформацію з бази даних.

NoSQL-бази даних мають більш гнучку модель даних, яка не вимагає таблиць і зв'язків, як у SQL-базах. Зазвичай дані в NoSQL-базах зберігаються в документах, колекціях або графах. Документ — це структурований контейнер для зберігання даних у форматі пари ключ-значення, де пари можуть мати різні типи даних. Колекція — це група документів, які пов'язані між собою. Граф — це набір вершин і зв'язків між ними. NoSQL-бази даних використовують спеціальні мови запитів, які дозволяють

користувачам отримувати та опрацьовувати дані, такі як: MongoDB Query Language для MongoDB, Cassandra Query Language для Apache Cassandra тощо.

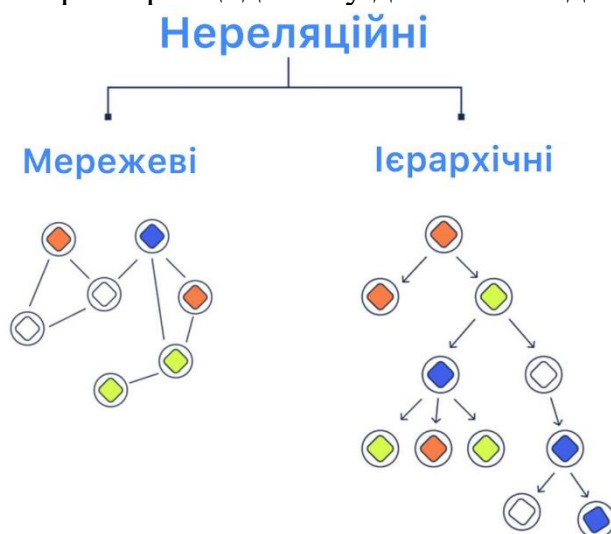


Способи зберігання даних та продуктивність

SQL- та NoSQL-бази даних використовують різні методи зберігання даних та мають унікальні характеристики продуктивності, які слід враховувати при виборі моделі бази даних для свого проекту.

SQL-бази даних зазвичай використовують реляційну модель, де дані зберігаються в таблицях, пов'язаних між собою. Дані в таблицях структуровані за строгими правилами і можуть бути пов'язані за допомогою зовнішніх ключів. Ця структура даних робить SQL-бази даних відмінною для зберігання та обробки структурованих даних. Однак ця модель не підходить для зберігання неструктурованих даних, таких як зображення, звукові файли, відеоматеріали і т. д.

NoSQL-бази даних використовують нереляційну модель зберігання даних, де дані можуть бути представлені у вигляді «ключ-значення», документів, графів тощо. В NoSQL-базах даних немає жорстких правил для організації даних, що робить їх більш гнучкими для зберігання різних типів даних та структур. Однак ця гнучкість може вплинути на продуктивність при обробці даних у деяких випадках.



Щодо продуктивності, SQL-бази даних зазвичай мають високу продуктивність при роботі з сильно структурованими даними та великими обсягами інформації,

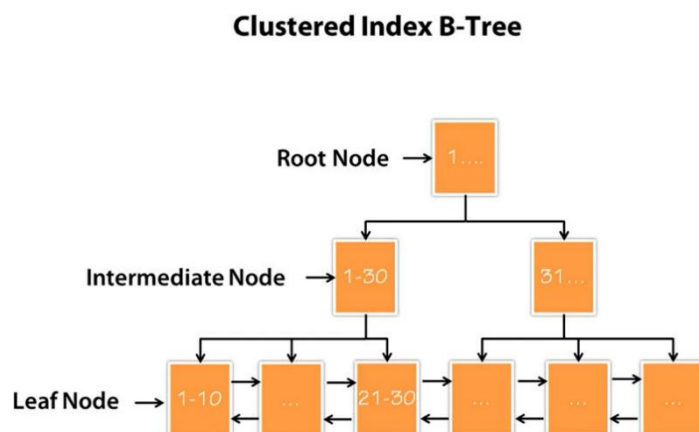
оскільки вони можуть використовувати оптимізовані індекси та швидкі алгоритми сортування/фільтрації даних. Однак при роботі з неструктурованими даними продуктивність SQL-баз може знижуватися.

NoSQL-бази даних зазвичай можуть обробляти великі обсяги неструктурованих даних з високою продуктивністю, оскільки вони не мають жорстких правил структуризації. Однак при роботі з складними запитаннями або при запиті пов'язаних даних можуть виникати труднощі з продуктивністю.

Методи індексації та відповідні можливості обробки запитів

Індексація є важливим способом підвищення продуктивності баз даних під час обробки запитів. В моделях SQL та NoSQL використовуються різні методи індексації даних, які забезпечують швидкий доступ до даних під час виконання запитів.

У SQL-базах даних використовується індекс B-Tree, який є структурою дерева, де кожен вузол має набір ключів, впорядкованих за зростанням або спаданням. Цей тип індекса дозволяє швидко виконувати запити SELECT, JOIN, WHERE та ORDER BY, а також забезпечує швидкий доступ до даних при виконанні операцій вставки, оновлення та видалення даних. Також в SQL-базах даних застосовуються індекси на кількох стовбцях, які дозволяють оптимізувати виконання запитів, що використовують кілька стовбців.



У NoSQL-базах даних використовуються різні типи індексів залежно від конкретної моделі. У MongoDB, яка відноситься до документо-орієнтованих баз даних, застосовується індексування на основі ключів, що дозволяє швидко здійснювати пошук за значеннями в полях документів. У базах даних "ключ-значення", таких як Redis, необхідно явно вказувати, які ключі повинні бути проіндексовані. Цей тип індексування дозволяє швидко отримувати значення за ключами, але не дозволяє виконувати складні запити, обмежуючись лише пошуком за ключами. У графових базах даних, таких як Neo4j, використовуються графові індекси, які забезпечують швидкий доступ до зв'язків між вузлами графа.

В цілому, SQL-бази даних мають більш розвинене індексування, ніж NoSQL-бази, що забезпечує вищу продуктивність при виконанні складних запитів, але може знижувати швидкість роботи при виконанні простих запитів. У NoSQL-базах даних індексування залежить від конкретної моделі і вимагає більш детального планування та налаштувань для забезпечення потрібної продуктивності.

Надійність та стійкість баз даних

Надійність та стійкість баз даних – один із найважливіших аспектів під час їх розробки та експлуатації. Базы даних можуть зіткнутися з різними проблемами, такими як збої та відмови, тому необхідно мати стійку та надійну систему, яка дозволить швидко відновити роботу у випадку виникнення неполадок.

SQL- та NoSQL-базы даних використовують різні механізми для забезпечення надійності та стійкості, але їхні методи залежать від конкретного рішення та його вимог до надійності та безпеки.

У SQL-базах даних використовується транзакційна модель, яка дозволяє зберігати цілісність даних та забезпечує ACID (Atomicity, Consistency, Isolation, Durability) властивості для стійкості та надійності бази даних. Також SQL-базы даних можуть використовувати процеси резервного копіювання та відновлення для забезпечення збереження даних та мінімізації втрат у випадку збою.

У NoSQL-базах даних, які рідко використовують транзакційні моделі, зазвичай застосовується розподілена архітектура для підвищення надійності та стійкості. Кластеризація та реплікація даних допомагають мінімізувати ймовірність втрати даних та забезпечують їх доступність у випадку відмови вузла чи вузлів.

Порівнюючи ці два типи баз даних, у SQL-баз є кілька переваг. Наприклад, в SQL легко встановити обмеження на доступ до даних для різних користувачів, а також застосовувати різні аутентифікаційні механізми для забезпечення безпеки даних. Крім того, SQL-базы даних мають кращу підтримку транзакційності, що дозволяє автоматично відкатувати зміни при виявленні проблемних транзакцій і, таким чином, усуває можливі проблеми безпеки.

З іншого боку, у NoSQL-баз даних є кілька переваг в сфері безпеки. Наприклад, MongoDB та Couchbase використовують документно-орієнтовану модель, що робить їх більш стійкими до атак за допомогою SQL-ін'єкцій. У NoSQL-базах даних також застосовуються методи шифрування даних, що забезпечує більш високий рівень безпеки. Однак у NoSQL-базах даних розробники повинні самостійно реалізовувати механізми безпеки, що може призводити до можливих проблем, якщо розробник недостатньо досвідчений або уважний під час розробки.

Гнучкість та масштабованість SQL і NoSQL

SQL- та NoSQL-базы даних мають різні рівні гнучкості та масштабованості, які залежать від конкретного рішення та його вимог до масштабованості та гнучкості.

Однією з переваг NoSQL-баз даних є гнучкість та масштабованість. Завдяки своїй децентралізованій архітектурі, вони можуть легко масштабуватися горизонтально, додаючи нові сервери та розподіляючи навантаження між ними. Також в NoSQL-базах даних немає жорстких вимог до структури даних, що дозволяє швидко змінювати схеми даних та додавати нові поля без переходу на нову версію схеми. Однак це може призвести до труднощів з обробкою запитів, якщо структура даних не дуже добре обдумана і змінюється занадто часто.

SQL-базы даних, з іншого боку, можуть бути більш обмеженими у гнучкості структури даних та масштабованості. У них необхідно заздалегідь визначити структуру таблиць та типи даних, що може ускладнити процес зміни схеми у майбутньому. Однак кешування в SQL-базах даних дозволяє покращити продуктивність та точніше відстежувати зміни, що може допомогти в оптимізації обробки запитів.

Отже, NoSQL-бази даних є більш гнучкими та масштабованими, що особливо важливо для великих проектів з великим обсягом даних, але при цьому менш передбачуваними та менш організованими, що може призвести до труднощів із управлінням. SQL-бази даних мають більшу структурованість та краще відповідають принципам ACID, але при цьому вони більш обмежені у гнучкості структури даних та масштабованості, що може бути проблемою для великих проектів.

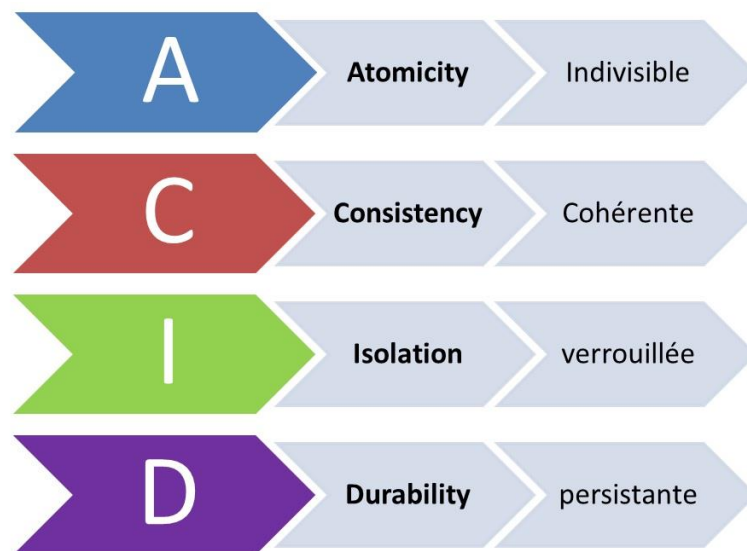
Вплив типу бази даних на виконання транзакцій

Транзакції - це операції, які виконуються з даними в базі даних і підтримують властивості ACID (*Atomicity, Consistency, Isolation, Durability*).

ACID - це скорочення, що позначає чотири основні властивості транзакцій в СУБД:

- **Атомарність (Atomicity)** - це властивість, яка гарантує, що транзакція буде виконана цілком або взагалі не буде виконана. Якщо транзакцію не можна виконати повністю, вона буде відкочена, і база даних повернеться до стану, який був до початку транзакції.
- **Співгласність (Consistency)** - це властивість, яка означає, що транзакція повинна привести базу даних з одного згідного стану в інший згідний стан. В результаті виконання транзакції база даних не може бути порушена, і дані повинні відповідати всім обмеженням і правилам цілісності.
- **Ізоляція (Isolation)** - це властивість, яка допомагає уникнути непередбачуваних результатів при одночасному доступі до даних кількох транзакцій. Властивість ізоляції гарантує, що кожна транзакція виконується незалежно від інших, навіть якщо вони працюють з тими ж даними.
- **Довговічність (Durability)** - це властивість, яка гарантує, що після виконання транзакції дані будуть збережені в базі даних і не будуть втрачені або пошкоджені у випадку можливого збою системи. Транзакція повинна бути виконана так, щоб зміни, внесені в базу даних, були збережені навіть в разі відмови системи і необхідності її перезапуску.

Ці властивості необхідні для забезпечення цілісності даних та відсутності суперечливостей не лише при звичайній роботі з базою даних, але і при збоях системи.



Забезпечення транзакцій - один із ключових аспектів баз даних, які гарантують цілісність даних та правильну роботу додатків, які використовують ці дані.

SQL- та NoSQL-бази даних мають різні моделі підтримки транзакцій та власні методи транзакційної обробки.

SQL-бази даних зазвичай використовують набір властивостей ACID, що означає, що транзакції виконуються спільно, і будь-які відкати транзакцій стануться для всіх з'єднань з даними. Це означає, що якщо транзакція завершиться невдачею, всі зміни будуть відмінені, і база даних повернеться до попереднього стану. У SQL-базах даних транзакції можуть бути підтримані за допомогою механізмів спеціальних мовних конструкцій (наприклад, в SQL є оператори BEGIN TRANSACTION, COMMIT та ROLLBACK).

З іншого боку, NoSQL-бази даних, такі як MongoDB чи Cassandra, зазвичай не використовують транзакції, оскільки вони спрямовані на великі масштаби даних та швидкість обробки, а не на підтримку транзакцій. Замість цього NoSQL-бази даних використовують CAP-теорему, яка допомагає вибрати між узгодженістю даних, доступністю та стійкістю до розділення (partition tolerance). Це означає, що вони зазвичай надають споживачеві вибір між узгодженістю та доступністю, і що наші додатки можуть вибрати те, що їм більше потрібно в конкретній ситуації.

Отже, SQL- та NoSQL-бази даних мають різні методи реалізації транзакцій. SQL-бази даних побудовані на моделі підтримки транзакцій і включають в себе ACID-властивості, тоді як в NoSQL-базах даних підтримка транзакцій не завжди реалізується. Замість цього NoSQL-бази даних використовують CAP-теорему, щоб допомогти користувачам вибрати між узгодженістю та доступністю даних.

Налаштування та управління SQL- та NoSQL-базами даних

Для налаштування та управління SQL-базами даних найбільш поширеними інструментами є SQL Management Studio (для SQL Server), phpMyAdmin (для MySQL) та pgAdmin (для PostgreSQL). Вони дозволяють адміністраторам керувати базою даних,

створювати та змінювати таблиці, індекси, представлення, процедури та тригери, а також встановлювати права доступу та конфігурувати різні параметри.

Для налаштування та управління NoSQL-базами даних найбільш поширеними інструментами є MongoDB Compass, Cassandra Query Language Shell (CQLSH) та Neo4j Browser. Ці інструменти дозволяють адміністраторам керувати базою даних, створювати та змінювати колекції (у випадку MongoDB) чи таблиці (у випадку Cassandra та Neo4j), виконувати запити та керувати різними параметрами конфігурації.

Основні відмінності в налаштуванні та управлінні SQL- та NoSQL-базами даних пов'язані з їх різноманітною структурою. SQL-бази даних часто вимагають більш жорстких схем для даних, що може обмежувати гнучкість деяких операцій, наприклад, додавання нових стовпців до таблиці. NoSQL-бази даних, навпаки, зазвичай надають більш гнучкі схеми, дозволяючи швидко створювати та змінювати колекції чи таблиці, що робить їх більш масштабованими та зручними для деяких видів додатків.

Відмінності

Основні відмінності між SQL та NoSQL:

1. **Схема даних:** SQL-база даних має жорстку схему даних, яка визначає типи даних та зв'язки між таблицями. У NoSQL-базах даних немає жорсткої схеми даних.
2. **Масштабованість:** SQL-бази даних мають обмеження на масштабованість, через що вони можуть бути неефективними в обробці великої кількості даних. NoSQL-бази даних мають велику масштабованість і можуть обробляти великі обсяги даних.
3. **Гнучкість запитів:** SQL має дуже потужну мову запитів, що робить його кращим вибором для складних запитів, пов'язаних з великою кількістю таблиць. З іншого боку, NoSQL має просту мову запитів, яка добре підходить для запитів, пов'язаних з великою кількістю даних.
4. **Швидкість обробки:** Незважаючи на те, що SQL зазвичай працює повільніше, ніж NoSQL, його потужна мова запитів дозволяє швидко обробляти складні запити. З іншого боку, NoSQL працює дуже швидко з неструктурованими даними великими обсягами.

Особливості SQL:

- SQL-бази даних мають жорстко визначений формат зберігання даних, що робить їх найбільш відповідним вибором для представлення складних пов'язаних даних.
- SQL має потужну мову запитів, що дозволяє обробляти складні запити.
- SQL-бази даних зазвичай вимагають значних витрат на обслуговування та налаштування.

Висновок

Узагальнюючи, SQL- та NoSQL-бази даних мають унікальні переваги та недоліки в залежності від типу даних, які ми намагаємося зберігати та обробляти. SQL-бази даних зазвичай підходять для зберігання структурованих даних, тоді як NoSQL краще підходить для неструктурованих даних. Але, в кінцевому підсумку, вибір бази даних залежить від конкретних вимог проекту.

12.5 Що таке MongoDB

MongoDB представляє найпопулярнішу на даний момент документо-орієнтовану систему управління базами даних. За різними оцінками входить до десятки найбільш використовуваних баз даних у світі.

