Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

М. В. Коломицев, С. О. Носок

БАЗИ ДАНИХ ТА ІНФОРМАЦІЙНІ СИСТЕМИ

Підручник

Затверджено Вченою радою КПІ ім. Ігоря Сікорського як підручник для здобувачів ступеня бакалавра за спеціальністю 125 Кібербезпека та захист інформації

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025

УДК 004.65 (075.8)

Автори: *Коломицев Михайло Володимирович*, канд. техн. наук, доц.
Носок Світлана Олександрівна, канд. техн. наук, доц.

Рецензенти: *Шелестов Андрій Юрійович*, д.т.н., проф., г.н.с., Інститут космічних досліджень НАН України
Поята Сергій Русланович, магістр зі спеціальності «Захист інформації в комп'ютерних системах та мережах», заст. директора ТОВ «ІССП СЕРВІС»

Відповідальний редактор *Литвинова Тетяна Василівна,* канд. техн. наук, доц.

Гриф надано Вченою радою КПІ ім. Ігоря Сікорського
(протокол № 11 від 10.11.2025 р.)

Бази даних та інформаційні системи [Електронний ресурс] : підруч. для здобувачів ступеня бакалавра за спец. 125 Кібербезпека та захист інформації / М. В. Коломицев, С. О. Носок ; КПІ ім. Ігоря Сікорського. – Електрон. текст. дані (1 файл: 6,07 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2025. – 294 с.

У підручнику міститься систематичний виклад дисципліни «Бази даних та інформаційні системи», розглядаються основні поняття і технологія систем баз даних (СБД), класифікація СБД, компоненти СБД, їх взаємодія. Детально розглянуто ключові етапи процесу проєктування баз даних – концептуальний, логічний і фізичний дизайн бази даних. Основну увагу приділено проєктуванню реляційних баз даних (РБД), реляційній моделі, мовному середовищу роботи з РБД. Викладено основні відомості про архітектуру інформаційних систем, їхні особливості.

Розглянуто особливості сучасних напрямів розвитку баз даних – бази даних у ВЕБ, бази даних NoSQL. Окремо розглянуто питання забезпечення безпеки БД.

Мета підручника – формування професійних компетентностей у галузі систем баз даних. Підручник призначений для здобувачів ступеня бакалавра спеціальності 125 Кібербезпека та захист інформації.

УДК 004.65 (075.8)

Реєстр. № П XX/XX-XXX. Обсяг 12,25 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© М. В. Коломицев, С. О. Носок, 2025
© КПІ ім. Ігоря Сікорського, 2025

ЗМІСТ

ВСТУП 7
1. ВСТУП ДО СИСТЕМ БАЗ ДАНИХ 10
1.1. Дані, інформація, інформаційні системи 10
1.2. Класифікація інформаційних систем 11
1.3. Типи інформаційних систем за характером інформації, що обробляється 16
1.4. Підхід до управління даними з використанням БД 18
1.5. Класифікація Систем баз даних 20
1.6. Функції СКБД 21
1.7. Функціональні компоненти СКБД 22
1.8. Архітектура баз даних ANSI / SPARC 24
1.9. Загальні властивості моделей даних 27
1.10. Класифікація моделей даних 29
1.11. Еволюція моделей даних 30
Контрольні запитання 32
Тестові завдання 32
2. КОНЦЕПТУАЛЬНЕ ПРОЄКТУВАННЯ БАЗ ДАНИХ 35
2.1. Побудова функціональної моделі 35
2.2. Концептуальне проєктування та концептуальна модель 39
2.2.1. Завдання концептуального проєктування 39
2.2.2. Створення ER-діаграми 40
2.2.3. Створення діаграми «тільки сутності» (Entity-Only) 40
2.2.4. Визначення атрибутів сутностей 44
2.2.5. Створення зв’язку між сутностями 51
Контрольні запитання 57
Тестові завдання 58
3. РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ 61
3.1. Загальні поняття реляційного підходу до організації даних 61
3.2. Структурна частина РМД 62
3.3. Цілісна частина реляційної моделі 67
3.3.1. Обмеження домену 68
3.3.2. Цілісність сутностей 68
3.3.3. Цілісність посилань 71
3.3.4. Null-значення і тризначна логіка 73
3.3.5. Основні правила перетворення моделі «сутність – зв’язок» в реляційну 74
3.4. Маніпуляційна частина реляційної моделі 76
3.4.1. Об’єднання відношень 78
3.4.2. Перетин відношень 78
3.4.3. Різниця відношень 79
3.4.4. Розширений декартів добуток 80
3.4.5. Вибірка 81
3.4.6. Проекція 81
3.4.7. Умовне з’єднання 82
3.4.8. Ділення відношень 86
3.4.9. Перетворення операцій реляційної алгебри 87