На момент написання цього матеріалу останньою версією платформи була версія 7.0, яка вийшла влітку 2023 року і яку використовуватимуть далі в цьому посібнику. Використання конкретної версії може дещо відрізнятися від застосування інших версій платформи MongoDB. Але загалом відмінності в побудові запитів у MongoDB між версіями мінімальні, має місце доволі велика сумісність попередніх версій із наступними.

Документи замість рядків

Якщо реляційні бази даних зберігають рядки, то MongoDB зберігає документи. На відміну від рядків документи можуть зберігати складну за структурою інформацію. Документ можна уявити як сховище ключів і значень.

Ключ являє собою просту мітку, з якою асоційовано певний шматок даних.

Однак за всіх відмінностей є одна особливість, яка зближує MongoDB і реляційні бази даних. У реляційних СУБД зустрічається таке поняття як первинний ключ. Це поняття описує якийсь стовпець, який має унікальні значення. У MongoDB для кожного документа є унікальний ідентифікатор, який називається `_id`. І якщо явно не вказати його значення, то MongoDB автоматично згенерує для нього значення.

Кожному ключу зіставляється певне значення. Але тут також треба враховувати одну особливість: якщо в реляційних базах є чітко окреслена структура, де є поля, і якщо якесь поле не має значення, йому (залежно від налаштувань конкретної бд) можна присвоїти значення NULL. У MongoDB все інакше. Якщо якомусь ключу не зіставлено значення, то цей ключ просто опускається в документі і не вживається.

Колекції

Якщо в традиційному світі SQL є таблиці, то у світі MongoDB є колекції. І якщо в реляційних БД таблиці зберігають однотипні жорстко структуровані об'єкти, то в колекції можуть містити найрізноманітніші об'єкти, що мають різну структуру і різний набір властивостей.

Реплікація

Уся система MongoDB може представляти не тільки одну базу даних, яка розташовується на одному фізичному сервері. Функціональність MongoDB дає

змогу розташувати кілька баз даних на кількох фізичних серверах, і ці бази даних зможуть легко обмінюватися даними і зберігати цілісність.

Система зберігання даних у MongoDB являє собою набір реплік. У цьому наборі є основний вузол, а також може бути набір вторинних вузлів. Усі вторинні вузли зберігають цілісність і автоматично оновлюються разом з оновленням головного вузла. І якщо основний вузол з якихось причин виходить з ладу, то один із вторинних вузлів стає головним.

Формат даних у MongoDB

Одним із популярних стандартів обміну даними та їхнього зберігання є JSON (JavaScript Object Notation). JSON ефективно описує складні за структурою дані. Спосіб зберігання даних у MongoDB у цьому плані схожий на JSON, хоча формально JSON не використовується. Для зберігання в MongoDB застосовується формат, який називається BSON (BiCon) або скорочення від binary JSON.

BSON дає змогу працювати з даними швидше: швидше виконується пошук і обробка. Хоча треба зазначити, що BSON на відміну від зберігання даних у форматі JSON має невеликий недолік: загалом дані в JSON-форматі займають менше місця, ніж у форматі BSON, з іншого боку, цей недолік з лишком окупається швидкістю.

Кросплатформеність

MongoDB написана на C++, тому її легко портувати на найрізноманітніші платформи. MongoDB може бути розгорнута на платформах Windows, Linux, MacOS, Solaris. Можна також завантажити вихідний код і самому скомпілювати MongoDB, але рекомендується використовувати бібліотеки з оф.сайту.

Простота у використанні

Відсутність жорсткої схеми бази даних і у зв'язку з цим потреби при найменшій зміні концепції зберігання даних перестворювати цю схему значно полегшують роботу з базами даних MongoDB і подальшим їх масштабуванням. Крім того, економиться час розробників. Їм більше не треба думати про перестворення бази даних і витрачати час на побудову складних запитів.

Але, навіть з огляду на всі недоліки традиційних баз даних і переваги MongoDB, важливо розуміти, що завдання бувають різні, і методи їх вирішення бувають різні. У якійсь ситуації MongoDB дійсно поліпшить продуктивність вашого застосунку, наприклад, якщо треба зберігати складні за структурою дані. В іншій же ситуації краще буде використовувати традиційні реляційні бази даних. Крім того, можна використовувати змішаний підхід: зберігати один тип даних у MongoDB, а інший тип даних - у традиційних БД.

GridFS

Однією з проблем під час роботи з будь-якими системами баз даних є збереження даних великого розміру. Можна зберігати дані у файлах, використовуючи різні мови програмування. Деякі СУБД пропонують спеціальні типи даних для зберігання бінарних даних у БД (наприклад, BLOB у MySQL).

На відміну від реляційних СУБД MongoDB дає змогу зберігати різні документи з різним набором даних, однак при цьому розмір документа обмежується 16 мб. Але MongoDB пропонує рішення - спеціальну технологію GridFS, яка дає змогу зберігати дані за розміром більшим, ніж 16 мб.

Система GridFS складається з двох колекцій. У першій колекції, яка називається `files`, зберігаються імена файлів, а також їхні метадані, наприклад, розмір. А в іншій колекції, яка називається `chunks`, у вигляді невеликих сегментів зберігаються дані файлів, зазвичай сегментами по 256 кб.

Для тестування GridFS можна використовувати спеціальну утиліту `mongofiles`, яка йде в пакеті `mongodb`.

Встановлення та початок роботи з MongoDB на Windows

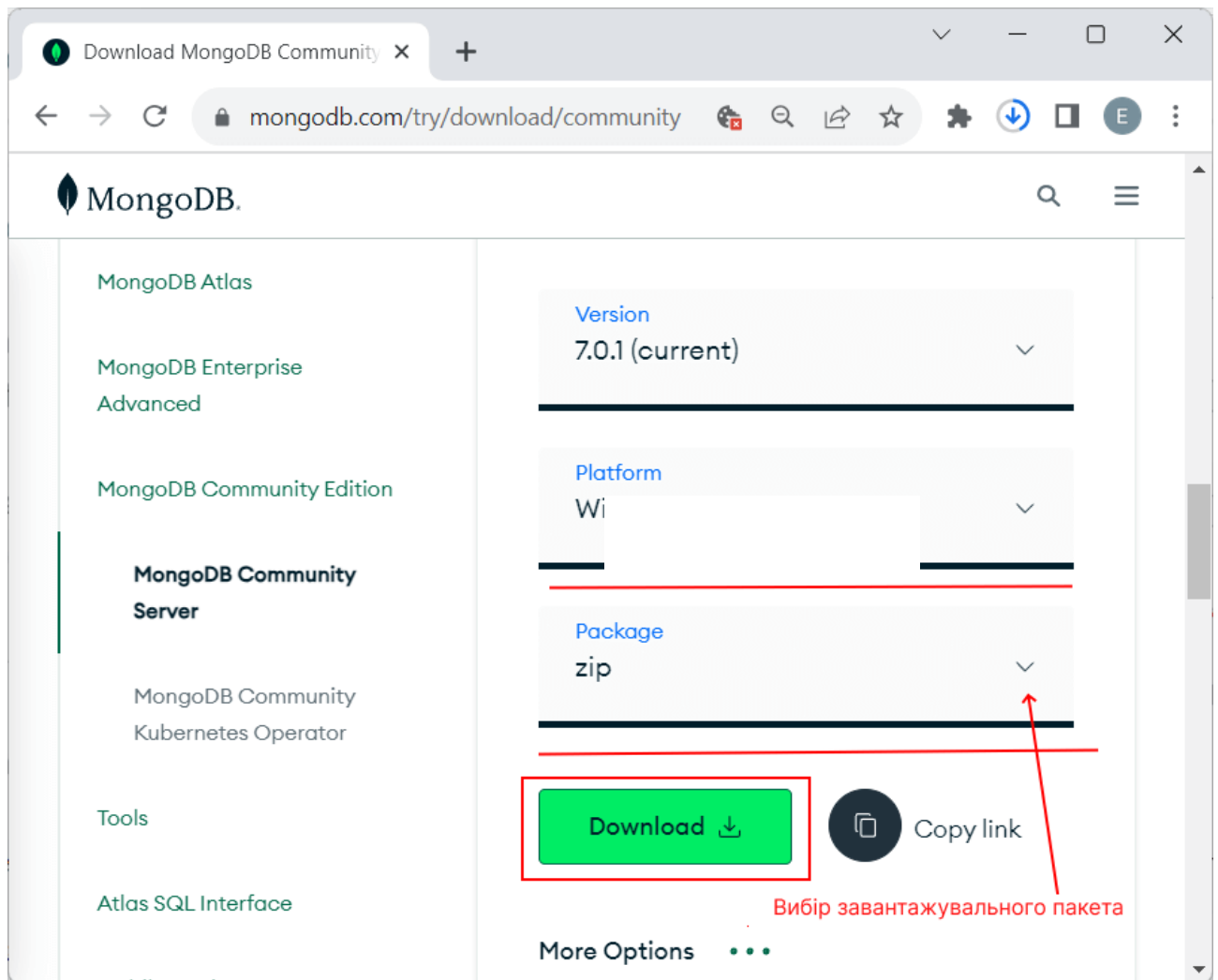
Встановлення сервера Mongod

Офіційний сайт надає пакети дистрибутивів для різних платформ: Windows, Linux, MacOS, Solaris. І для кожної платформи доступно кілька дистрибутивів. Причому є два види серверів - безкоштовний Community і платний Enterprise. У цьому посібнику будемо використовувати безкоштовну версію Community.

Для встановлення MongoDB завантажимо один з розповсюджуваних пакетів з офіційного сайту <https://www.mongodb.com/try/download/community>.

Для завантаження всіх необхідних файлів виберемо потрібну операційну систему і відповідний тип пакета. Розглянемо на прикладі встановлення на ОС Windows.

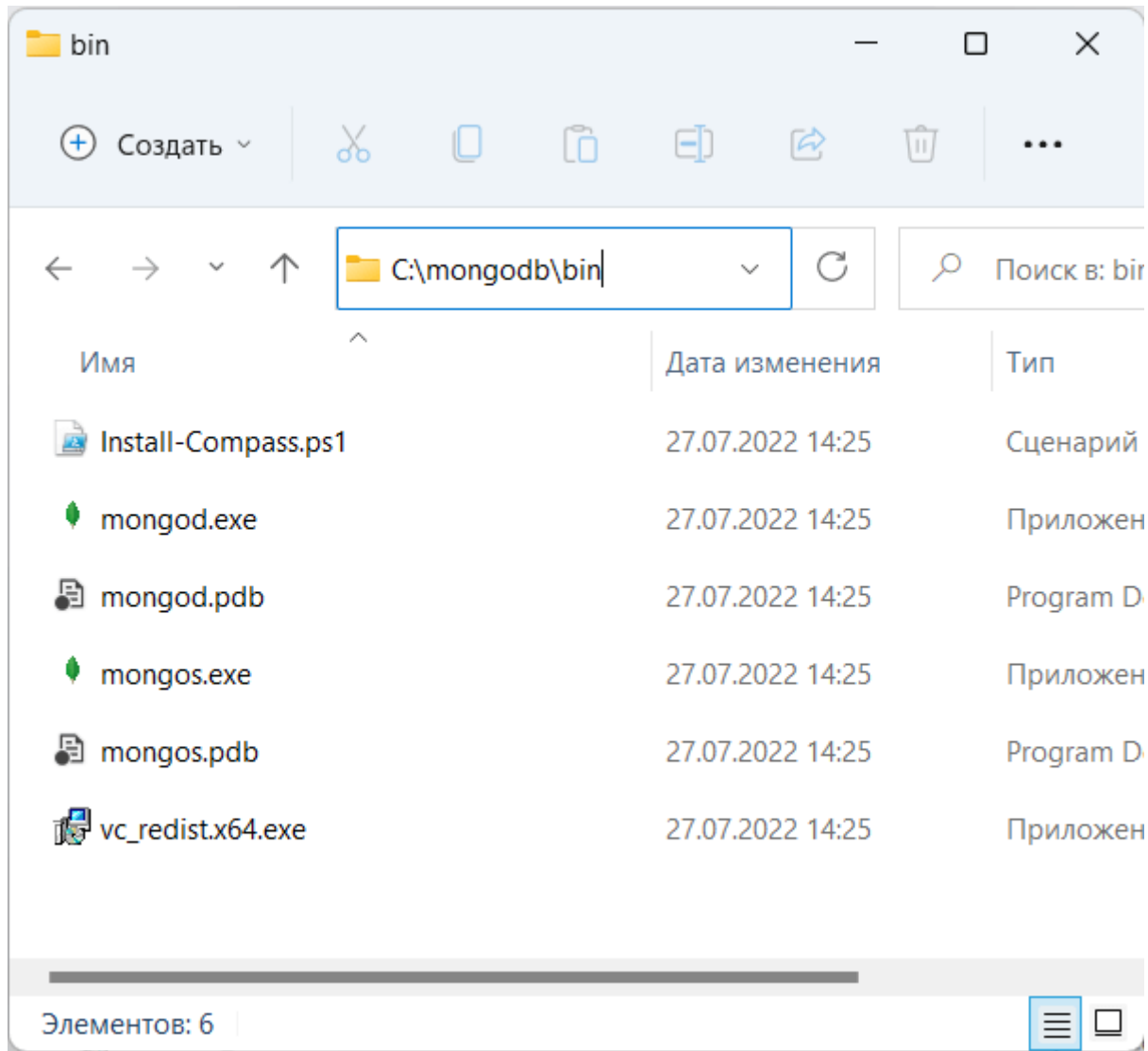
MongoDB можна завантажити в низці варіантів. Так, для Windows доступне завантаження установника `msi` і також доступне завантаження архіву `zip`. У реальності нам достатньо завантажити `zip`-архів і розпакувати в потрібній нам папці. Тому виберемо цей варіант завантаження (хоча також можна вибрати варіант установника `msi`):



Якщо до встановлення вже було встановлено більш ранню версію MongoDB, то її необхідно видалити.

Після завантаження архівного пакета розпакуємо його в папку C:\mongodb.

Якщо після встановлення ми відкриємо папку bin у розпакованому архіві (C:\mongodb\bin), то зможемо знайти там купу додатків, які виконують певну роль. Коротенько розглянемо їх.



mongod: сервер баз даних MongoDB. Він обробляє запити, керує форматом даних і виконує різні операції у фоновому режимі з управління базами даних

mongos: служба маршрутизації MongoDB, яка допомагає обробляти запити і визначати місце розташування даних у кластері MongoDB

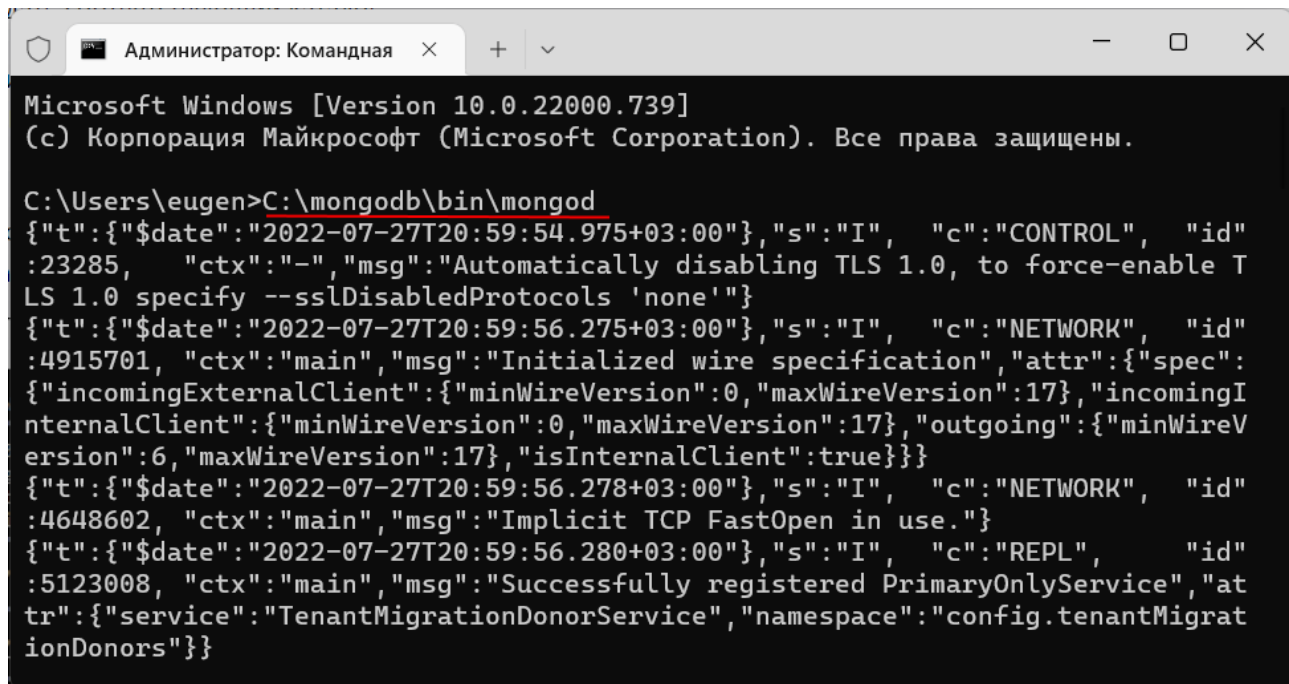
Створення каталогу для БД і запуск MongoDB

Після інсталяції треба створити на жорсткому диску каталог, у якому розміщуватимуться бази даних MongoDB.