Контрольні запитання ... **88**
Тестові завдання ... **89**
4. ЛОГІЧНЕ ПРОЄКТУВАННЯ БАЗ ДАНИХ ... 92
4.1. Засоби логічного проєктування реляційної бази даних ... **92**
4.2. Надмірність даних і аномалії поновлення ... **92**
4.3. Функціональні залежності ... **94**
4.3.1. Визначення функціональної залежності ... 94
4.3.2. Типи функціональних залежностей ... 95
4.4. Аксіоми виведення (Армстронга) ... **97**
4.5. Послідовність нормальних форм ... **99**
4.5.1. Декомпозиція відношень ... 100
4.5.2. Перша нормальна форма (1NF) ... 101
4.5.3. Друга нормальна форма (2NF) ... 103
4.5.4. Третя нормальна форма (3NF) ... 104
4.5.5. Нормальна форма Бойса–Кодда (Boyce–Codd normal form, BCNF) ... 106
4.5.6. Четверта нормальна форма (4NF) ... 107
4.5.7. Денормалізація схеми відношень ... 109
Контрольні запитання ... **110**
Тестові завдання ... **110**
5. МОВА SQL ... 113
5.1. Загальна інформація ... **113**
5.2. Управління базами даних за допомогою SQL ... **114**
5.2.1. Синтаксис мови SQL ... 114
5.2.2. Оператори ... 115
5.2.3. Типи даних в SQL ... 118
5.3. Команди DDL ... **119**
5.3.1. Обмеження команд ... 119
5.3.2. Команди створення модифікації і видалення таблиць ... 120
5.3.3. Зміна таблиці після того, як вона була створена ... 122
5.4. Читання даних ... **123**
5.4.1. Функції і оператори ... 124
5.4.2. Агрегатні функції ... 129
5.4.3. Багатотабличні запити ... 132
5.4.4. Вкладені запити ... 137
5.4.5. Прості вкладені запити ... 138
5.4.6. Корельовані вкладені запити ... 141
5.4.7. Загальний табличний вираз (Common Table Expression *CTE*) ... 142
5.4.8. З’єднання, підзапити, CTE ... 145
5.5. Подання ... **146**
5.6. Процедури і тригери ... **149**
5.6.1. Збережені процедури ... 149
5.6.2. Тригери ... 151
5.7. Операції зміни даних ... **154**
5.7.1. Команда INSERT ... 154
5.7.2. Команда UPDATE ... 155
5.7.3. Команда DELETE ... 156
Контрольні запитання ... **157**

Тестові завдання **157**
6. АНАЛІТИЧНА ОБРОБКА ДАНИХ ЗАСОБАМИ SQL 161
6.1. Реалізація базових операцій над множинами **161**
6.2. Логічні умови в запитах SQL **163**
6.3. Ранжування даних **165**
6.4. Проміжні підсумки в агрегованих даних **167**
6.5. Віконні функції **169**
Контрольні запитання **172**
Тестові завдання **173**
7. ФІЗИЧНА РЕАЛІЗАЦІЯ БАЗИ ДАНИХ 177
7.1. Організація пам’яті **177**
7.2. Організація вторинної пам’яті **178**
7.2.1. Роль диспетчера записів 178
7.2.2. Реалізація файлу записів 182
7.2.3. Організація записів у файлах 185
7.3. Індексування **187**
7.3.1. Загальні відомості 187
7.3.2. Впорядковані індекси 190
7.3.3. B+ – деревоподібні індексні файли 193
7.3.4. Індексування на основі хешування 195
7.3.5. Вартість індексування 199
7.3.6. Рекомендації щодо створення індексів 200
Контрольні запитання **201**
Тестові завдання **202**
8. УПРАВЛІННЯ ТРАНЗАКЦІЯМИ 205
8.1. Поняття транзакцій **205**
8.2. Базові властивості транзакцій **207**
8.3. Проблеми одночасного доступу до даних **209**
8.4. Паралельне виконання транзакцій **212**
8.4.1. Поняття серіалізації транзакцій 212
8.4.2. Рівні ізоляції транзакцій 214
8.5. Механізми управління паралелізмом **215**
8.5.1. Протоколи на основі блокування 215
8.5.2. Тупикове блокування (deadlock) 218
8.5.3. Інші протоколи контролю паралелізму 221
8.6. Відновлення після збою **222**
8.6.1. Базові компоненти відновлення 222
8.6.2. Відновлення транзакцій 224
8.6.3. Відновлення після збою носія 225
8.7. Забезпечення доступності даних **228**
8.7.1. Підтримання системи в робочому стані 228
8.7.2. Планування резервного копіювання і відновлення 229
8.7.3. Висока доступність для критично важливих систем 230
Контрольні запитання **231**
Тестові завдання **232**
9. BIG DATA І БАЗИ ДАНИХ NOSQL 235

9.1. Великі дані (Big Data) ... **235**
9.2. Hadoop ... **238**
9.2.1. Розподілена файлова система HDFS ... 238
9.2.2. MapReduce ... 240
9.3. Засоби масштабування сховищ ... **243**
9.4. Засоби забезпечення доступності і відмовостійкості ... **248**
9.5. Бази даних NoSQL ... **250**
9.5.1. Особливості технології NoSQL ... 250
9.5.2. Теорема CAP (Брюєра) ... 251
9.5.3. Сховища «ключ-значення» ... 253
9.5.4. Сховища документів ... 255
9.5.5. Сховища на основі стовпців ... 259
9.5.6. Графові сховища ... 261
Контрольні запитання ... **264**
Тестові завдання ... **265**
10. БЕЗПЕКА БАЗ ДАНИХ ... 268
10.1. Базові концепції безпеки баз даних ... **268**
10.2. Загрози безпеці баз даних ... **269**
10.2.1. SQL ін’єкції ... 269
10.2.2. Інші види загроз базам даних ... 273
10.3. Механізми безпеки баз даних ... **276**
10.3.1. Ідентифікація і автентифікація ... 277
10.3.2. Авторизація і контроль доступу ... 280
10.3.3. Аудит ... 284
10.3.4. Криптографічний захист даних ... 287
Контрольні запитання ... **289**
Тестові завдання ... **290**
ВИКОРИСТАНА ЛІТЕРАТУРА ... 293
ПОКАЖЧИКИ ... 294

ВСТУП

Сучасні системи баз даних відіграють важливу роль у різних галузях, таких як фінанси, охорона здоров'я, електронна комерція та соціальні мережі, забезпечуючи єдину платформу для зберігання даних та управління ними. Бази даних становлять основу інформаційних систем (ІС), а технології, які в них використовуються, різко змінили спосіб зберігання та доступу до інформації. Останнім часом ці технології досягли значного прогресу, завдяки чому системи баз даних стають все більш доступними для широкої аудиторії.

Керування базами даних еволюціонувало від спеціалізованої комп'ютерної програми до центрального компонента практично всіх підприємств, і, як наслідок, знання про системи баз даних стали невід'ємною частиною освіти з інформатики. У цьому підручнику викладені основні поняття керування базами даних. Ці поняття включають аспекти проєктування бази даних, мови баз даних і реалізацію системи баз даних.

Концепції та технології баз даних складалися поступово і завжди були тісно пов'язані з розвитком систем автоматизованого оброблення інформації. Нині це цілком сформована дисципліна, яка ґрунтується на доволі формалізованих підходах і включає широкий спектр прийомів і методів створення баз даних. Навчальну дисципліну «Бази даних» включено до стандартів багатьох спеціальностей, пов'язаних із підготовкою спеціалістів з обробки та аналізу даних і обчислювальної техніки.

Цей підручник призначений для студентів другого курсу з дисципліни «Бази даних та інформаційні системи». В підручнику охоплено важливі теоретичні результати, але формальні докази випущено. Замість доказів використовуються приклади, щоб зрозуміти, чому результат є істинним.

Мета підручника полягає в систематичному викладенні теоретичних і практичних основ побудови систем баз даних, можливостей сучасних систем керування базами даних, технологій їхнього використання. У підручнику викладаються базові знання, необхідні для подальшого набуття знань у галузі технологій і методів проєктування надійних баз даних, захисту та забезпечення безпеки інформаційних систем з використанням технології баз даних.

Пояснюється підхід до процесу створення реляційних баз даних, який був випробуваний і перевірений протягом багатьох років як в академічних, так і в практичних умовах. Концептуальний, логічний і фізичний дизайн бази даних є трьома ключовими етапами, які складають процес створення. Створення концептуальної моделі даних, яка не пов'язана з жодними фізичними факторами, є першим етапом процесу. На другому етапі, усуваючи компоненти, які не можуть бути створені в реляційних системах, концептуальна модель перетворюється на логічну модель даних. На третьому етапі, етапі фізичного проєктування, розробляються структури для зберігання даних та методи доступу до них.

Книга поділена на десять розділів, кожен з яких охоплює важливу тему сучасних систем баз даних.

Розділ 1 ознайомлює з основними поняттями систем баз даних, а також з деякими загальними та важливими основними концепціями баз даних. Надано класифікацію інформаційних систем, пояснені недоліки файлової системи та переваги системи баз даних. Розглянуто функціональні компоненти системи баз даних і СКБД. Наведено абстрактну архітектуру ANSI / SPARC. Надано визначення моделі даних та їх класифікація.

Розділ 2 присвячений першим етапам проєктування – створенню функціональної і концептуальної моделей. Розглядаються питання використання діаграм потоків даних (DFD) для створення функціональної моделі. Детально розглянуто питання використання моделі Entity-Relationship (ER) в концептуальному моделюванні. Показані основні етапи створення ER-моделі визначення сутностей, атрибутів сутностей і зв'язків між сутностями.

У розділі 3 розглянуто найпопулярнішу модель даних – реляційну. Надані основні визначення і поняття реляційної моделі. Проаналізовано структурну і цілісну частину реляційної моделі, визначені поняття ключа і зовнішнього ключа, обмежень цілісності. Надані правила перетворення ER-моделі в реляційну. Маніпуляційна частина моделі, що присвячена можливим операціям над даними, розглянута з використанням апарату реляційної алгебри.

У розділі 4 розглядаються питання побудови логічної моделі. Проаналізовано різного роду аномалії оновлення даних, надано поняття функціональної залежності. Показано використання властивостей функціональних залежностей і нормальних форм для побудови оптимальної логічної моделі. Обговорюються концепції нормалізації з детальним поясненням нормальних форм, тобто першої, другої, третьої та BCNF. Продемонстровано перетворення моделі відповідно до процесу нормалізації. Показано доцільність процесу денормалізації моделі.

Розділ 5 присвячений базовим поняттям мови SQL. Розглянуто синтаксис мови SQL і використання команд SQL для отримання бажаних результатів. Розглядаються команди визначення даних (DDL), маніпулювання даними (DML). Основна увага приділяється команді читання даних SELECT. На прикладах продемонстровано використання функцій і операторів, групування даних і агрегатних функцій, одно– і багатотабличні запити, прості і корельовані підзапити. Розглянуті подання (view) як засіб управління доступом. Показана можливість створення і використання таких об'єктів бази даних, як збережені процедури, тригери і функції.

Розділ 6 присвячений розширеним можливостям мови SQL, що використовуються для аналітичної обробки даних. Розглянуто такі питання, як ранжування даних, проміжні підсумки в агрегованих даних, віконні функції, засоби ранжування, зсуву даних, розрахунку ковзного середнього.

У розділі 7 розглядаються питання фізичного проєктування бази даних. Основну увагу приділено розгляду методів індексування, їх порівняльному аналізу та плануванню індексів. Надані рекомендації щодо використання індексів.

В розділі 8 визначається поняття транзакції, їхніх властивостей ACID. Розглянуті проблеми, пов'язані з одночасним виконанням транзакцій, і різні схеми, які

використовуються для управління паралельним доступом. Пояснюються різні протоколи, що використовуються для управління транзакціями і запобіганню тупиків. Пояснюються різні методи відновлення після втрати даних у разі збою системи бази даних.