В ОС Windows за замовчуванням MongoDB зберігає бази даних за шляхом `C:\data\db`, тому, якщо ви використовуєте Windows, вам треба створити відповідний каталог.

Якщо ж виникла потреба використовувати якийсь інший шлях до файлів, то його можна передати під час запуску MongoDB у прапорі `--dbpath`.

Отже, після створення каталогу для зберігання БД можна запустити сервер MongoDB. Сервер представляє додаток `mongod`, який знаходиться в каталозі `bin` у папці сервера. Для цього запустимо термінал/командний рядок і там введемо відповідні команди. Для ОС Windows це матиме такий вигляд:



```
Администратор: Командная
Microsoft Windows [Version 10.0.22000.739]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\eugen>C:\mongodb\bin\mongod
{"t":{"$date":"2022-07-27T20:59:54.975+03:00"},"s":"I", "c":"CONTROL", "id":
:23285, "ctx":"-", "msg":"Automatically disabling TLS 1.0, to force-enable T
LS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2022-07-27T20:59:56.275+03:00"},"s":"I", "c":"NETWORK", "id"
:4915701, "ctx":"main", "msg":"Initialized wire specification", "attr":{"spec":
{"incomingExternalClient":{"minWireVersion":0, "maxWireVersion":17}, "incomingI
nternalClient":{"minWireVersion":0, "maxWireVersion":17}, "outgoing":{"minWireV
ersion":6, "maxWireVersion":17}, "isInternalClient":true}}}
{"t":{"$date":"2022-07-27T20:59:56.278+03:00"},"s":"I", "c":"NETWORK", "id"
:4648602, "ctx":"main", "msg":"Implicit TCP FastOpen in use."}
{"t":{"$date":"2022-07-27T20:59:56.280+03:00"},"s":"I", "c":"REPL", "id"
:5123008, "ctx":"main", "msg":"Successfully registered PrimaryOnlyService", "at
tr":{"service":"TenantMigrationDonorService", "namespace":"config.tenantMigrat
ionDonors"}}
```

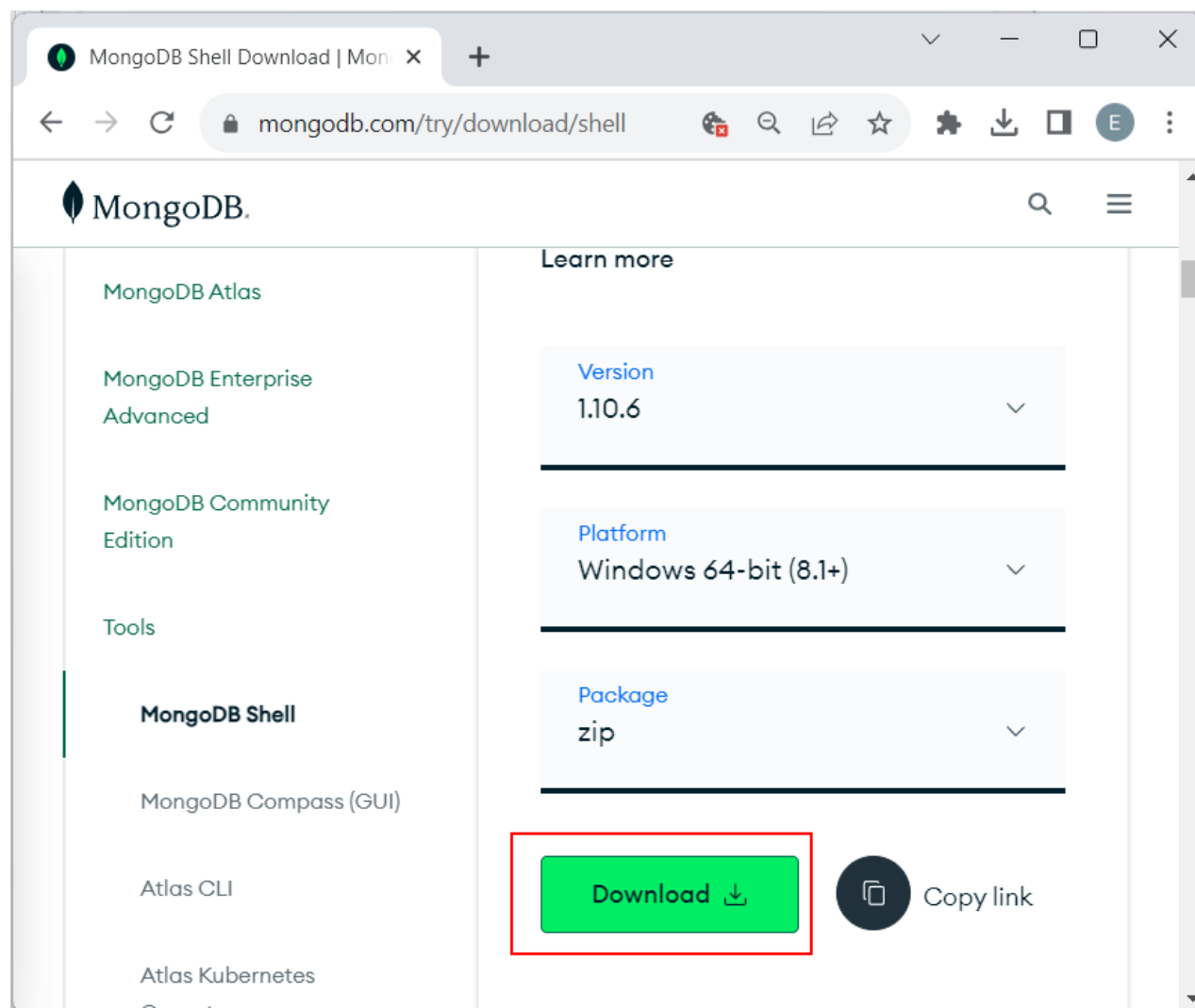
Командний рядок відобразить нам низку службової інформації, наприклад, що сервер запускається на `localhost` на порту `27017`.

І після вдалого запуску сервера ми зможемо проводити операції з бд через клієнт.

Встановлення клієнта Mongosh

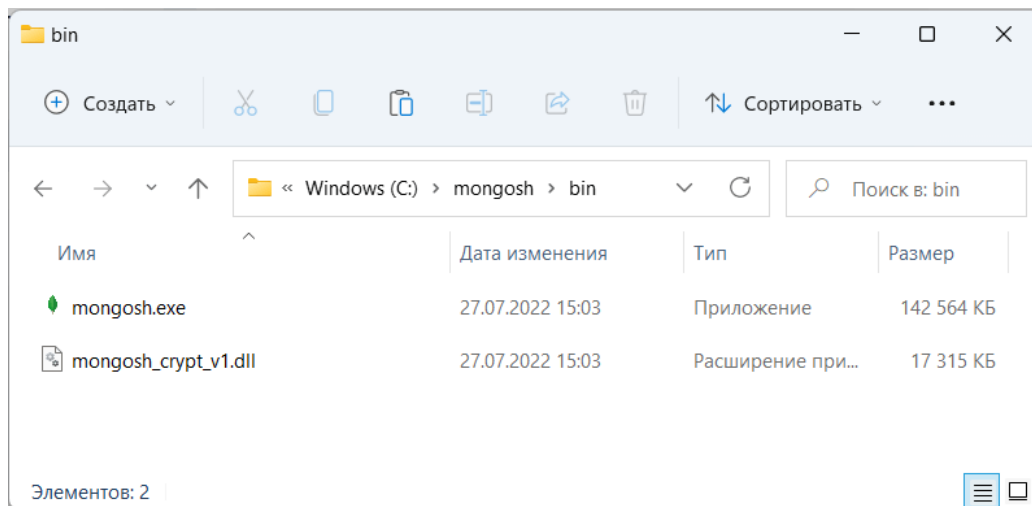
Вище ми встановили сервер MongoDB. Однак для роботи з сервером нам потрібен клієнт. Найпростішим клієнтом у даному випадку є MongoDB Shell або `mongosh` - консольна оболонка для надсилання запитів до сервера, яка також надається безпосередньо компанією MongoDB.

Для встановлення клієнта перейдемо за адресою <https://www.mongodb.com/try/download/shell>



Тут знову ж таки ми можемо вибрати версії клієнта для різних операційних систем. Для Windows пакет клієнта доступний у вигляді архіву zip. Завантажимо цей пакет і розпакуємо його в папку C:\mongosh.

Якщо в розпакованому архіві ми зайдемо в папку bin (тобто C:\mongosh\bin), то знайдемо там консольну утиліту mongosh, яка застосовуватиметься для роботи із сервером MongoDB:



Підключення до сервера з клієнта

Використовуємо вище встановлений клієнтський додаток `mongosh` для взаємодії із сервером `mongodb`. (Під час роботи з `mongosh` не варто забувати, що у нас має бути запущений сервер `mongod`). Отже, запустимо файл `mongosh`, який розташовується у вище розглянутій папці встановлення:

```
mongosh mongodb://127.0.0.1:27017
Please enter a MongoDB connection string (Default: mongodb://localhost/):
Current Mongosh Log ID: 64f96c2c4e02b69eb1a4479a
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=
2000&appName=mongosh+1.10.6
Using MongoDB:      7.0.1
Using Mongosh:      1.10.6

For mongosh info see: https://docs.mongodb.com/mongosh-shell/

The server generated these startup warnings when booting
2023-09-07T09:09:35.472+03:00: Access control is not enabled for the database. Read and write a
ccess to data and configuration is unrestricted
2023-09-07T09:09:35.473+03:00: This server is bound to localhost. Remote systems will be unable
to connect to this server. Start the server with --bind_ip <address> to specify which IP addresse
s it should serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavio
r is desired, start the server with --bind_ip 127.0.0.1 to disable this warning

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test>
```

Під час запуску програми `mongosh` спочатку вона запитає користувача, який рядок підключення використовувати для підключення до сервера `MongoDB`. У цьому моменті просто натиснемо на `Enter`, щоб використовувати рядок підключення до `MongoDB` за замовчуванням. А за замовчуванням сервер `mongodb` запускається на порту `27017`, а повний рядок підключення має такий вигляд:
`mongodb://localhost:27017` або `mongodb://127.0.0.1:27017`

Після підключення консоль відобразить низку службової інформації та підключиться до бази даних test.

Тепер зробимо якісь найпростіші дії. Введемо в консоль послідовно такі команди і після кожної команди натиснемо на Enter:

```
db.users.insertOne( { name: "Tom" } )
db.users.find()
```

За допомогою функції `db.users.insertOne()` у колекцію `users` бази даних `test` додається об'єкт `{ name: "Tom" }`. Ідентифікатор `db` представляє поточну базу даних. У нашому випадку ми підключені до бази даних за замовчуванням - тобто до бази даних `test`, відповідно `db` тут представляє базу даних `test`. При цьому не важливо, є чи немає така бд - якщо її немає, то вона створюється.

Після `db` йде `users` - це колекція, в яку потім ми додаємо новий об'єкт. Якщо в SQL нам треба створювати таблиці заздалегідь, то колекції MongoDB створює самостійно за їх відсутності.

Опис об'єкта, що додається, визначається у форматі, з яким ви, можливо, знайомі, якщо мали справу з форматом JSON. Тобто в цьому разі в об'єкта визначено один ключ `"name"`, якому зіставляється значення `"Tom"`. Тобто ми додаємо користувача з ім'ям Tom.

Якщо об'єкт було успішно додано, то консоль виведе результат операції, зокрема, ідентифікатор доданого об'єкта.

А друга команда `db.users.find()` виводить на екран усі об'єкти з бд `test`.



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

-----
The server generated these startup warnings when booting
test>

test> db.users.insertOne({name: "Tom"})
{
  acknowledged: true,
  insertedId: ObjectId("62e18207487bdaed0b570287")
}
test> db.users.find()
[ { _id: ObjectId("62e18207487bdaed0b570287"), name: 'Tom' } ]
test> _
```

З виведення ви можете побачити, що до початкових значень об'єкта було додано якесь незрозуміле поле `ObjectId`. Як ви пам'ятаєте, MongoDB як унікальні ідентифікатори документа використовує поле `_id`. І в даному випадку `ObjectId` якраз і представляє значення для ідентифікатора `_id`.

Встановлення драйверів MongoDB

Надалі в рамках цього посібника ми розглядатимемо взаємодію із сервером MongoDB переважно через вище використану оболонку mongo. Однак, ми також можемо взаємодіяти з mongodб у наших додатках, написаних на PHP, C++, C# та інших мовах програмування. І для цієї мети необхідний спеціальний драйвер.

На офсайті на сторінці <https://docs.mongodb.com/ecosystem/drivers/> можна знайти список драйверів для всіх підтримуваних мов програмування, зокрема, для PHP, C, C++, C#, Java, Go, Python, Rust, Ruby, Scala, Swift, а також для Node.js.

Роботу з драйверами на конкретних мовах програмування буде розглянуто у відповідних розділах, присвячених цим мовам.

Встановлення та початок роботи з MongoDB на Mac OS

Для встановлення MongoDB на Mac OS рекомендованим і при цьому найпростішим способом є застосування пакетного менеджера homebrew. Тому для початку необхідно встановити сам пакетний менеджер homebrew. Для цього в терміналі необхідно виконати команду:

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Встановлення mongodб

Після встановлення homebrew для встановлення mongodб введемо в терміналі таку команду:

```
brew tap mongodb/brew
```

І після цього введемо в терміналі ще одну команду:

```
brew install mongodb-community@7.0
```

Зверніть увагу на цифри версії - mongodb-community@7.0. Тобто в цьому випадку встановлюється версія 7.0 Community-випуску Mongo. Але за необхідності можна змінити ці цифри і задати іншу версію.

У процесі встановлення буде встановлено такі файли:

mongosh: представляє консольний інтерфейс для взаємодії з базами даних, свого роду консольний клієнт

mongod: сервер баз даних MongoDB. Він обробляє запити, керує форматом даних і виконує різні операції у фоновому режимі з управління базами даних

mongos: служба маршрутизації MongoDB, яка допомагає обробляти запити і визначати місце розташування даних у кластері MongoDB

Крім того, під час встановлення залежно від архітектури комп'ютера (intel або arm) буде створено такі файли та каталоги:

Файли/каталоги	Intel	Apple M1
Файл конфігурації	/usr/local/etc/mongod.conf	/opt/homebrew/etc/mongod.conf
Каталог логів	/usr/local/var/log/mongodb	/opt/homebrew/var/log/mongodb
Каталог баз даних	/usr/local/var/mongodb	/opt/homebrew/var/mongodb

Запуск MongoDB

Для запуску сервера MongoDB виконаємо в терміналі таку команду:

```
brew services start mongodb-community@7.0
```

Для вимкнення сервера застосовується команда

```
brew services stop mongodb-community@7.0
```

Підключення до сервера MongoDB

Отже, після вдалого запуску сервера ми зможемо проводити операції з бд через оболонку mongosh. Запустимо її, виконавши в терміналі команду:

```
mongosh
```

```
eugene — mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...
MacBook-Pro-Eugene:~ eugene$ mongosh

Current Mongosh Log ID: 622ca5a5921183310bf8297b
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverS
electionTimeoutMS=2000&appName=mongosh+1.2.3
Using MongoDB:      5.0.6
Using Mongosh:       1.2.3

For mongosh info see: https://docs.mongodb.com/mongosh-shell/

To help improve our products, anonymous usage data is collected and sent to Mong
oDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
  The server generated these startup warnings when booting:
  2022-03-12T16:50:31.145+03:00: Access control is not enabled for the database
  . Read and write access to data and configuration is unrestricted
-----

test>
```

Це консольна оболонка для взаємодії із сервером, через яку можна керувати даними. Другим рядком ця оболонка говорить про підключення до сервера mongod.

Тепер зробимо якісь найпростіші дії. Введемо в терміналі послідовно такі команди і після кожної команди натиснемо на Enter:

```
use userdb
db.users.insertOne( { name: "Tom" } )
db.users.find()
```

Перша команда `use userdb` встановлює як використовувану базу даних `userdb`. Навіть якщо такої бд немає, то вона створюється автоматично. І далі `db` представлятиме поточну базу даних - тобто базу даних `userdb`. Після `db` йде `users` - це колекція, в яку потім ми додаємо новий об'єкт. Якщо в SQL нам треба створювати таблиці заздалегідь, то колекції MongoDB створює самостійно за їх відсутності.

За допомогою методу `db.users.insertOne()` в колекцію `users` бази даних `userdb` додається об'єкт `{name: "Tom" }`. Опис об'єкта, що додається, визначається у форматі, з яким ви, можливо, знайомі, якщо мали справу з форматом JSON. Тобто в цьому разі в об'єкта визначено один ключ `"name"`, якому зіставляється значення `"Tom"`. Тобто ми додаємо користувача з ім'ям Tom.

Якщо об'єкт було успішно додано, то консоль виведе результат операції, зокрема, ідентифікатор доданого об'єкта.

А третя команда `db.users.find()` виводить на екран усі об'єкти з бд `test`.