Розділ 9 розповідає про особливості великих даних (Big Data) і про технології, що використовуються для зберігання та управління ними. Розглянуті засоби реалізації конвеєру обробки великих даних: Hadoop (зокрема розподілена файлова система HDFS та парадигма паралельного програмування MapReduce) та NoSQL. Визначаються підходи до масштабування сховищ, засоби забезпечення доступності і відмовостійкості, існуючі варіанти побудови сховищ NoSQL.

Розділ 10 зосереджений на основах засобах безпеки бази даних, що використовуються для захисту від несанкціонованого доступу. Розглядаються актуальні загрози безпеці баз даних. Показано комплекс основних механізмів захисту і особливості їх реалізації, у тому числі ідентифікація і автентифікація, контроль доступу, реєстрація і аудит.

Підручник призначений для студентів Навчально-наукового фізико-технічного інституту національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського", а також може бути використаний студентами інших спеціальностей.

1. ВСТУП ДО СИСТЕМ БАЗ ДАНИХ

Концепції та технології баз даних складалися поступово і завжди були тісно пов'язані з розвитком систем автоматизованого оброблення інформації. Нині це цілком сформована дисципліна, яка ґрунтується на доволі формалізованих підходах і включає широкий спектр прийомів і методів створення баз даних. Сьогодні теорія баз даних є обов'язковою для вивчення студентами практично всіх технічних спеціальностей. Навчальну дисципліну «Бази даних» включено до стандартів усіх спеціальностей, пов'язаних із підготовкою спеціалістів з обробки та аналізу даних і обчислювальної техніки.

1.1. Дані, інформація, інформаційні системи

У сучасних комп'ютерних системах поняття даних є узагальненим. У широкому розумінні дані включають числа, текст, графіку, зображення, аудіо, відео та багато інших форм, і після оцифрування вони можуть зберігатися в комп'ютері.

Дані – це необроблені факти. Слово «необроблені» вказує на те, що факти ще не були оброблені щоб розкрити їх значення. Наприклад, цифри 2023 як дані можуть вказувати на загальну кількість працівників підприємства, на рік народження або на PIN-код.

Оброблені дані, окрім форми представлення, також включають семантику, тобто значення даних. Дані та семантика даних тісно пов'язані між собою. Інформація є даними, яким надається деякий зміст (семантику) у конкретній ситуації. Інформація виникає як результат обробки даних для розкриття їхнього значення. Обробка даних може бути як простою, як упорядкування даних для виявлення закономірностей, так і складною, як прогнозування чи формування висновків за допомогою статистичного моделювання (бізнес-аналітика, BI). Така інформація може бути використана як основа для прийняття рішень.

Звичайно, такі перетворення неможливі без використання комп'ютерних систем, орієнтованих на зберігання і обробку даних. Такі системи мають назву інформаційні системи. Найбільш складні інформаційні системи дозволяють поєднувати потужність обчислюваної техніки і людський досвід (експерти), отримуючи те, що має назву знання.

Обробка даних на такому рівні передбачає використання методів штучного інтелекту (Artificial Intelligence – AI) – розділу інформатики, який займається вирішенням когнітивних завдань, які зазвичай призначені для людського інтелекту. Також широко використовуються алгоритми машинного навчання (Machine Learning – ML).

Процес перетворення даних в знання наведено на рис. 1.1.

Рис. 1.1. Різниця між даними, інформацією, бізнес-аналітикою та знаннями

Таким чином, важливішій компонент процесу перетворення даних є інформаційна системи. Інформаційна система (ІС) – це програмний комплекс, в основні функції якого входять:

- підтримка надійного зберігання даних у пам'яті комп'ютера;
- виконання необхідних для користувача перетворень інформації та обчислень;
- надання користувачам зручного інтерфейсу.

Існують багато варіантів архітектури і технологій функціонування ІС.

1.2. Класифікація інформаційних систем

За способом взаємодії ІС можна поділити на дві категорії: файл-серверні і клієнт-серверні. Перші – більш старі. В таких системах на сервері знаходяться тільки файли БД. На комп'ютері користувача встановлено прикладне програмне забезпечення і СКБД. Приклад – MS Access, DBase, FoxPro.

Клієнт-серверні ІС – це сучасні інформаційні системи, що в свою чергу поділяються за різними критеріями.

Клієнт-серверна архітектура

Архітектура «клієнт-сервер» розділяє функції застосунка користувача (званого клієнтом) і сервера. Застосунок-клієнт формує запит до сервера, на якому розташована БД. Ресурси клієнта не беруть участі у фізичному виконанні запиту. Оскільки клієнтському застосунку надсилається тільки результат виконання запиту, мережею надсилаються тільки ті дані, які необхідні клієнту. У підсумку знижується навантаження на мережу [13].

Крім того, сервер БД, якщо це можливо, оптимізує отриманий запит таким чином, щоб він був виконаний у мінімальний час з найменшими накладними витратами. Усе це підвищує швидкодію системи і знижує час очікування результату запиту.

Зазвичай у застосунку виділяють такі групи функцій:

- функції введення та відображення даних;
- прикладні функції, що визначають основні алгоритми вирішення завдань застосунка;
- функції обробки даних усередині застосунка,
- функції управління інформаційними ресурсами;
- службові функції, що відіграють роль зв'язок між функціями перших чотирьох груп

Клієнт-серверна архітектура найбільш популярний варіант організації ІС. Класифікуються за кількістю рівнів ІС.