```
eugene — mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...

test>

[test> use userdb
switched to db userdb
[userdb> db.users.insertOne( { name: "Tom" } )
{
  acknowledged: true,
  insertedId: ObjectId("622ca765921183310bf8297c")
}
[userdb> db.users.find()
[ { _id: ObjectId("622ca765921183310bf8297c"), name: 'Tom' } ]
userdb>
```

З виведення ви можете побачити, що до початкових значень об'єкта було додано якесь незрозуміле поле ObjectId. Як ви пам'ятаєте, MongoDB як унікальні ідентифікатори документа використовує поле `_id`. І в даному випадку ObjectId якраз і представляє значення для ідентифікатора `_id`.

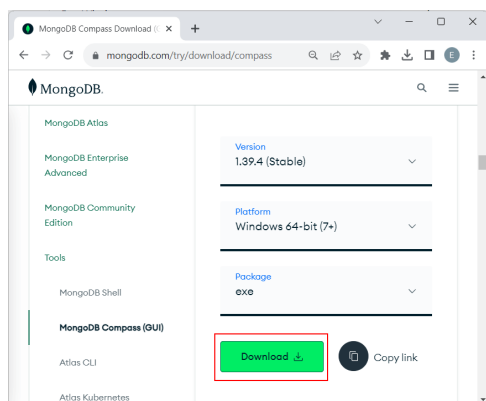
Графічний клієнт Compass

Для роботи з MongoDB також можна використовувати офіційний графічний клієнт MongoDB Compass. Для його завантаження перейдемо за адресою <https://www.mongodb.com/try/download/compass>.

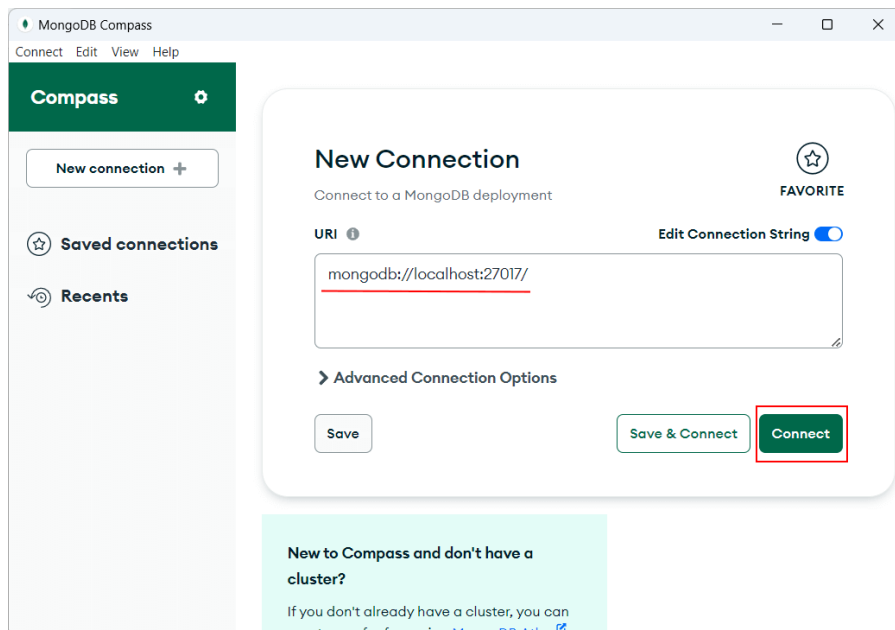
На цій сторінці ми можемо вибрати опції для завантаження - версію Compass і цільову операційну систему.

Розглянемо на прикладі встановлення на Windows.

Для Windows нам доступний скомпільований додаток у вигляді єдиного файлу exe. Завантажимо його:

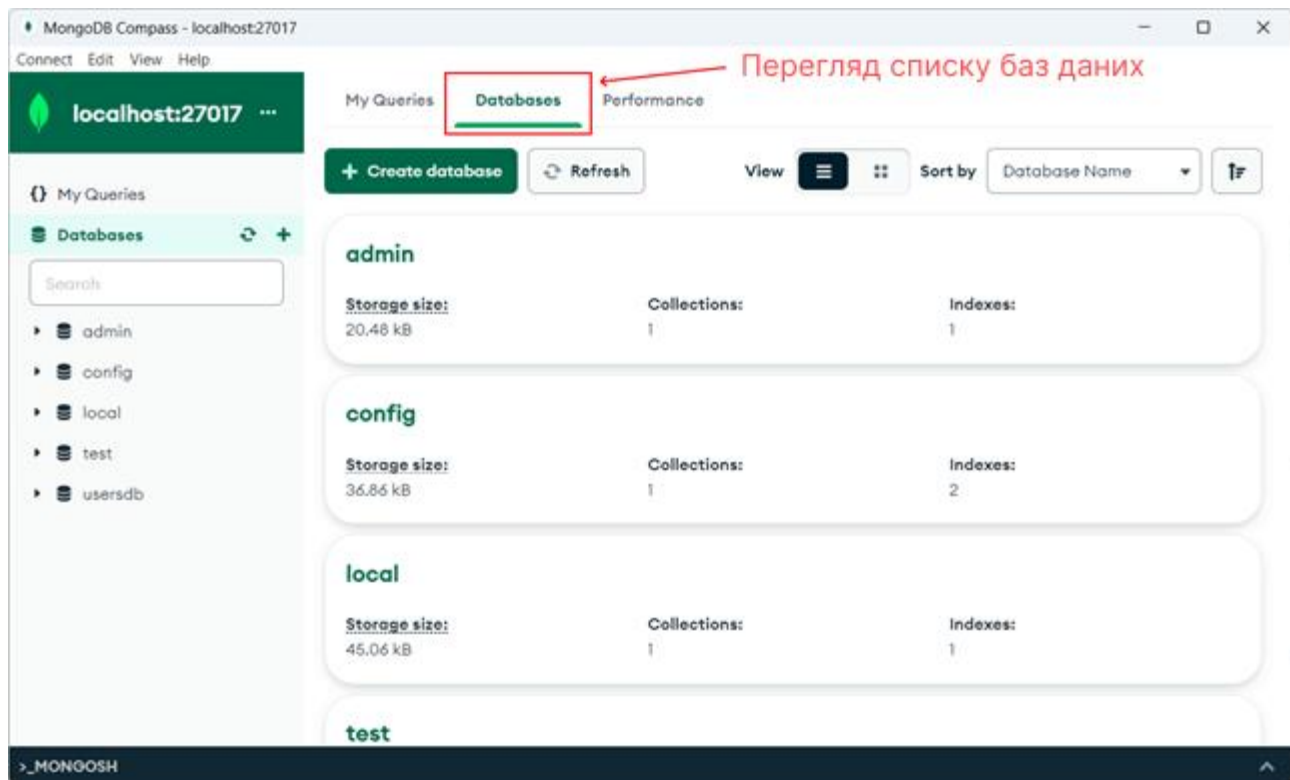


Після його завантаження спочатку запусимо сервер `mongod`, а потім запусимо завантажений файл `Mongodb Compass`:

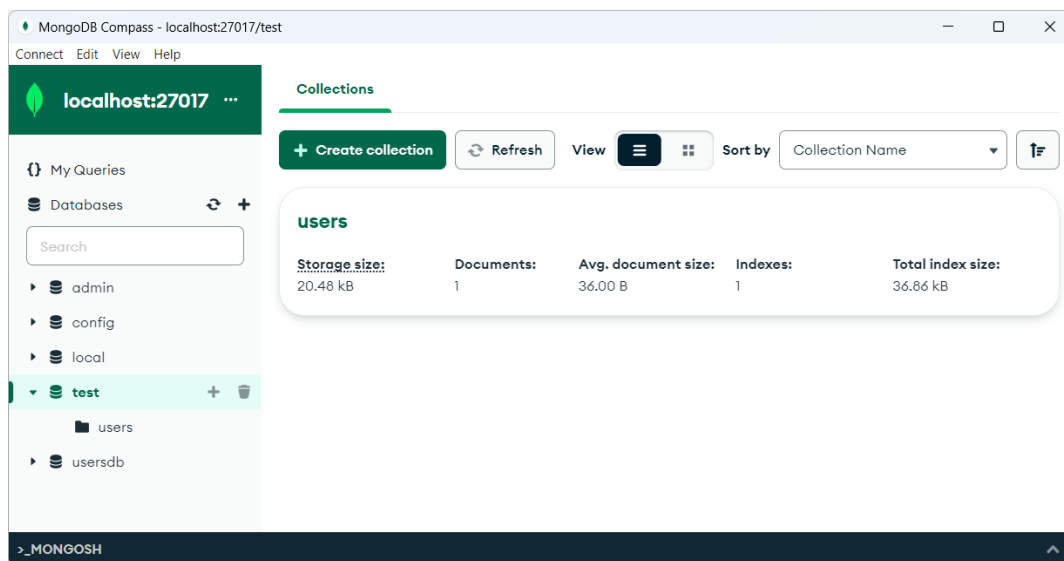


За замовчуванням нам відкриється вікно для створення нового підключення. Воно має тільки одне поле - URI, куди треба ввести рядок підключення. За замовчуванням тут може відображатися рядок підключення до локального сервера - `mongodb://localhost:27017/`. Рядок підключення може відрізнятись залежно від місця запуску сервера MongoDB (локально на поточному комп'ютері або десь на іншому мережевому ресурсі), використовуваного логіна і пароля та інших налаштувань. У цьому разі ми розглянемо підключення до локально запущеного сервера `mongodb`, установку якого було розглянуто в минулій темі. Тому залишимо в ньому значення `mongodb://localhost:27017` (або введемо цей рядок, якщо поле порожнє) і натиснемо на кнопку **Connect**.

Після успішного підключення нам відкриється вміст сервера, де ми можемо подивитися різну інформацію, наприклад, список баз даних, що є на сервері:

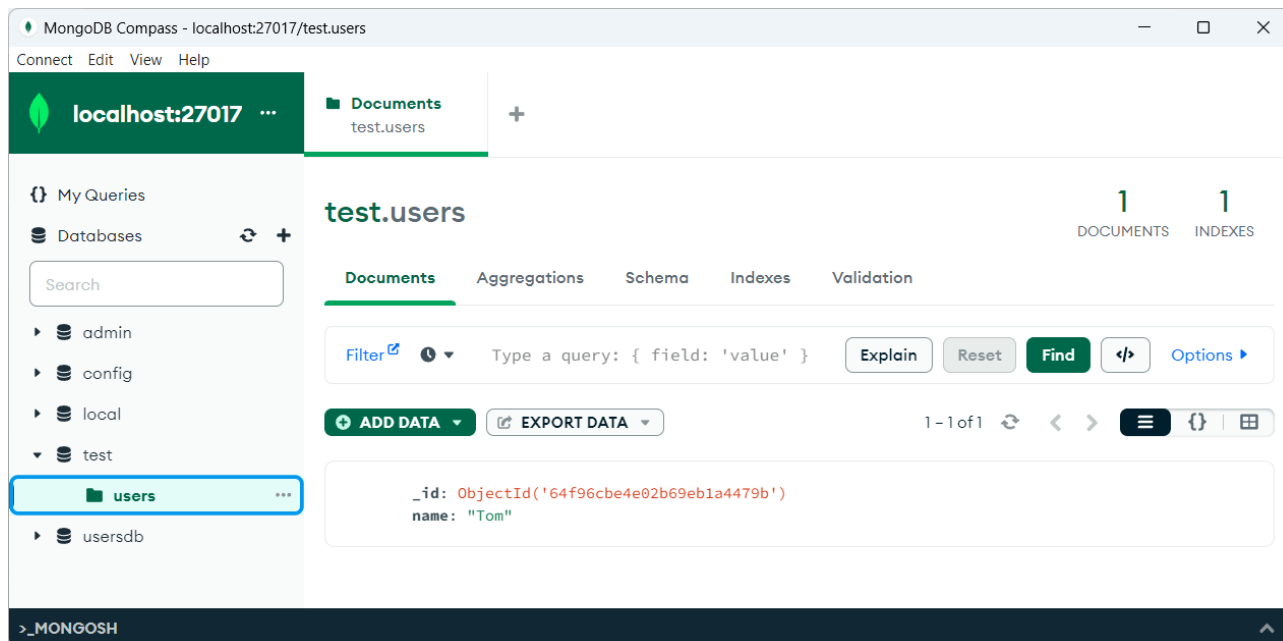


Ми можемо вибрати певну базу даних і отримати по ній інформацію, зокрема, побачити набір колекцій у бд, скільки вони займають даних.



У цьому випадку на скріншоті відображається список колекцій бази даних test, створених у минулій темі за допомогою оболонки mongosh.

Натиснувши на певну колекцію, можна побачити графічно всі дані, які є в колекції:

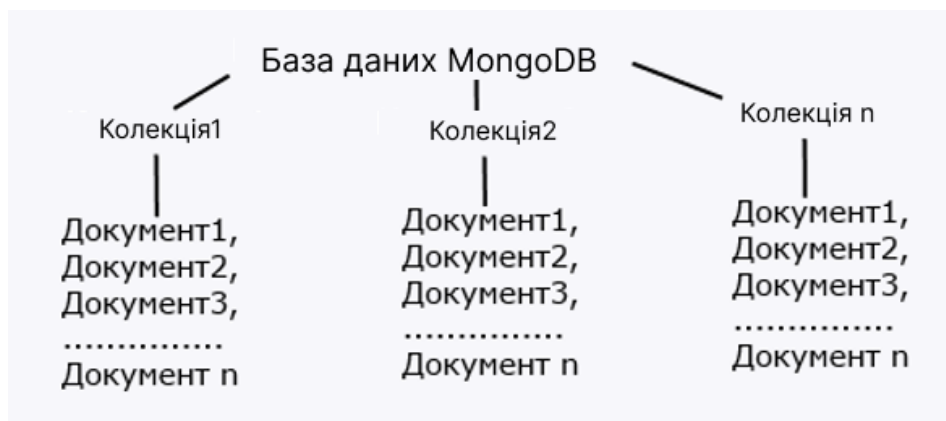


Використовуючи графічний інтерфейс програми Compass, ми можемо керувати цими даними, додавати, змінювати, видаляти їх.

Робота з базою даних

Як влаштована база даних. Документи

Усю модель влаштування бази даних у MongoDB можна представити таким чином:



Якщо в реляційних бд вміст складають таблиці, то в mongodb база даних складається з колекцій.

Кожна колекція має своє унікальне ім'я - довільний ідентифікатор, що складається з не більше ніж 128 різних алфавітно-цифрових символів і знака підкреслення.

На відміну від реляційних баз даних MongoDB не використовує табличний пристрій із чітко заданою кількістю стовпців і типів даних. MongoDB є документо-орієнтованою системою, в якій центральним поняттям є документ.

Документ можна уявити як об'єкт, що зберігає деяку інформацію. У певному сенсі він подібний до рядків у реляційних субд, де рядки зберігають інформацію про окремий елемент. Наприклад, типовий документ:

```
{
  "name": "Tom",
  "surname": "Smith",
  "age": "37",
  "company": {
    "name" : "Microsoft",
    "salary" : "100"
  }
}
```

Документ представляє набір пар ключ-значення. Наприклад, у виразі "name": "Tom" "name" представляє ключ, а "Tom" - значення.

Ключі представляють рядки. Значення ж можуть відрізнятися за типом даних. У даному випадку у нас майже всі значення також представляють строковий тип, і лише один ключ (company) посилається на окремий об'єкт.

Усього є такі типи значень:

String: строковий тип даних, (для рядків використовується кодування UTF-8)

Array (масив): тип даних для зберігання масивів елементів

Binary data (двійкові дані): тип для зберігання даних у бінарному форматі

Boolean: булевий тип даних, що зберігає логічні значення TRUE або FALSE, наприклад, {"married": FALSE}

Date: зберігає дату у форматі часу Unix

Double: числовий тип даних для зберігання чисел із плаваючою крапкою

Integer: використовується для зберігання цілочисельних значень розміром 32 біти, наприклад, {"age": 29}

Long: використовується для зберігання цілочисельних значень розміром 64 біти

JavaScript: тип даних для зберігання коду javascript

Min key/Max key: використовуються для порівняння значень з найменшим/найбільшим елементів BSON

Null: тип даних для зберігання значення Null

Object: об'єкт, який містить набір властивостей

ObjectId: тип даних для зберігання id документа

Regular expression: застосовується для зберігання регулярних виразів

Decimal128: тип даних для зберігання десяткових дробових чисел розміром 128 біт, які дають змогу розв'язати проблеми з проблемою точності обчислень під час використання дробових чисел, що становлять тип Double.

Timestamp: застосовується для зберігання часу

На відміну від рядків документи можуть містити різномірну інформацію. Так, поруч із документом, описаним вище, в одній колекції може перебувати інший об'єкт, наприклад:

```
{
  "name": "Bob",
  "birthday": "1985.06.28",
  "place" : "Berlin",
  "languages" :[
    "english",
    "german",
    "spanish"
  ]
}
```

Здавалося б різні об'єкти за винятком окремих властивостей, але всі вони можуть перебувати в одній колекції.

Ще пара важливих зауважень: у MongoDB запити мають регістрозалежність і сувору типізацію. Тобто наступні два документи не будуть ідентичні:

```
{"age" : "28"}
{"age" : 28}
```

Якщо в першому випадку для ключа age визначено як значення рядок, то в другому випадку значенням є число.

Ідентифікатор документа

Для кожного документа в MongoDB визначено унікальний ідентифікатор, який називається `_id`. Під час додавання документа в колекцію цей ідентифікатор

створюється автоматично. Однак розробник може сам явно задати ідентифікатор, а не покладатися на автоматично генеровані, вказавши відповідний ключ і його значення в документі.

Це поле повинно мати унікальне значення в межах колекції. І якщо ми спробуємо додати в колекцію два документи з однаковим ідентифікатором, то додасться тільки один із них, а під час додавання другого ми отримаємо помилку.

Якщо ідентифікатор не задано явно, то MongoDB створює спеціальне бінарне значення розміром 12 байт. Це значення складається з декількох сегментів: значення типу timestamp розміром 4 байти (яке представляє кількість секунд з моменту початку епохи Unix), випадкове число з 5 байт і лічильник з 3 байт, який ініціалізовано випадковим числом. Така модель побудови ідентифікатора гарантує з високою часткою ймовірності, що він матиме унікальне значення.

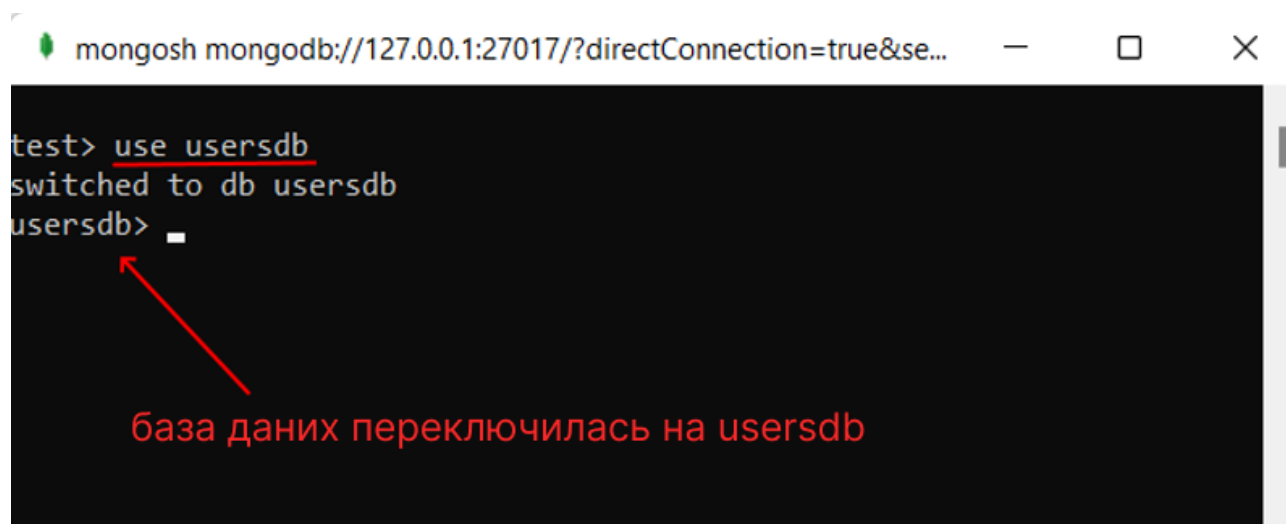
Установка та адміністрування бази даних

У цій та подальших статтях ми розглянемо базові операції з даними в MongoDB із застосуванням як консольної оболонки `mongosh`, так і графічного клієнта MongoDB Compass. Однак у будь-якому разі під час початку роботи із сервером слід не забувати запускати сам сервер - тобто застосунок `mongod`.

Починаючи працювати з MongoDB у консольній оболонці `mongosh`, насамперед треба встановити потрібну нам базу даних як поточну, щоб потім її використовувати. Для цього треба використовувати команду `use`, після якої йде назва бази даних. При цьому не важливо, чи існує така бд чи ні. Якщо її немає, то MongoDB автоматично створить її під час додавання в неї даних.

Отже, запусимо консольну оболонку `mongosh` і введемо там таку команду:

use usersdb



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...
test> use usersdb
switched to db usersdb
usersdb> 
```

база даних переключилась на usersdb

Тепер як поточну буде встановлено БД usersdb. При цьому не важливо, що спочатку такої бази даних може не існувати: якщо її не існує, то під час першої операції вона створюється.

Якщо ви раптом не впевнені, а чи існує вже база даних з такою назвою, то за допомогою команди `show dbs` можна вивести назви всіх наявних бд на консоль:

```
usersdb> show dbs
admin  40.00 KiB
config 72.00 KiB
local  72.00 KiB
test   40.00 KiB
usersdb>
```

Зверніть увагу, що в списку баз даних ще немає бд usersdb, тому що я з нею не проводив ще ніяких операцій.

Для бази даних можна задати будь-яке ім'я, проте є деякі обмеження. Наприклад, в імені не повинно бути символів /, \, ., ", *, <, >, :, |, ?, \$. Крім того, імена баз даних обмежені 64 байтами.

Також є зарезервовані імена, які не можна використовувати: local, admin, config. Ці імена представляють бази даних, які вже мають за замовчуванням на сервері і призначені для службових цілей.

Причому, як ви бачите, бд test у цьому списку немає, оскільки я в неї ще не додав дані.

Крім баз даних ми можемо переглянути список усіх колекцій у поточній бд за допомогою команди

`show collections`

Отримання статистики

Використовуючи команду `db.stats()`, можна отримати статистику за поточною базою даних. Наприклад, у нас як поточну встановлено базу даних test:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...
usersdb> use test
switched to db test
test> show collections
users
test> db.stats()
{
  db: 'test',
  collections: 1,
  views: 0,
  objects: 1,
  avgObjSize: 36,
  dataSize: 36,
  storageSize: 20480,
  indexes: 1,
  indexSize: 20480,
  totalSize: 40960,
  scaleFactor: 1,
  fsUsedSize: 179737477120,
test>
ok: 1
```

Схожим чином ми можемо дізнатися всю статистику за окремою колекцією. Наприклад, дізнаємося статистику за колекцією users: `db.users.stats()`

Додавання даних

Встановивши бд, тепер ми можемо додати в неї дані. Усі дані зберігаються в бд у форматі BSON, який близький до JSON, тому нам треба також вводити дані в цьому форматі. І хоча у нас, можливо, наразі немає жодної колекції, але під час додавання в неї даних вона автоматично створюється.

Як раніше йшлося, ім'я колекції - довільний ідентифікатор, що складається з не більше ніж 128 різних алфавітно-цифрових символів і знака підкреслення. Водночас ім'я колекції не повинно починатися з префікса `system`, оскільки він зарезервований для внутрішніх колекцій (наприклад, колекція `system.users` містить усіх користувачів бази даних). І також ім'я не повинно містити знака долара - `$`.

Для додавання в колекцію можуть використовуватися три її методи:

`insertOne()`: додає один документ

`insertMany()`: додає кілька документів

Припустимо, ми використовуємо базу даних `test`. Додамо в неї один документ:

```
test> db.users.insertOne({"name": "Tom", "age": 28, "languages": ["english", "spanish"]})
```

Документ являє собою набір пар ключ-значення. У цьому випадку документ, що додається, має три ключі: name, age, languages, і кожному з них зіставляє певне значення. Наприклад, ключу languages в якості значення зіставляється масив.

Варто зазначити, що назви ключів можуть використовуватися в лапках, а можуть і без лапок.

Деякі обмеження при використанні імен ключів:

Символ \$ не може бути першим символом в імені ключа

Ім'я ключа не може містити символ крапки .

Під час додавання даних, якщо ми явно не надали значення для поля "_id" (тобто унікального ідентифікатора документа), то воно генерується автоматично. Так, після виконання операції додавання консоль виведе згенерований для доданого документа ідентифікатор:

```
test> db.users.insertOne({"name": "Tom", "age": 28, "languages": ["english", "spanish"]})
{
  acknowledged: true,
  insertedId: ObjectId("62e27b1b06adfcddf4619fc1")
}
test>
```

У відповіді сервера ми отримаємо об'єкт, у кого параметр insertedId міститиме ідентифікатор.

Але в принципі ми можемо самі встановити цей ідентифікатор під час додавання даних:

```
test> db.users.insertOne({"_id": 123457, "name": "Tom", "age": 28, "languages": ["english", "spanish"]})
```

або використовувати для ідентифікатора тип ObjectId

```
test> db.users.insertOne({"_id": ObjectId("62e27b1b06adfcddf4619fc6"), "name": "Tom", "age": 28, "languages": ["english", "spanish"]})
```

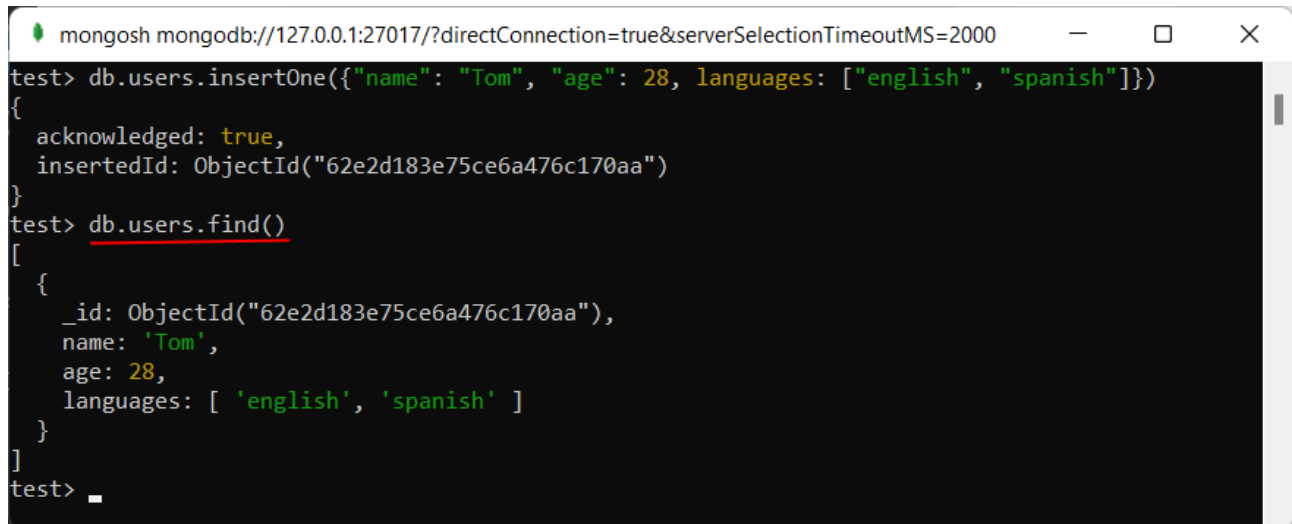
Варто враховувати, що якщо для визначення ідентифікатора застосовується тип ObjectId, то він має містити рядок 12 байт або рядок із 24 символів.

У разі вдалого додавання на консоль буде виведено ідентифікатор доданого документа.

І щоб переконатися, що документ у бд, ми його виводимо функцією `find`.

```
test> db.users.find()
```

За замовчуванням він виводить усі документи колекції:

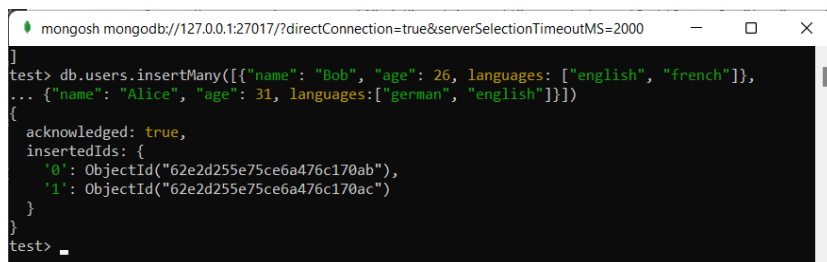


```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
test> db.users.insertOne({"name": "Tom", "age": 28, "languages": ["english", "spanish"]})
{
  acknowledged: true,
  insertedId: ObjectId("62e2d183e75ce6a476c170aa")
}
test> db.users.find()
[
  {
    _id: ObjectId("62e2d183e75ce6a476c170aa"),
    name: 'Tom',
    age: 28,
    languages: [ 'english', 'spanish' ]
  }
]
test> .
```

Якщо треба додати низку документів, то ми можемо скористатися методом `insertMany()`, який приймає масив об'єктів:

```
db.users.insertMany([{"name": "Bob", "age": 26, "languages": ["english", "french"]},
{"name": "Alice", "age": 31, "languages": ["german", "english"]}])
```

Після додавання консоль виводить ідентифікатори доданих документів:



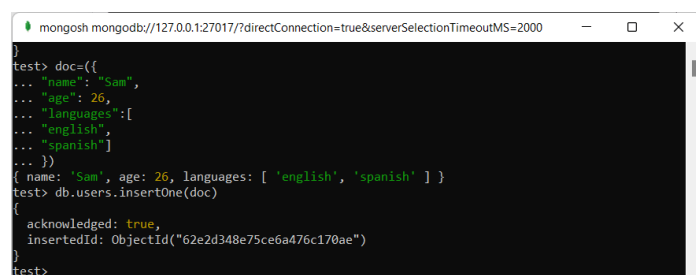
```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
test> db.users.insertMany([{"name": "Bob", "age": 26, "languages": ["english", "french"]},
... {"name": "Alice", "age": 31, "languages": ["german", "english"]}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("62e2d255e75ce6a476c170ab"),
    '1': ObjectId("62e2d255e75ce6a476c170ac")
  }
}
test> .
```

Є ще один спосіб додавання в бд документа, який включає два етапи: визначення документа (`document = ( { ... } )`) і власне його додавання:

```
document=({"name": "Bill", "age": 32, "languages": ["english", "french"]})
db.users.insertOne(document)
```

Можливо, не всім буде зручно вводити в один рядок усі пари ключів і властивостей. Але інтелектуальний інтерпретатор MongoDB на основі javascript дозволяє також вводити і багаторядкові команди. Якщо вираз не закінчено (з точки зору мови 500

JavaScript), і ви натискаєте Enter, то введення наступної частини виразу автоматично переноситься на наступний рядок:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
test> doc={
...   "name": "Sam",
...   "age": 26,
...   "languages": [
...     "english",
...     "spanish"
...   ]
... }
{ name: 'Sam', age: 26, languages: [ 'english', 'spanish' ] }
test> db.users.insertOne(doc)
{
  acknowledged: true,
  insertedId: ObjectId("62e2d348e75ce6a476c170ae")
}
test>
```

Завантаження даних із файлу

Дані для бази даних mongodb можна визначати у звичайному текстовому файлі, що досить зручно, оскільки ми можемо переносити або пересилати цей файл незалежно від бази даних mongodb. Наприклад, визначимо де-небудь на жорсткому диску файл users.js з таким вмістом:

```
db.users.insertMany([
  {"name": "Alice", "age": 31, "languages": ["english", "french"]},
  {"name": "Lene", "age": 29, "languages": ["english", "spanish"]},
  {"name": "Kate", "age": 30, "languages": ["german", "russian"]}
])
```

Тобто тут за допомогою методу insertMany додаються три документи в колекцію users.

Для загузки файлу в поточну базу даних застосовується функція load(), в яку як параметр передається шлях до файлу:

```
load("D:/users.js")
```

У цьому випадку передбачається, що файл розташовується за шляхом "D:/users.js".

Вибірка і фільтрація

Найпростішим способом вибірки документів з колекції є використання функції find(). Дія цієї функції багато в чому аналогічна звичайному запити SELECT * FROM Table, який застосовується в SQL і який витягує всі рядки. Наприклад, щоб витягти всі документи з колекції users, створеної в минулій темі, ми можемо використовувати команду:

```
db.users.find()
```



```
Выбрать mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutM...
test> db.users.find()
[
  {
    _id: ObjectId("62e2d183e75ce6a476c170aa"),
    name: 'Tom',
    age: 28,
    languages: [ 'english', 'spanish' ]
  },
  {
    _id: ObjectId("62e2d255e75ce6a476c170ab"),
    name: 'Bob',
    age: 26,
    languages: [ 'english', 'french' ]
  },
  {
    _id: ObjectId("62e2d255e75ce6a476c170ac"),
    name: 'Alice',
    age: 31,
  }
]
```

Фільтрація даних

Однак що, якщо нам треба отримати не всі документи, а тільки ті, які задовольняють певній вимозі. Наприклад, ми раніше в базу додали такі документи:

```
db.users.insertOne({"name": "Tom", "age": 28, "languages": ["english", "spanish"]})
db.users.insertOne({"name": "Bill", "age": 32, "languages": ["english", "french"]})
db.users.insertOne({"name": "Tom", "age": 32, "languages": ["english", "german"]})
```

Виведемо всі документи, в яких name=Tom:

```
db.users.find({name: "Tom"})
```

Такий запит виведе нам два документи, в яких name=Tom.

```
test> db.users.find({name: "Tom"})
[
  {
    _id: ObjectId("62e2d6a5e75ce6a476c170b3"),
    name: 'Tom',
    age: 28,
    languages: [ 'english', 'spanish' ]
  },
  {
    _id: ObjectId("62e2d348e75ce6a476c170ae"),
    _id: ObjectId("62e2d6a5e75ce6a476c170b5"),
    name: 'Tom',
    age: 32,
    languages: [ 'english', 'spanish' ]
  },
  {
    _id: ObjectId("62e2d3d5e75ce6a476c170af"),
    name: 'Tom',
    age: 32,
    languages: [ 'english', 'german' ]
  }
]
```

Тепер складніший запит: нам треба вивести ті об'єкти, у яких name=Tom і одночасно age=32. Тобто мовою SQL це могло б виглядати так: SELECT * FROM Table WHERE Name='Tom' AND Age=32. Цьому критерію в нас відповідає останній доданий об'єкт. Тоді ми можемо написати такий запит:

```
db.users.find({name: "Tom", age: 32})
```

Фільтрація за відсутніми властивостями

Якись документи можуть мати певну властивість, інші можуть її не мати. Що якщо ми хочемо отримати документи, в яких відсутня певна властивість? У цьому випадку для властивості передається значення `null`. Наприклад, знайдемо всі документи, де відсутня властивість `languages`:

```
db.users.find({languages: null})
```

Або знайдемо всі документи, де `name="Tom"`, але властивість `languages` не визначена.

```
db.users.find({name: "Tom", languages: null})
```

Фільтрація за елементами масиву

Також нескладно відшукати за елементом у масиві. Наприклад, такий запит виводить усі документи, у яких у масиві `languages` є `english`:

```
db.users.find({languages: "english"})
```

Ускладнимо запит і отримаємо ті документи, у яких у масиві `languages` одночасно дві мови: `"english"` і `"german"`:

```
db.users.find({languages: ["english", "german"]})
```

Причому саме в цьому порядку, де `"english"` визначено першим, а `"german"` - другим.