Дворівнева архітектура клієнт-сервер

Дворівневі архітектури, складаються з клієнтського комп'ютера і серверного комп'ютера, які взаємодіють за допомогою чітко визначеного протоколу. Яка частина функціональності реалізується клієнтом, а яка – сервером, може бути зроблено в різний спосіб. У традиційній архітектурі клієнт-сервер клієнт реалізує лише графічний інтерфейс користувача, а сервер – бізнес-логіку та управління даними; такі клієнти часто називають тонкими клієнтами.

Можливий і інший поділ, наприклад, більш потужні клієнти, які реалізують як інтерфейс користувача, так і бізнес-логіку, або клієнти, які реалізують інтерфейс користувача і частину бізнес-логіки, а решта реалізується на рівні сервера. Такі клієнти часто називають товстими клієнтами, а така архітектура має назву дворівнева і зображена на рис. 1.2.

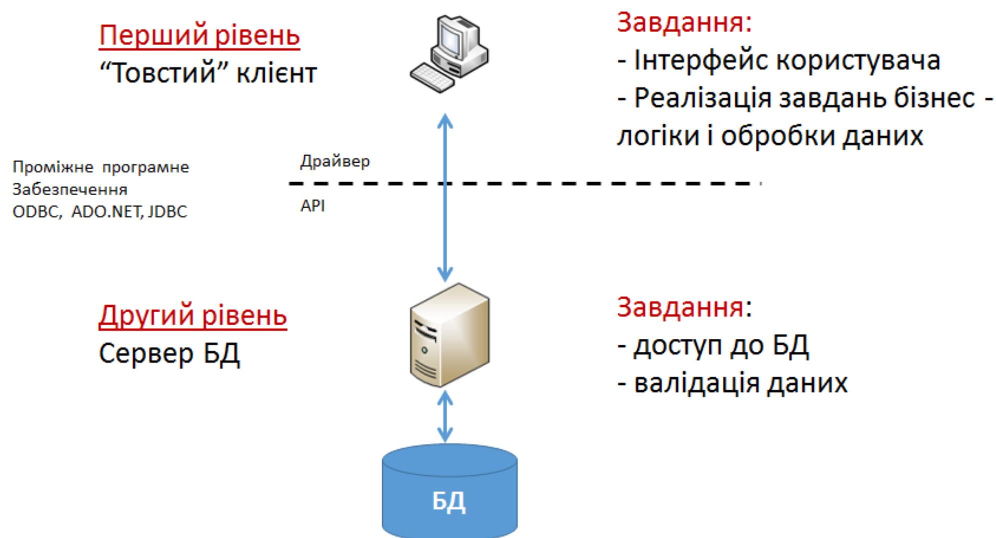


Рис. 1.2. Дворівнева архітектура клієнт-сервер

Підключення програми, написаної загальною мовою програмування до бази даних досягається за допомогою спеціального програмного забезпечення, яке називається проміжним програмним забезпеченням, орієнтованим на базу даних. Проміжне програмне забезпечення необхідне для підключення програми до бази даних і складається з двох частин: інтерфейсу прикладного програмування (API) і драйвера бази даних для

підключення до бази даних певного типу (наприклад, SQL Server або Oracle). Найпоширенішими API є ODBC, ADO.NET та JDBC.

Модель «товстий клієнт» має кілька недоліків у порівнянні з моделлю «тонкий клієнт». По-перше, немає центрального місця для оновлення та підтримки бізнес-логіки, оскільки код програми виконується на багатьох клієнтських комп'ютерах.

По-друге, необхідний великий рівень довіри між сервером і клієнтами. Наприклад, СКБД банку повинна довіряти банкомату, щоб залишити базу даних в узгодженому стані.

Третім недоліком архітектури з товстим клієнтом є те, що вона погано масштабується з ростом кількості клієнтів. Зазвичай вона не може працювати з кількома сотнями клієнтів. Логіка програми на клієнті надсилає SQL-запити до сервера, а сервер повертає результат запиту клієнту, де відбувається подальша обробка. Між клієнтом і сервером можуть передаватися результати запитів дуже великого обсягу.

По-четверте, системи з товстим клієнтом не масштабуються також тому, що додаток звертається до все більшої кількості систем баз даних. Припустимо, що є N різних систем баз даних, до яких звертаються M клієнтів, тоді в будь-який момент часу буде відкрито $N \bullet M$ різних з'єднань, що явно не є масштабованим рішенням.

Ці недоліки систем з товстим клієнтом і широке впровадження стандартних клієнтів – веббраузерів – призвели до широкого використання архітектури тонких клієнтів.

Трирівнева архітектура клієнт-сервер

Трирівнева архітектура з'явилася в 1995 році і є подальшим удосконаленням технології «клієнт-сервер». В такої архітектурі є три шари, кожен з яких потенційно працює на іншій платформі (рис. 1.3):

1. Рівень інтерфейсу користувача, який працює на комп'ютері кінцевого користувача (клієнта).
2. Бізнес-логіка та рівень обробки даних. Цей середній рівень працює на сервері і його часто називають сервером застосунків.
3. СКБД, яка зберігає дані, необхідні середньому рівню. Цей рівень може працювати на окремому сервері, який називається сервером баз даних.