Тепер виведемо всі документи, у яких `"english"` у масиві `languages` перебуває на першому місці:

```
db.users.find({"languages.0": "english"})
```

Зверніть увагу, що `"languages.0"` надає складну властивість і тому береться в лапки. Відповідно якщо нам треба вивести документи, де `english` на другому місці (наприклад, `["german", "english"]`), то замість нуля ставимо одиницю: `"languages.1"`.

Розглянемо складніший приклад, де елемент масиву представляє складний об'єкт. Припустимо, у нас у базі даних такі документи:

```
db.users.insertOne({"name": "Bob", "age": 28, friends: [{"name": "Tim"}, {"name": "Tom"}]})
```

```
db.users.insertOne({"name": "Tim", "age": 29, friends: [{"name": "Bob"}, {"name": "Tom"}]})
```

```
db.users.insertOne({"name": "Sam", "age": 31, friends: [{"name": "Tom"}]})
```

```
db.users.insertOne({"name": "Tom", "age": 32, friends: [{"name": "Bob"}, {"name": "Tim"}], {"name": "Sam"}})
```

Виберемо всі документи, де в масиві friends властивість name першого елемента дорівнює "Bob":

```
test> db.users.find({"friends.0.name": "Bob"})
```

Консольне виведення:

```
test> db.users.find({"friends.0.name": "Bob"})
[
  {
    _id: ObjectId("62e39da1c881653067e87901"),
    name: 'Tim',
    age: 29,
    friends: [ { name: 'Bob' }, { name: 'Tom' } ]
  },
  {
    _id: ObjectId("62e39da1c881653067e87903"),
    name: 'Tom',
    age: 32,
    friends: [ { name: 'Bob' }, { name: 'Tim' }, { name: 'Sam' } ]
  }
]
test>
```

Проекція

Документ може мати безліч полів, але не всі ці поля нам можуть бути потрібні й важливі під час запиту. І в цьому випадку ми можемо включити у вибірку тільки потрібні поля, використавши проекцію. Наприклад, виведемо тільки порцію інформації, наприклад, значення полів "age" у всіх документах, у яких name=Tom:

```
db.users.find({name: "Tom"}, {age: 1})
```

Використання одиниці як параметра {age: 1} вказує, що запит має повернути тільки зміст властивості age.

```
test> db.users.find({name: "Tom"}, {age: 1})
[
  { _id: ObjectId("62e2d6a5e75ce6a476c170b3"), age: 28 },
  { _id: ObjectId("62e2d6a5e75ce6a476c170b5"), age: 32 },
  { _id: ObjectId("62e2d799e75ce6a476c170b7"), age: 28 },
  { _id: ObjectId("62e39da1c881653067e87903"), age: 32 }
]
test>
```

І зворотна ситуація: ми хочемо знайти всі поля документа крім властивості `age`. У цьому разі як параметр вказуємо `0`:

```
db.persons.find({name: "Tom"}, {age: 0})
```

При цьому треба враховувати, що навіть якщо ми зазначимо, що ми хочемо отримати тільки поле `name`, поле `_id` також буде включено в результуючу вибірку. Тому, якщо ми не хочемо бачити це поле у вибірці, то треба явно вказати: `{"_id":0}`

Альтернативно замість `1` і `0` можна використовувати `true` і `false`:

```
db.users.find({name: "Tom"}, {age: true, _id: false})
```

Якщо ми не хочемо при цьому конкретизувати вибірку, а хочемо вивести всі документи, то можна залишити перші фігурні дужки порожніми:

```
db.users.find({}, {age: 1, _id: 0})
```

Запит до вкладених об'єктів

Попередні запити застосовувалися до простих об'єктів. Але документи можуть бути дуже складними за структурою. Наприклад, додамо в колекцію `users` такий документ:

```
db.users.insertOne({"name": "Alex", "age": 28, company: {"name": "Microsoft", "country": "USA"}})
```

Тут визначається вкладений об'єкт із ключем `company`. І щоб знайти всі документи, у яких у ключі `company` вкладена властивість `name=microsoft`, нам треба використовувати оператор крапку:

```
db.users.find({"company.name": "Microsoft"})
```

Використання JavaScript

Крім виконання запитів до бази даних ми можемо виконувати вирази JavaScript. Наприклад, ми можемо створити якусь функцію і застосовувати її:

```
function sqrt(n) { return n*n; }  
sqrt(5)
```

Консольне виведення:

```
test> function sqrt(n) { return n*n; }  
[Function: sqrt]  
test> sqrt(5)  
25  
test>
```

І подібні функції та вирази JavaScript ми можемо застосовувати в запитах до БД. Наприклад, знайдемо всі документи, де поле `age` дорівнює `sqrt(5)+3`:

```
test> db.users.find({age: sqrt(5)+3})  
[  
  {  
    _id: ObjectId("62e2d6a5e75ce6a476c170b3"),  
    name: 'Tom',  
    age: 28,  
    languages: [ 'english', 'spanish' ]  
  },  
  { _id: ObjectId("62e2d76ae75ce6a476c170b6"), name: 'Tomas', age: 28 },  
  { _id: ObjectId("62e2d799e75ce6a476c170b7"), name: 'Tom', age: 28 },  
  {  
    _id: ObjectId("62e39da1c881653067e87900"),  
    name: 'Bob',  
    age: 28  
  }  
]  
test>
```

Використання регулярних виразів

Ще однією чудовою можливістю при побудові запитів є використання регулярних виразів. Наприклад, знайдемо всі документи, у яких значення ключа `name` починається з літери `B`:

```
db.users.find({name:/^B\w+/i})
```

Приблизний консольний вивід:

```
test> db.users.find({name:/^B\w+/i})
[
  {
    _id: ObjectId("62e2d6a5e75ce6a476c170b4"),
    name: 'Bill',
    age: 32,
    languages: [ 'english', 'french' ]
  },
  {
    _id: ObjectId("62e39da1c881653067e87900"),
    name: 'Bob',
    age: 28
  }
]
test>
```

Пошук одиночного документа

Якщо всі документи витягуються функцією `find`, то одиночний документ витягується функцією `findOne`

Наприклад, виберемо один елемент з `name="Tom"`:

```
test> db.users.findOne({name: "Tom"})
{
  _id: ObjectId("62e2d6a5e75ce6a476c170b3"),
  name: 'Tom',
  age: 28,
  languages: [ 'english', 'spanish' ]
}
test>
```

Курсори

Результат вибірки, одержуваної за допомогою функції `find`, називається курсором. За необхідності ми можемо передати курсор в окрему змінну:

```
var cursor = db.users.find()
```

Курсори інкапсулюють у собі набори об'єктів, які отримують із бд. Використовуючи синтаксис мови javascript і методи курсорів, ми можемо вивести отримані документи на екран і якось їх обробити. Наприклад:

```
var cursor = db.users.find()
```

```
while(cursor.hasNext()){
  obj = cursor.next();
  print(obj["name"]);
}
```

Курсор має метод `hasNext`, який показує під час перебору, чи є ще в наборі документ. А метод `next` витягує поточний документ і переміщує курсор до наступного документа в наборі. У підсумку у змінній `obj` опиняється документ, до полів якого ми можемо отримати доступ.

```
test> var cursor = db.users.find()

test> while(cursor.hasNext()){
...  obj = cursor.next();
...  print(obj["name"]);
... }
Tom
Bob
Sam
test>
```

Також для перебору документів у курсорі як альтернативу ми можемо використовувати конструкцію ітератора javascript - `forEach`:

```
var cursor = db.users.find()
cursor.forEach(function(obj){
  print(obj.name);
})
```

Пагінація та сортування

MongoDB представляє низку функцій, які допомагають керувати вибіркою з бд. Одна з них - функція `limit`. Вона задає максимально допустиму кількість одержуваних документів. Кількість передається у вигляді числового параметра. Наприклад, обмежимо вибірку трьома документами:

```
db.users.find().limit(3)
```

У цьому разі ми отримаємо перші три документи (якщо в колекції 3 і більше документів). Але що, якщо ми хочемо зробити вибірку не спочатку, а пропустивши якусь кількість документів? У цьому нам допоможе функція `skip`. Наприклад, пропустимо перші три записи:

```
db.users.find().skip(3)
```

Комбінуючи обидві функції, ми можемо отримати певну кількість документів, починаючи з певного документа. Наприклад, виберемо документи з 4 по 6:

```
db.users.find().skip(3).limit(3)
```

MongoDB надає можливості відсортувати отриманий з бд набір даних за допомогою функції `sort`. Передаючи в цю функцію значення 1 або -1, ми можемо вказати в якому порядку сортувати: за зростанням (1) або за спаданням (-1). Багато в чому ця функція за дією аналогічна виразу `ORDER BY` у `SQL`. Наприклад, сортування за зростанням за полем `name`:

```
db.users.find().sort({name: 1})
```

Наприклад, виведемо з бд тільки значення поля `"name"`, відсортувавши їх за зростанням:

```
test> db.users.find({}, {name:1, _id: 0}).sort({name: 1})
[
  { name: 'Bill' },
  { name: 'Bob' },
  { name: 'Sam' },
  { name: 'Tim' },
  { name: 'Tom' },
  { name: 'Tom' },
  { name: 'Tom' },
  { name: 'Tom' },
  { name: 'Tomas' }
]
test>
```

Оператор \$slice

`$slice` є певною мірою комбінацією функцій `limit` і `skip`. Але на відміну від них `$slice` може працювати з масивами.

Оператор `$slice` має дві форми:

`$slice: limit`

`$slice: skip, limit`

Параметр `limit` вказує на загальну кількість документів, що повертаються. Параметр `skip` вказує на зміщення щодо початку (як функція `skip`).

Наприклад, у кожному документі визначено масив `languages` для зберігання мов, якими розмовляє людина. Їх може бути і 1, і 2, і 3, і більше. І припустимо, раніше ми додали такий об'єкт:

```
db.users.insertOne({"name": "Tom", "age": 32, "languages": ["english", "german", "spanish"]})
```

І ми хочемо під час виведення документів зробити так, щоб у вибірку потрапляла тільки одна мова з масиву `languages`, а не весь масив:

```
db.users.find ({name: "Tom"}, {languages: {$slice : 1}})
```

Цей запит під час вилучення документа залишить у результаті тільки першу мову з масиву `languages`, тобто в цьому випадку `english`.

```
test> db.users.find ({name: "Tom"}, {languages: {$slice : 1}})
[
  {
    _id: ObjectId("62e3c70079a0a7792a9de20c"),
    name: 'Tom',
    age: 32,
    languages: [ 'english' ]
  }
]
test>
```

Зворотна ситуація: нам треба залишити в масиві також один елемент, але не з початку, а з кінця. У цьому випадку необхідно передати в параметр від'ємне значення:

```
db.users.find ({name: "Tom"}, {languages: {$slice : -1}});
```

Тепер у масиві опиниться `"spanish"`, оскільки він перший з кінця в доданому елементі.

Використовуємо відразу два параметри:

```
db.users.find ({name: "Tom"}, {languages: {$slice : [-2, 1]}});
```

Перший параметр говорить почати вибірку елементів з кінця (оскільки від'ємне значення), тобто вибірка йде починаючи з другого елемента з кінця, а другий параметр вказує на кількість елементів масиву, що повертаються. У підсумку в масиві `language` виявиться `"german"`.

Індекси

Під час пошуку документів у невеликих колекціях ми не зазнаємо особливих проблем. Однак коли колекції містять мільйони документів, а нам треба зробити вибірку за певним полем, то пошук потрібних даних може зайняти певний час, який може виявитися критичним для нашого завдання. У цьому випадку нам можуть допомогти індекси.

Індекси дають змогу впорядкувати дані за певним полем, що згодом прискорить пошук. Наприклад, якщо ми у своєму додатку або завданні, як правило, виконуємо пошук за полем `name`, то ми можемо індексувати колекцію за цим полем.

Створення індексу

Для створення індексу застосовується функція `createIndex()`, в яку передається об'єкт із зазначенням полів, для яких створюється індекс. Наприклад, створення індексу за полем `"name"`:

```
db.users.createIndex({"name" : 1})
```

Під час створення індексу консоль поверне нам назву індексу:

```
test> db.users.createIndex({"name" : 1})
name_1
test>
```

Тобто в прикладі вище був створений індекс з ім'ям `"name_1"` за полем `name`. MongoDB дає змогу встановити до 64 індексів на одну колекцію.

Для створення декількох індексів застосовується функція `createIndexes()` - в неї передається масив об'єктів, які встановлюють поля для індексів:

```
db.users.createIndexes([{"name": 1}, {"age": 1}])
```

У цьому випадку створюються два індекси - один для поля `name`, інший для поля `age`.

Видалення індексів

Для видалення індексів застосовується функція `dropIndex()`, в яку передається ім'я індексу. Наприклад, видалимо вище визначений індекс `"name_1"`:

```
db.users.dropIndex("name_1")
```

Налаштування індексів

Якщо ми просто визначимо індекс для колекції, наприклад, `db.users.createIndex({"name" : 1})`, то ми все ще зможемо додавати в колекцію документи з однаковим значенням ключа `name`. Однак, якщо нам потрібно, щоб у колекцію можна було додавати документ з одним і тим самим значенням ключа тільки один раз, ми можемо встановити прапор `unique`:

```
db.users.createIndex({"name" : 1}, {"unique" : true})
```

При цьому під час додавання унікального індексу в колекції не повинно бути документів, у яких для певного поля є однакові значення.

Тепер, якщо ми спробуємо додати в колекцію два документи з одним і тим самим значенням `name`, то ми отримаємо помилку.

Водночас тут є свої тонкощі. Так, документ може не мати ключа `name`. У цьому разі для документа, що додається, автоматично створюється ключ `name` зі значенням `null`. Тому під час додавання другого документа, в якому не визначено ключ `name`, буде викинуто виняток, оскільки ключ `name` зі значенням `null` уже присутній у колекції.

Також можна задати один індекс відразу для двох полів:

```
db.users.createIndex({"name" : 1, "age" : 1})
```

Однак у цьому разі всі документи, що додаються, повинні мати унікальні значення для обох полів.

Крім того, тут є свої обмеження. Наприклад, значення поля, за яким іде індексація, не має бути більшим за 1024 байти.

Керування індексами

Усі індекси бази даних зберігаються в системній колекції `indexes`. Для звернення до неї ми можемо використовувати функцію `getIndexes`, наприклад, щоб вивести всю інформацію про індекси для конкретної колекції:

```
db.users.getIndexes()
```

Ця команда поверне виведення на кшталт такого:

```
test> db.users.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { name: 1 }, name: 'name_1' }
]
test>
```

Як ми бачимо, тут для колекції users (з бд test) визначено 2 індекси: id і name. Поле key використовується для пошуку максимального і мінімального значень, для різних операцій, де треба застосовувати цей індекс. Поле name застосовується як ідентифікатор для операцій адміністрування, наприклад, для видалення індексу.

Агрегатні функції

Кількість елементів у колекції

За допомогою функції `countDocuments()` можна отримати загальну кількість документів у колекції:

```
db.users.countDocuments()
```

Якщо нам треба дізнатися не сукупну кількість документів у колекції, а тільки кількість документів у конкретній вибірці, то можна застосовувати функцію `count()`. Наприклад, підрахуємо кількість документів, у яких `name=Tom`:

```
db.users.find({name: "Tom"}).count()
```

Ба більше, ми можемо створювати ланцюжки функцій, щоб конкретизувати умови підрахунку:

```
db.users.find({name: "Tom"}).skip(2).count(true)
```

Тут треба зазначити, що за замовчуванням функція `count` не використовується з функціями `limit` і `skip`. Щоб їх використовувати, як у прикладі вище, у функцію `count` треба передати булеве значення `true`.