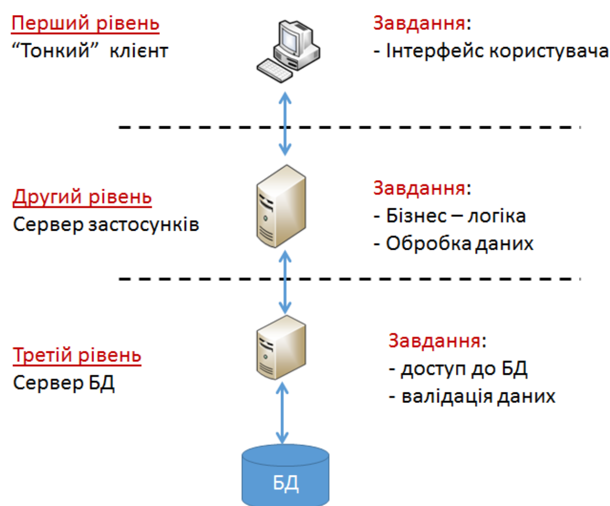


Рис. 1.3. Трирівнева архітектура клієнт-сервер

Трирівнева архітектура додатково відокремлює логіку програми від управління даними. Вся бізнес-логіка, що раніше входила в клієнтські застосунки, виділяється в окремий рівень, який називається сервером застосунків. При цьому клієнтським застосунком залишається лише користувальницький інтерфейс. Крім того, максимально знижуються вимоги до апаратного забезпечення клієнтських комп'ютерів.

Були розроблені різні технології, що дозволяють розподіляти три рівні додатку між різними апаратними платформами і різними фізичними об'єктами.

Переваги трирівневої моделі клієнт/сервер полягають у наступному:

- Централізована реалізація логіки застосунків вирішує проблеми управління і оновлення.
- Покращена масштабованість, оскільки за потреби можна додати кілька серверів застосунків. (Те ж саме можна зробити і з серверами баз даних, але для цього потрібна технологія розподілених баз даних для синхронізації будь-яких оновлень даних між усіма копіями даних).
- Вона зберегла переваги користувацького інтерфейсу дворівневої моделі.
- Клієнтські робочі станції були набагато дешевшими. Стандартні персональні комп'ютери можуть легко виконувати цю роботу з представлення результатів обробки.

Але є і недоліки:

- Вона все ще складнішою порівняно з централізованою моделлю.
- Спеціальні методи презентації та логіка збільшували витрати і обмежували можливість перенесення на різні клієнтські платформи.

N-рівнева клієнт-серверна архітектура

З поширенням веббраузерів інформаційні системи (ІС) перейшли на використання вебсторінок як основного методу представлення інформації. N-рівнева модель клієнт/сервер (яку іноді називають моделлю інтернет-обчислень) показана на рис. 1.4.

Еволюція від трирівневої до N-рівневої включала додавання вебсервера для обробки відповідей на запити клієнтів і рендерингу (створення) вебсторінок, а також

заміну власної логіки відображення на робочій станції на стандартний веббраузер. Взаємодія між клієнтом та вебсервером відбувається наступним чином:

1. За допомогою веббраузера клієнт надсилає запит у вигляді URL-адреси (Uniform Resource Locator – уніфікований покажчик ресурсів).
2. Вебсервер перетворює URL-адресу в запит до серверу застосунків Сервер застосунків звертається до сервера баз даних, отримує від нього результати запиту і надає їх вебсерверу.
3. Вебсервер обробляє запит, формує веб-сторінку і надсилає її клієнту.
4. Користувач на клієнтській робочій станції працює з вебсторінкою і, врешті-решт, надсилає новий запит на вебсервер, після чого цикл повторюється.

Рис. 1.4. N-рівнева архітектура клієнт-сервер

Ця архітектура дуже популярна у сучасних інформаційних систем. Переваги N-рівневої моделі клієнт/сервер полягають у наступному:

- Вона пропонує стандартний метод презентації за допомогою вебсторінок.
- Однакова архітектура може бути використана для внутрішніх (Інтранет) і зовнішніх (інтернет) застосунків.
- Вона зберігає всі переваги трирівневої моделі клієнт/сервер.
- Ось недоліки N-рівневої моделі клієнт/сервер:
- Проблеми з безпекою існують тому, що Інтернет не був розроблений з урахуванням безпеки.
- N-рівнева модель клієнт/сервер потенційно потребує більших команд розробників, оскільки на кожному рівні потрібен фахівець.
- Модель потенційно вимагає більше апаратного забезпечення (за вищу загальну вартість). Можна об’єднати деякі рівні на загальних серверах, але такий підхід не рекомендується, оскільки поділ за функціями підвищує безпеку.

- Збільшення витрат на адміністрування більшого технологічного стеку.

Для ІС з вебсервером використовують поділ функцій і компонентів системи на фронтенд (frontend) і бекенд (backend) (рис. 1.5).

Фронтенд – це публічна частина вебзастосунків (вебсайтів), з якою користувач може взаємодіяти і контактувати безпосередньо. До фронтенд входить відображення функціональних завдань, призначеного для користувача інтерфейсу, що виконуються на стороні клієнта, а також обробка призначених для користувача запитів. По суті, фронтенд – це все те, що бачить користувач під час відкриття вебсторінки.

Бекенд – це програмно-апаратна частина ІС. Це все те, що відбувається на стороні сервера і що залишається невидимим користувачеві (сам сервер теж є частиною бекенда, тільки програмно-апаратною).

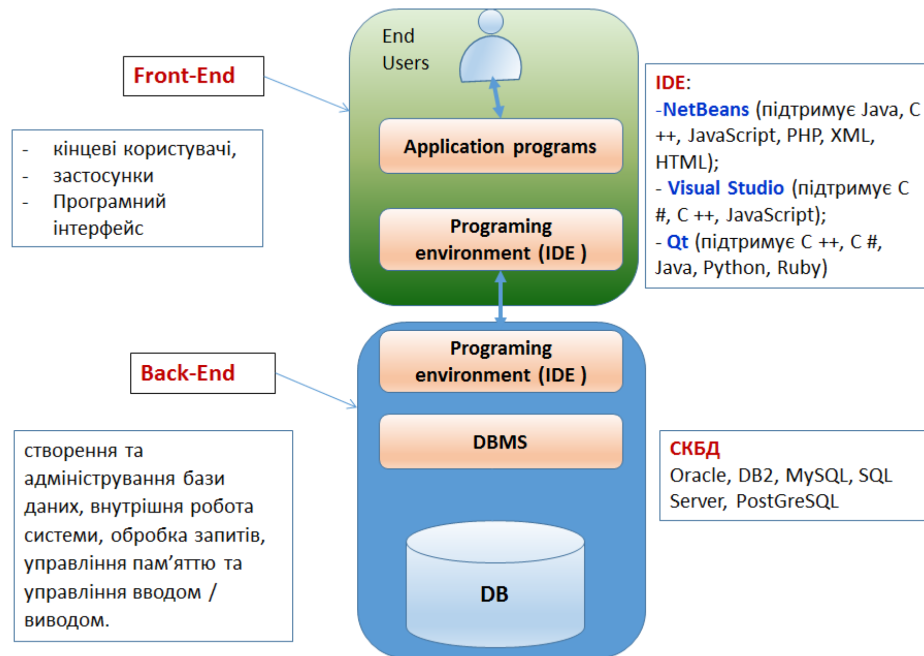


Рис. 1.5. Розподіл функцій вебзастосунку між клієнтською і серверною частинами

В ІС як правило, фронтенд і бекенд функціонують на різних машинах і один бекенд (сервер) обслуговує багато фронтенд (клієнтів). Але в розподілених системах можлива така організація, коли на кожної машині знаходиться як фронтенд, так і бекенд.

1.3. Типи інформаційних систем за характером інформації, що обробляється

Навіть в межах одного підприємства інформаційні системи можуть вирішувати завдання, для рішення яких потрібні різні програмно-апаратні конфігурації, технології баз даних і математичні моделі. Багато в чому це залежить від рівня складності прикладних завдань, що вирішує ІС (рис. 1.6).

Рис. 1.6. Типи інформаційних систем за характером обробки інформації

На стратегічному рівні менеджери розробляють загальні бізнес-стратегії, цілі та завдання в рамках стратегічного плану компанії. Стратегічні рішення – це дуже неструктуровані рішення, що виникають у ситуаціях, коли немає процедур чи правил, які б скеровували осіб, відповідальних за прийняття рішення до правильного вибору. Наприклад, рішення про вихід на новий ринок протягом, скажімо, наступних трьох років. Додавання в систему нових даних відбувається відносно рідко, великими блоками, наприклад, один раз на місяць або квартал. Дані, додані в систему, як правило, ніколи не видаляються.

ІС такого класу мають назву Системи підтримки прийняття рішень (Decision Support System – DSS). Характерні технології баз даних, що використовуються в таких ІС:

- оперативна аналітична обробка – OLAP (OnLine Analysis Processing);
- сховища даних (Data Warehouse);
- системи інтелектуального аналізу даних (Data Mining).

На управлінському рівні постійно оцінюють діяльність компанії, щоб покращити можливості фірми виявляти, адаптуватися та впливати на зміни. Управлінські рішення охоплюють коротко- та середньострокові плани, графіки та бюджети разом із політикою, процедурами та бізнес-цілями для фірми. ІС такого рівня мають назву Інформаційні системи управління (ІСУ). Прикладами таких систем є системи класу ERP (Enterprise Resource Planning) і CRM (Client Relationship Management).

На оперативному рівні розробляють, контролюють та підтримують основні види операцій, необхідні для повсякденної діяльності. Оперативні рішення вважаються структурованими рішеннями, які виникають у ситуаціях, коли встановлені процеси пропонують потенційні рішення. Структуровані рішення приймаються часто і мають повторюваний характер. Для ІС такого рівня характерні:

- Регулярний потік досить простих транзакцій, що грають роль замовлень, платежів, запитів і т. і.
- Вимоги до систем: висока продуктивність обробки транзакцій і гарантована доставка інформації при віддаленому доступі до БД.
- В таких ІС використовуються технології баз даних що мають назву Оперативна обробка транзакцій – OLTP (OnLine Transaction Processing).

Приклад: бухгалтерська програма обліку господарчих операцій.

1.4. Підхід до управління даними з використанням БД

Недоліки підходу без баз даних

Спочатку інформаційні системи будувалися таким чином, що кожен тип програмного забезпечення (ПЗ) мав доступ до окремого файлу з даними. Усі особливості таких даних «знало» тільки відповідне ПЗ. Якщо було необхідно використовувати зберігання даних для кількох прикладних програм, створювали кілька файлів із даними. Такий підхід мав принципові недоліки:

- надмірність та суперечність даних. Одна і та ж інформація може дублюватися в декількох місцях (файлах);
- залежність структур даних та прикладних програм. Оскільки дані розкидані в різних файлах, а файли можуть бути в різних форматах, написання нових прикладних програм для отримання відповідних даних стає складним завданням;
- проблеми цілісності. Коли додаються нові обмеження, важко змінити всі програми для їх застосування. Особливо коли обмеження включають кілька елементів даних з різних файлів;
- проблеми узгодженості. Послідовність декількох операцій повинна виконуватись як єдине ціле. Це важко забезпечити у звичайній системі обробки файлів;
- проблеми одночасного доступу. Система повинна підтримувати певну форму управління послідовністю операцій. Але це важко забезпечити, оскільки доступ до даних може мати багато різних прикладних програм;
- проблеми з безпекою. Не кожен користувач системи баз даних повинен мати можливість отримати доступ до всіх даних. Але, оскільки додаткові програми додаються до обробки файлів, застосування таких обмежень ускладнено.