Функція `distinct`

У колекції можуть бути документи, які містять однакові значення для одного або декількох полів. Наприклад, у кількох документах визначено `name: "Tom"`. І нам треба знайти тільки унікальні відмінні значення для одного з полів документа. Для цього ми можемо скористатися функцією `distinct`. Наприклад, нехай у базу даних додано такі документи:

```
db.users.insertOne({"ім'я": "Том", "вік": 38, мови: ["english", "spanish"]})
db.users.insertOne({"ім'я": "Боб", "вік": 41, мови: ["англійська", "французька"]})
db.users.insertOne({"ім'я": "Сем", "вік": 28, мови: ["англійська"]})
db.users.insertOne({"name": "Tom", "age": 22, languages: ["english", "german"]})
```

Виведемо всі унікальні значення за полем "name":

```
test> db.users.distinct("name")
[ 'Bob', 'Sam', 'Tom' ]
test>
```

Функції min і max

Функції min і max встановлюють для певного поля мінімальне значення для потрапляння у вибірку. При цьому ці функції можуть використовувати тільки ті поля, для яких встановлено індекси. Наприклад, візьмемо вище визначену колекцію db.users і визначимо в ній індекс для поля age:

```
db.users.createIndex({"age": 1})
```

Під час виконання функції також треба використовувати функцію hint(), у яку передається індекс. Наприклад, виберемо всі документи, в яких поле age більше 30:

```
test> db.users.find().min({age:30}).hint({age:1})
[
  {
    _id: ObjectId("62e3d63a79a0a7792a9de210"),
    name: 'Tom',
    age: 38,
    languages: [ 'english', 'spanish' ]
  },
  {
    _id: ObjectId("62e3d63a79a0a7792a9de211"),
    name: 'Bob',
    age: 41,
    languages: [ 'english', 'french' ]
  }
]
test>
```

Аналогічним чином працює функція max(), яка встановлює максимальне значення. Наприклад, виберемо документи, де age менше 30:

```
test> db.users.find().max({age:30}).hint({age:1})
[
  {
    _id: ObjectId("62e3d64079a0a7792a9de213"),
    name: 'Tom',
    age: 22,
    languages: [ 'english', 'german' ]
  },
  {
    _id: ObjectId("62e3d63a79a0a7792a9de212"),
    name: 'Sam',
    age: 28,
    languages: [ 'english' ]
  }
]
test>
```

Оператори вибірки

Умовні оператори

Умовні оператори задають умову, якій має відповідати значення поля документа:

- `$eq` (дорівнює)
- `$ne` (не дорівнює)
- `$gt` (більше ніж)
- `$lt` (менше ніж)
- `$gte` (більше або дорівнює)
- `$lte` (менше або дорівнює)
- `$in` визначає масив значень, одне з яких повинно мати поле документа
- `$nin` визначає масив значень, які не повинно мати поле документа

Наприклад, знайдемо всі документи, у яких значення ключа `age` менше 30:

```
db.users.find ({age: {$lt : 30}})
```

Аналогічно буде використання інших операторів порівняння. Наприклад, той самий ключ, тільки більше 30:

```
db.users.find ({age: {$gt : 30}})
```

Зверніть увагу, що порівняння тут проводиться над цілочисельними типами, а не рядками. Якщо ключ age представляє строкові значення, то відповідно треба проводити порівняння над рядками: `db.users.find ({age: {$gt : "30"}})`, однак результат буде тим самим.

Але уявімо ситуацію, коли нам треба знайти всі об'єкти зі значенням поля age більше 30, але менше 50. У цьому випадку ми можемо комбінувати два оператори:

```
db.users.find ({age: {$gt: 30, $lt: 50}})
```

Знайдемо користувачів, вік яких дорівнює 22:

```
db.users.find ({age: {$eq: 22}})
```

По суті це аналогія такого запиту:

```
db.users.find ({age: 22})
```

Зворотна операція - знайдемо користувачів, вік яких НЕ дорівнює 22:

```
db.users.find ({age: {$ne: 22}})
```

Оператор `$in` визначає масив можливих виразів і шукає ті ключі, значення яких є в масиві:

```
db.users.find ({age: {$in : [22, 32]}})
```

Протилежним чином діє оператор `$nin` - він визначає масив можливих виразів і шукає ті ключі, значення яких відсутні в цьому масиві:

```
db.users.find ({age: {$nin : [22, 32]}})
```

Логічні оператори

Логічні оператори виконуються над умовами вибірки:

- `$or`: з'єднує дві умови, і документ має відповідати одній із цих умов

- `$and`: з'єднує дві умови, і документ має відповідати обом умовам
- `$not`: документ повинен НЕ відповідати умові
- `$nor`: з'єднує дві умови, і документ повинен НЕ відповідати обом умовам

Оператор `$or`

Оператор `$or` представляє логічну операцію АБО і визначає набір пар ключ-значення, які повинні бути в документі. І якщо документ має хоч одну таку пару ключ-значення, то він відповідає даному запиту і витягується з бд:

```
db.users.find ({ $or : [{name: "Tom"}, {age: 22}] })
```

Цей вираз поверне нам усі документи, в яких або `name=Tom`, або `age=22`.

Інший приклад поверне нам усі документи, в яких `name=Tom`, а `age` дорівнює або 22, або серед значень `languages` є "german":

```
db.users.find ({name: "Tom", $or : [{age: 22}, {languages: "german"}] })
```

У підвиразах `or` можна застосовувати умовні оператори:

```
db.users.find ({ $or : [{name: "Tom"}, {age: {$gte: 30}}] })
```

У цьому випадку ми вибираємо всі документи, де `name="Tom"` або поле `age` має значення від 30 і вище.

Оператор `$and`

Оператор `$and` представляє логічну операцію І (логічне множення) і визначає набір критеріїв, яким обов'язково має відповідати документ. На відміну від оператора `$or` документ має відповідати всім зазначеним критеріям. Наприклад:

```
db.users.find ({ $and : [{name: "Tom"}, {age: 22}] })
```

Тут обрані документи обов'язково повинні мати ім'я `Tom` і вік 22 - обидві ці ознаки.

Пошук за масивами

Ряд операторів призначені для роботи з масивами:

- `$all`: визначає набір значень, які повинні бути в масиві

- `$size`: визначає кількість елементів, які повинні бути в масиві
- `$elemMatch`: визначає умову, яким повинні відповідати елементи в масиві

\$all

Оператор `$all` визначає масив можливих виразів і вимагає, щоб документи мали весь визначуваний набір виразів. Відповідно він застосовується для пошуку за масивом. Наприклад, у документах є масив `languages`, що зберігає іноземні мови, якими розмовляє користувач. І щоб знайти всіх людей, які розмовляють одночасно і англійською, і французькою, ми можемо використовувати такий вираз:

```
db.users.find ({languages: {$all : ["english", "french"]}})
```

Оператор \$elemMatch

Оператор `$elemMatch` дає змогу вибрати документи, у яких масиви містять елементи, що потрапляють під певні умови. Наприклад, нехай у базі даних буде колекція, яка містить оцінки користувачів за певними курсами. Додамо кілька документів:

```
db.grades.insertMany([{$student: "Tom", courses:[{$name: "Java", grade: 5}, {$name: "MongoDB", grade: 4}]},
{$student: "Alice", courses:[{$name: "C++", grade: 3}, {$name: "MongoDB", grade: 5}]}])
```

Кожен документ має масив `courses`, який зі свого боку складається з вкладених документів.

Тепер знайдемо студентів, які для курсу MongoDB мають оцінку вище 4:

```
db.grades.find({courses: {$elemMatch: {$name: "MongoDB", grade: {$gt: 4}}}})
```

Оператор \$size

Оператор `$size` використовується для знаходження документів, у яких масиви мають число елементів, що дорівнює значенню `$size`. Наприклад, витягнемо всі документи, в яких у масиві `languages` два елементи:

```
db.users.find ({languages: {$size:2}})
```

Такий запит відповідатиме, наприклад, такому документу:

```
{"name": "Tom", "age": 32, "languages": ["english", "german"]}
```

Оператор \$exists

Оператор \$exists дає змогу витягти тільки ті документи, в яких певний ключ присутній або відсутній. Наприклад, повернемо всі документи, у яких є ключ company:

```
db.users.find ({company: {$exists:true}})
```

Якщо ми вкажемо в оператора \$exists як параметр false, то запит поверне нам тільки ті документи, в яких не визначено ключ company.

Оператор \$type

Оператор \$type витягує тільки ті документи, в яких певний ключ має значення певного типу, наприклад, рядок або число:

```
db.users.find ({age: {$type: "string"}})
db.users.find ({age: {$type: "number"}})
```

Оператор \$regex

Оператор \$regex задає регулярний вираз, якому має відповідати значення поля. Наприклад, нехай поле name обов'язково має букву "b":

```
db.users.find ({name: {$regex: "b"}})
```

Важливо розуміти, що \$regex приймає не просто рядки, а саме регулярні вирази, наприклад: name: {\$regex: "om\$"} - значення name має закінчуватися на "om".

Оновлення даних

Як і інші системи управління базами даних MongoDB надає можливість оновлення даних. Для цього доступно низку функцій.

replaceOne

Якщо нам треба повністю замінити один документ іншим, також може використовуватися функція replaceOne:

```
db.collection.replaceOne(filter, update, options)
```

- filter: приймає запит на вибірку документа, який треба оновити

- **update:** представляє новий документ, який замінить старий під час оновлення
- **options:** визначає додаткові параметри під час оновлення документів, основним з яких є параметр **upsert**.

Якщо параметр **upsert** має значення **true**, то **mongodb** буде оновлювати документ, якщо він знайдений, і створювати новий, якщо такого документа немає. Якщо ж він має значення **false**, то **mongodb** не буде створювати новий документ, якщо запит на вибірку не знайде жодного документа.

Наприклад:

```
db.users.replaceOne({name: "Bob"}, {name: "Bob", age: 25})
```

У цьому випадку знаходимо документ, у якому **name = "Bob"**, і замінюємо його документом **{name: "Bob", age: 25}**

Після виконання операції консоль поверне на результат оновлення:

```
test> db.users.replaceOne({name: "Bob"}, {name: "Bob", age: 25})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

В отриманому результаті параметр **matchedCount** вказує на кількість документів, які відповідають запиту. А параметр **modifiedCount** вказує на кількість змінених документів. Тобто в даному випадку відповідає запиту 1 документ і він же був змінений.

updateOne і updateMany

Часто не потрібно оновлювати весь документ, а тільки значення однієї або декількох його властивостей. Для цього застосовуються функції **updateOne()** (оновлює тільки один документ) і **updateMany()** (оновлює безліч документів).

Для оновлення окремих полів у цих функціях застосовується оператор **\$set**. Якщо документ не містить оновлюваного поля, то воно створюється.

```
db.users.updateOne({name : "Tom", age: 22}, {$set: {age : 28}})
```

Тут ми шукаємо документ з **name="Tom"** і **age=22** і встановлюємо для його властивості **age** значення **28**

Якщо необхідно оновити всі документи, що відповідають деякому критерію, то застосовується функція `updateMany()`:

```
db.users.updateMany({name: "Tom"}, {$set: {name: "Tomas"}})
```

Якщо оновлюваного поля в документі немає, то воно додається:

```
db.users.updateOne({name : "Tom", age: 28}, {$set: {salary : 300}})
```

Якщо треба оновити значення декількох полів, то вони передаються оператору `$set` через кому:

```
db.users.updateOne({name : "Tom"}, {$set: {name: "Tomas", age : 25}})
```

Для простого збільшення значення числового поля на певну кількість одиниць застосовується оператор `$inc`. Якщо документ не містить оновлюваного поля, то воно створюється. Цей оператор застосовується тільки до числових значень.

```
db.users.updateOne({name: "Tom"}, {$inc: {age:2}})
```

Видалення поля

Для видалення окремого ключа використовується оператор `$unset`:

```
db.users.updateOne({name : "Tom"}, {$unset: {salary: 1}})
```

Якщо раптом подібного ключа в документі не існує, то оператор не має жодного впливу. Також можна видаляти відразу кілька полів:

```
db.users.updateOne({name : "Tom"}, {$unset: {salary: 1, age: 1}})
```

Оператор \$push

Оператор `$push` дає змогу додати ще одне значення до вже наявного. Наприклад, якщо ключ як значення зберігає масив:

```
db.users.updateOne({name : "Tom"}, {$push: {languages: "russian"}})
```

Вище використовувалася функція `updateOne`, але цей оператор також застосовується і в функції `updateMany`

```
db.users.updateMany({name : "Tom"}, {$push: {languages: "russian"}})
```

Якщо ключ, для якого ми хочемо додати значення, не представляє масив, то ми отримаємо помилку `Cannot apply $push/$pushAll modifier to non-array`.

Використовуючи оператор `$each`, можна додати відразу кілька значень:

```
db.users.updateOne({name : "Tom"}, {$push: {languages: {$each: ["russian", "spanish", "italian"]}}})
```

Ще пара операторів дає змогу налаштувати вставку. Оператор `$position` задає позицію в масиві для вставки елементів, а оператор `$slice` вказує, скільки елементів залишити в масиві після вставки.

```
db.users.updateOne({name : "Tom"}, {$push: {languages: {$each: ["німецька", "іспанська", "італійська"]}, $position: 1, $slice: 5}}})
```

У цьому разі елементи `["german", "spanish", "italian"]` будуть вставлятися в масив `languages` з 1-го індексу, і після вставки, в масиві залишаться тільки 5 перших елементів.

Оператор `$addToSet`

Оператор `$addToSet` подібно до оператора `$push` додає об'єкти в масив. Відмінність полягає в тому, що `$addToSet` додає дані, якщо їх ще немає в масиві (при додаванні через `$push` дані дублюються, якщо додаються елементи, які вже є в масиві):

```
db.users.updateOne({name : "Tom"}, {$addToSet: {languages: "russian"}})
```

Видалення елемента з масиву

Оператор `$pop` дозволяє видаляти елемент із масиву:

```
db.users.updateOne({name : "Tom"}, {$pop: {languages: 1}})
```

Вказуючи для ключа `languages` значення `1`, ми видаляємо перший елемент з кінця. Щоб видалити перший елемент спочатку масиву, треба передати від'ємне значення:

```
db.users.updateOne({name: "Tom"}, {$pop: {languages: -1}})
```

Дещо іншу дію передбачає оператор `$pull`. Він видаляє кожне входження елемента в масив. Наприклад, через оператор `$push` ми можемо додати одне й те саме значення в масив кілька разів. І тепер за допомогою `$pull` видалимо його:

```
db.users.updateOne({name: "Tom"}, {$pull: {languages: "english"}})
```

А якщо ми хочемо видалити не одне значення, а відразу кілька, тоді ми можемо застосувати оператор `$pullAll`:

```
db.users.updateOne({name : "Tom"}, {$pullAll: {languages: ["english", "german", "french"]}})
```

Видалення даних

Для видалення документів у MongoDB передбачено функції `deleteOne()` - видаляє один документ і `deleteMany()` - дає змогу видалити кілька документів. Як параметр у ці функції передається фільтр документів, що видаляються.

Наприклад, видалимо документ, у якому `name="Tom"`:

```
db.users.deleteOne({name : "Tom"})
```

У підсумку перший знайдений документ з `name=Tom` буде видалено. Після видалення консоль відображає нам об'єкт, в якому параметр `deletedCount` вказує на кількість видалених документів:

```
test> db.users.deleteOne({name : "Tom"})  
{ acknowledged: true, deletedCount: 1 }
```

Для видалення всіх документів, які відповідають фільтру, застосовується функція `deleteMany()`:

```
db.users.deleteMany({name : "Tom"})
```

Причому, як і у випадку з `find`, ми можемо задавати умови вибірки для видалення різними способами (у вигляді регулярних виразів, у вигляді умовних конструкцій тощо):

```
db.users.deleteOne({name : /^T\w+/i})  
db.users.deleteOne({age: {$lt : 30}})
```

Щоб видалити разом усі документи з колекції, треба залишити порожнім параметр запиту:

```
db.users.deleteMany({})
```

Видалення колекцій і баз даних

Ми можемо видаляти не тільки документи, а й колекції та бази даних. Для видалення колекцій використовується функція `drop`:

```
db.users.drop()
```

І якщо видалення колекції пройде успішно, то консоль виведе:

```
true
```

Щоб видалити всю базу даних, треба скористатися функцією `dropDatabase()`:

```
db.dropDatabase()
```

Установка посилань в БД

У реляційних базах даних можна встановлювати зовнішні ключі, коли поля з однієї таблиці посилаються на поля в іншій таблиці. У MongoDB також можна встановлювати посилання. Далі ми розглянемо встановлення посилань між колекцією `users`, яка зберігає користувачів, і колекцією `companies`, яка зберігає список компаній, у яких працюють користувачі.

Ручне встановлення посилань

Ручне встановлення посилань зводиться до присвоєння значення поля `_id` одного документа полю іншого документа. Припустимо, у нас можуть бути колекції, що представляють компанії та працівників, які працюють у цих компаніях. Отже, спочатку додамо в колекцію `companies` документ, що представляє компанію:

```
db.companies.insertOne({"_id": "microsoft", year: 1974})
```

Тепер додамо в колекцію `users` документ, який представляє працівника. У цьому документі буде поле `company`, що представляє компанію, де працює працівник. І дуже важливо, що як значення для цього поля ми встановлюємо не об'єкт `company`, а значення ключа `_id` доданого вище документа:

```
db.users.insertOne({name: "Tom", age: 28, company: "microsoft"})
```

Тепер отримаємо документ із колекції `users`:

```
user = db.users.findOne()
```

У цьому разі маєтись на увазі, що вище доданий елемент буде єдиним у колекції. Інакше треба зробити вибірку за `_id` останнього доданого документа.

Після цього консоль виводить отриманий документ. І тепер знайдемо посилання на його компанію в колекції `companies`:

```
db.companies.findOne({_id: user.company})
```

І якщо документ із таким ідентифікатором виявлено, він відображається на консолі:

```
test> db.companies.insertOne({"_id" : "microsoft", year: 1974})
{ acknowledged: true, insertedId: 'microsoft' }
test> db.users.insertOne({name: "Tom", age: 28, company: "microsoft"})
{
  acknowledged: true,
  insertedId: ObjectId("62e427069fcbf14521ea012a")
}
test> user = db.users.findOne()
{
  _id: ObjectId("62e427069fcbf14521ea012a"),
  name: 'Tom',
  age: 28,
  company: 'microsoft'
}
test> db.companies.findOne({_id: user.company})
{ _id: 'microsoft', year: 1974 }
test>
```

Автоматичне зв'язування

Використовуючи функціональність DBRef, ми можемо встановити автоматичне зв'язування між документами. Подивимось на прикладі застосування цієї функціональності. Спочатку додамо новий документ у колекцію companies:

```
google = db.companies.insertOne({ім'я: "google", рік: 1998})
```

Зверніть увагу, що ми отримуємо результат додавання у змінну google. При додаванні нового документа генерує `_id`, який ми можемо отримати за допомогою властивості `insertedId` результату функції.

Тепер створимо новий документ для колекції users, у якого ключ company зв'яжемо зі щойно доданим документом apple:

```
sam = ({name: "Sam", age: 25, company: { "$ref" : "companies", "$id" :
google.insertedId}})
db.users.insertOne(sam)
```

І ми можемо протестувати:

```
db.companies.findOne({_id: sam.company.$id})
```



```

test> google = db.companies.insertOne({name : "google", year: 1998})
{
  acknowledged: true,
  insertedId: ObjectId("62e427c29fcfbf14521ea012b")
}
test> sam = ({name: "Sam", age: 25, company: { "$ref" : "companies", "$id" : google.insertedId}})
{
  name: 'Sam',
  age: 25,
  company: { '$ref': 'companies', '$id': ObjectId("62e427c29fcfbf14521ea012b") }
}
test> db.users.insertOne(sam)
{
  acknowledged: true,
  insertedId: ObjectId("62e427dc9fcfbf14521ea012c")
}
test> db.companies.findOne({_id: sam.company.$id})
{
  _id: ObjectId("62e427c29fcfbf14521ea012b"),
  name: 'google',
  year: 1998
}
test>

```

Подивившись на прикладі, тепер розберемо організацію посилань між документами. Для зв'язування з документом google використовувався такий вираз { "\$ref" : "companies", "\$id" : google.insertedId}. Формальний синтаксис DBRef такий:

{ "\$ref" : назва_колекції, "\$id": значення [, "\$db" : назва_бд]}

Перший параметр \$ref вказує на колекцію, де зберігається пов'язаний документ. Другий параметр вказує на значення, яке і буде представляти щось типу зовнішнього ключа. Третій необов'язковий параметр вказує на базу даних.

Під час тестування як запит на вибірку вказується вираз _id: sam.company.\$id. Оскільки sam.company представляє тепер об'єкт { "\$ref" : "companies", "\$id" : google.insertedId}, то нам треба конкретизувати параметр sam.company.\$id .

Керування колекцією

Явне створення колекції

У попередніх темах колекція створювалася неявно автоматично під час додавання в неї перших даних. Але ми також можемо створити її явним чином, застосувавши

метод `db.createCollection(name, options)`, де `name` - назва колекції, а `options` - необов'язковий об'єкт із додатковими налаштуваннями ініціалізації. Наприклад:

```
db.createCollection("accounts")
{"ok" : 1}
```

Таким чином, створюється колекція `accounts`.

Перейменування колекції

У процесі роботи, можливо, буде потрібно змінити назву колекції. Наприклад, якщо під час першого додавання даних у її назві була помилка. І щоб не видаляти і потім перестворювати колекцію, слід використовувати функцію `renameCollection`:

```
db.users.renameCollection("нова_назва")
```

І якщо перейменування пройде вдало, то консоль відобразить:

```
{"ok" : 1}
```

Обмежені колекції

Коли ми відправляємо запит до бд на вибірку, то MongoDB повертає нам документи в тому порядку, як правило, в якому вони були додані. Однак такий порядок не завжди гарантується, оскільки дані можуть бути видалені, переміщені, змінені. Тому в MongoDB існує поняття обмеженої колекції (`capped collection`). Подібна колекція гарантує, що документи розташовуватимуться в тому самому порядку, в якому їх додавали в колекцію. Обмежені колекції мають фіксований розмір. І коли в колекції вже немає місця, найстаріші документи видаляються, і в кінець додаються нові дані.

На відміну від звичайних колекцій обмежені ми можемо задати явним чином.

Наприклад, створимо обмежену колекцію з назвою `profiles` і задамо для неї розмір у 9500 байт:

```
db.createCollection("profiles", {capped:true, size:9500})
```

І після вдалого створення колекції консоль виведе:

```
{"ok":1}
```

Також можна обмежити кількість документів у колекції, вказавши її в параметрі `max`:

```
> db.createCollection("profiles", {capped:true, size:9500, max: 150})
```

Однак за такого способу створення колекції слід враховувати, що якщо все місце під колекцію заповнене (наприклад, виділені нами 9500 байтів), а кількість документів ще не досягла максимуму, в даному випадку 150, то в цьому випадку під час додавання нового документа найстаріший документ видалятиметься, а на його місце вставлятиметься новий документ.

Під час оновлення документів у таких колекціях треба пам'ятати, що документи не повинні зростати в розмірах, інакше оновлення не вдасться зробити.

Також не можна видаляти документи з подібних колекцій, можна тільки видалити всю колекцію.

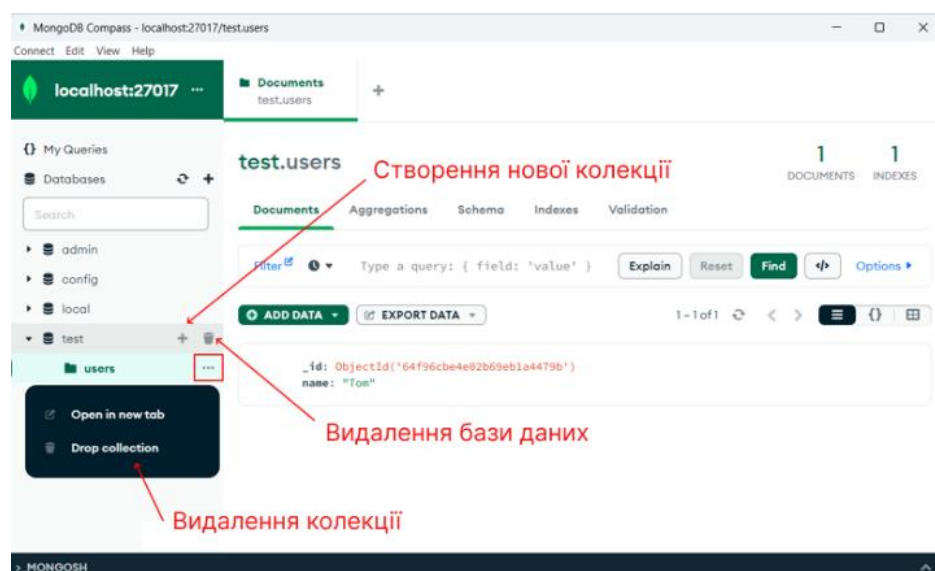
Підколекції

Для спрощення організації даних у колекціях ми можемо використовувати підколекції. Наприклад, дані з колекції `users` треба розмежувати на профілі та облікові дані. І ми можемо використовувати створити колекції `db.users.profiles` і `db.users.accounts`. При цьому вони не будуть ніяк пов'язані з колекцією `users`. Тобто в підсумку будуть три різні колекції, проте в плані логічної організації зберігання даних, можливо, для когось так буде простіше.

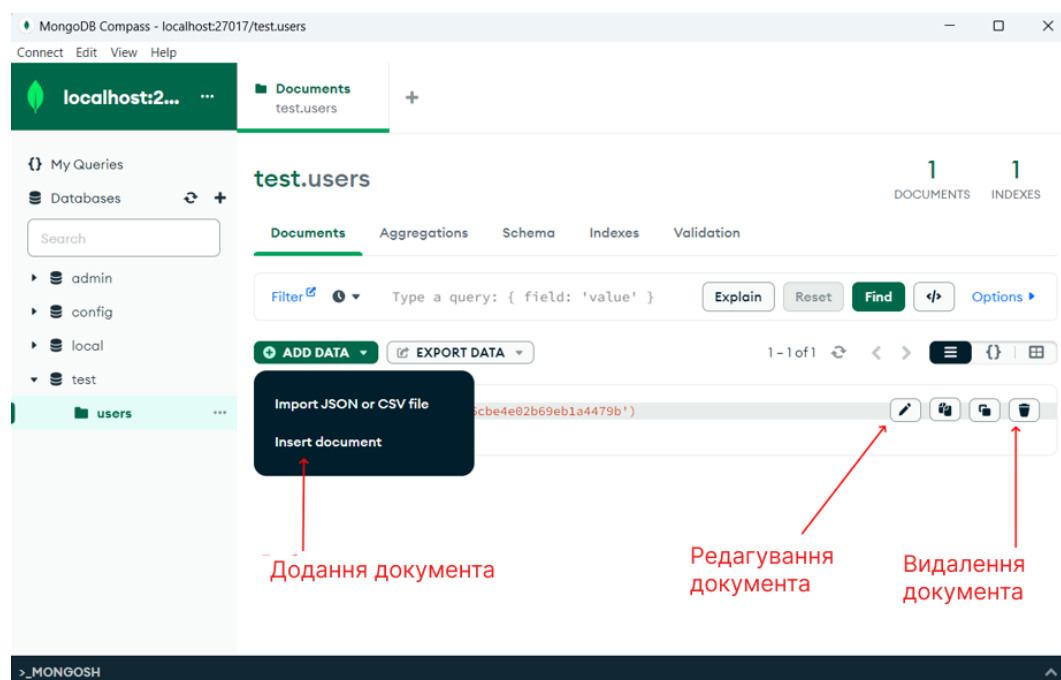
Робота з даними в MongoDB Compass

У попередніх темах розглядалася робота з сервером MongoDB через консольну оболонку `mongosh`. Але також ми можемо працювати з даними через графічний клієнт MongoDB Compass. Цей графічний клієнт доволі простий та інтуїтивно зрозумілий у використанні, а графічне представлення даних для когось може бути простішим для розуміння. Відповідно комусь, можливо, через графічний клієнт буде простіше і зручніше працювати.

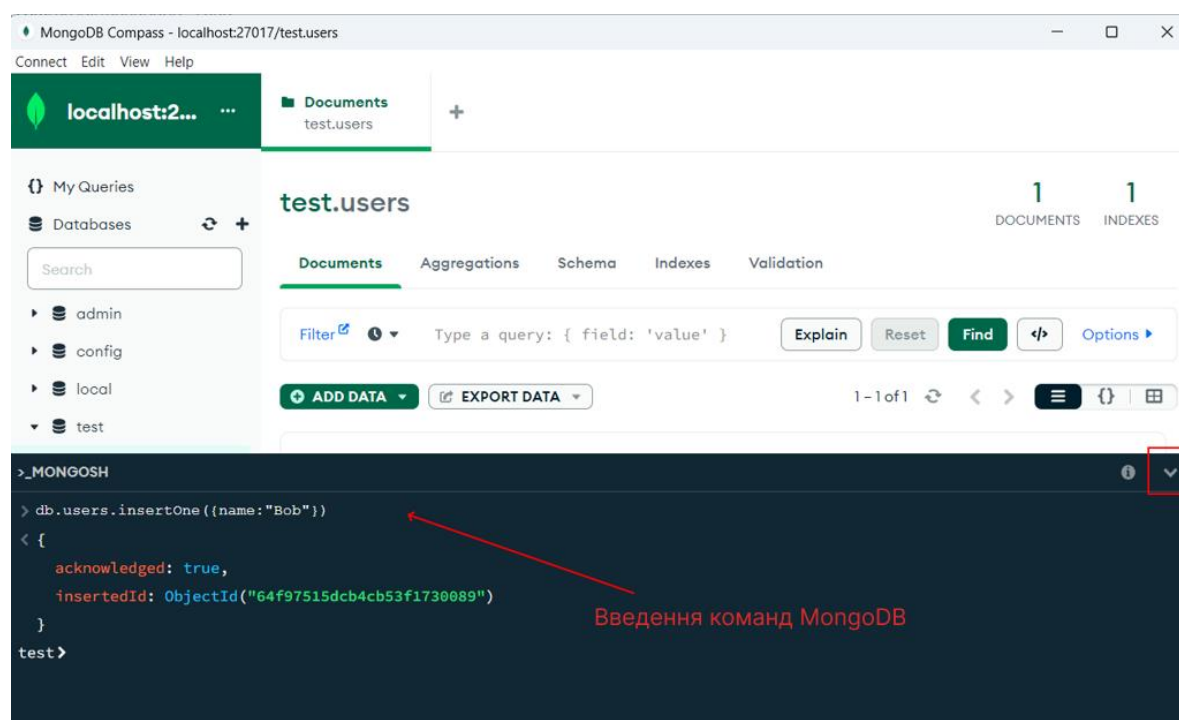
Для базових операцій, як то: створення/видалення колекцій, додавання, перегляд, зміни та видалення документів є відповідні елементи графічного інтерфейсу:



Під час вибору певної колекції та відображення її даних над списком даних є кнопка для додавання даних, у тому числі для імпорту із зовнішнього файлу. А навпроти кожного документа є опції для його перегляду, зміни та видалення:



Але незважаючи на наявність графічних можливостей для управління даними, вони можуть виявитися недостатніми для якихось складніших сценаріїв роботи з даними. І для цього в MongoDB Compass є вбудований консольний клієнт Mongosh, в якому можна вводити майже всі ті самі команди для роботи з даними, що і в консольній оболонці mongosh:



Але також варто зазначити, що консольний клієнт у Compass дещо спрощує написання запитів. Так, він має функцію автодоповнення, яка дозволяє нам запропонувати варіанти доступних функцій.

Питання:

- 12.1 Що таке MongoDB?
- 12.2 Яким чином MongoDB зберігає дані?
- 12.3 Що таке документ в MongoDB?
- 12.4 Що таке колекція в MongoDB?
- 12.5 Як створити базу даних в MongoDB?
- 12.6 Як додати документ до колекції в MongoDB?
- 12.7 Як зробити вибірку даних з MongoDB?
- 12.8 Як оновити дані в MongoDB?
- 12.9 Як видалити дані в MongoDB?
- 12.10 Як створити індекси в MongoDB?

Список рекомендованої літератури

1. Андрій Юліанович Берко, Олег Михайлович Верес, Володимир Володимирович Пасічник. Системи баз даних та знань. Кн. 1. Організація баз даних та знань. – Львів: Магнолія, 2016. – 438 с.
2. Андрій Юліанович Берко, Олег Михайлович Верес, Володимир Володимирович Пасічник. Системи баз даних та знань. Кн. 2. Системи управління базами даних та знань. – Львів: Магнолія, 2016. – 583 с.
3. Л. С. Глоба, М.Ю. Терновой., О.С. Штогріна Методичні вказівки щодо виконання лабораторних робіт з кредитного модулю «Інформаційне забезпечення телекомунікаційних мереж. Програмне забезпечення створення баз даних» [Електронна версія].
4. Connolly, Thomas M.; Begg, Carolyn E. (2014). Database Systems – A Practical Approach to Design Implementation and Management (6th ed.). Pearson. ISBN 978-1292061184.
5. Leavitt, Neal (2010). "Will NoSQL Databases Live Up to Their Promise?" (PDF). IEEE Computer. 43 (2): 12–14. doi:10.1109/MC.2010.58. S2CID 26876882
6. Strauch, Christof (2012). "NoSQL Databases" (PDF) (<http://www.christof-strauch.de/nosql dbs.pdf>).
7. Luc Perkins, Eric Redmond, Jim R. Wilson. Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement, 2nd ed. – Raleigh: Pragmatic Bookshelf, 2018. – 360 p.
8. Aaron Ploetz, Devram Kandhare, Sudarshan Kadambi, Xun Wu. Seven NoSQL Databases in a Week: Get up and running with the fundamentals and functionalities of seven of the most popular NoSQL databases. – Birmingham: Packt Publishing, 2018. – 308 p.
9. A. Siddiqua et al., “A survey of big data management: Taxonomy and state-of-the-art,” J. Netw. Comput. Appl., vol. 71, pp. 151–166, 2016.
10. C. Strozzi, “NoSQL—A relational Database management system. 2007–2010,” WWW page Ar-tic. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page. Accessed, vol. 10, 2013.
11. Y. Mansouri, V. Prokhorenko, and M. A. Babar, “An automated implementation of hybrid cloud for performance evaluation of distributed Databases,” J. Netw. Comput. Appl., vol. 167, 2020, doi: 10.1016/j.jnca.2020.102740.
12. Big Data and Artificial Intelligence in Digital Finance / Soldatos, John (editor). Kyriazis, Dimosthenis (editor) ; Springer Nature, Cham, 2022, <https://directory.doabooks.org/handle/20.500.12854/81702>
13. Knowledge Graphs and Big Data Processing / Janev, Valentina (editor) Graux, Damien (editor) Jabeen, Hajira (editor) Sallinger, Emanuel (editor) ; Springer Nature, 2020, <https://directory.doabooks.org/handle/20.500.12854/28210>
14. Linked Open Data / Okoye, Kingsley (editor) ; IntechOpen, 2020, <https://directory.doabooks.org/handle/20.500.12854/67759>
15. Programming Persistent Memory / Scargall, Steve; Springer Nature, Berkeley, CA, 2020, <https://directory.doabooks.org/handle/20.500.12854/26322>
16. Qu'est-ce qu'une archive du Web? / Musiani, Francesca. Paloque-Berges, Camille. Schafer, Valérie. Thierry, Benjamin ; OpenEdition Press, 2019-02-28, <https://directory.doabooks.org/handle/20.500.12854/39173>

17. Real-time Linked Dataspaces / Curry, Edward; Springer Nature, 2020, <https://directory.doabooks.org/handle/20.500.12854/31469>
18. Technologies and Applications for Big Data Value / Curry, Edward (editor) [et al] ; Springer Nature, 2022, <https://directory.doabooks.org/handle/20.500.12854/81694>
19. The Elements of Big Data Value / Curry, Edward (editor) [et al] ; Springer Nature, 2021, <https://directory.doabooks.org/handle/20.500.12854/71319>