Переваги використання баз даних

Вказані недоліки можна усунути, якщо між прикладним програмним забезпеченням і даними створити прошарок, який ізолює дані від програм і при цьому забезпечить управління даними. Такий прошарок має назву Система Керування Базою Даних (DBMS), а дані і СКБД разом створюють Систему Баз Даних (DBS) (рис. 1.7).

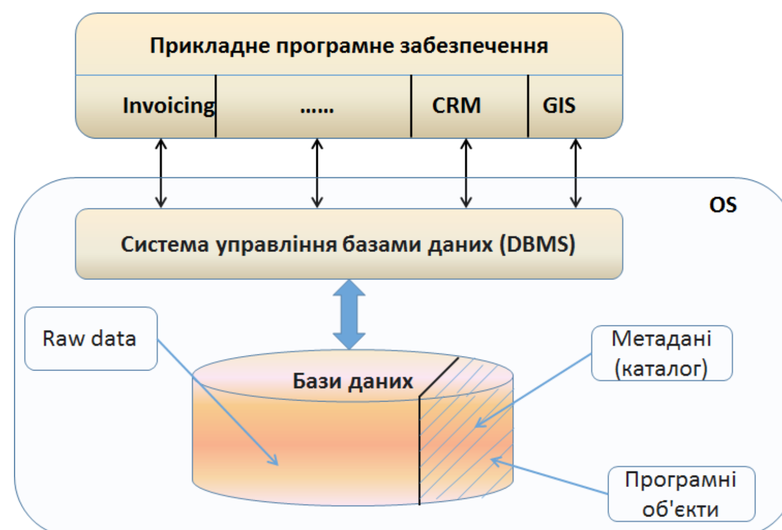


Рис. 1.7. Структура інформаційної системи с базою даних

Переваги використання БД:

- немає дублювання інформації за рахунок структурування даних;
- підвищення цілісності і збереження даних;
- простота і легкість використання даних;
- незалежність прикладних програм від даних;
- забезпечення необхідної швидкості доступу до даних;
- автоматизована реорганізація даних;
- захист від спотворення і знищення даних;
- обробка незапланованих запитів до інформації;
- створення умов для організації розподіленої обробки даних.

Основні компоненти системи баз даних

До основних компонентів системи баз даних відносяться (рис. 1.8):

- власне бази даних;
- програмна система керування даними (СКБД);
- прикладне програмне забезпечення і додаткові утиліти;
- користувачі БД, яких традиційно поділяють на адміністраторів, розробників і кінцевих користувачів.

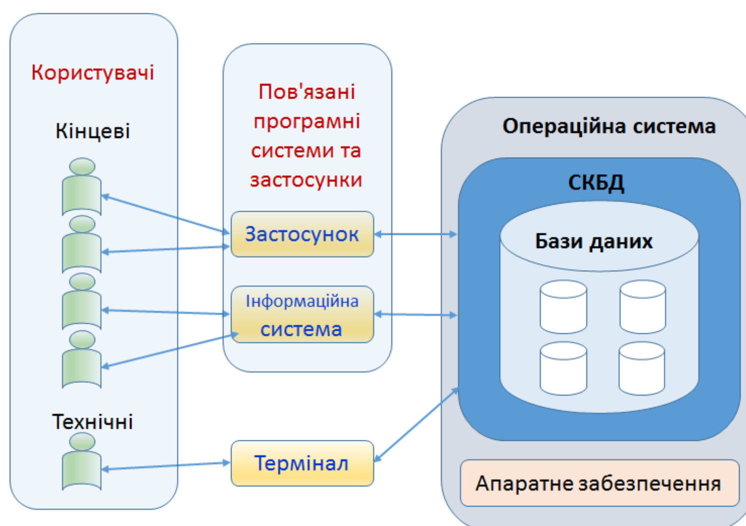


Рис. 1.8. Основні компоненти системи баз даних

Важливі терміни баз даних.

Система баз даних (DBS) – це комп'ютеризована система, загальна мета якої – зберігати інформацію та зробити її доступною, коли це потрібно.

Система керування базами даних (СКБД, DBMS) – це набір програм, які дозволяють керувати базою даних, що передбачає:

- визначення даних, яке полягає у створенні та управління об'єктами, залежностями, обмеженнями цілісності, представлення даних;
- маніпулювання даними, що включає введення, оновлення, видалення, пошук, реорганізацію та агрегування даних;

- забезпечення безпеки системи та даних, що означає контроль доступу до системи, ресурсів та даних;
- підтримку мови програмування.

Предметна область (ПО) – це частина реального світу, що підлягає вивченню для організації управління і, в кінцевому рахунку, автоматизації.

Об'єкти – функціонуючі в ПО люди, процеси, предмети, інформація про яких повинна зберігатися в БД.

Концептуальна модель предметної області – формальний опис об'єктів, їх властивостей і відносин.

База даних (БД) – динамічна інформаційна модель ПО, що адекватно відображає її стан в будь-який момент часу. Реалізується як іменована сукупність даних, організованих за певними правилами, що передбачають загальні принципи опису, зберігання і маніпулювання даними, не залежна від прикладних програм.

1.5. Класифікація Систем баз даних

За сферою можливого застосування:

- універсальні (Oracle, MySQL, PostgreSQL…);
- спеціалізовані або проблемно-орієнтовані (системи військового призначення, державні інформаційні системи).

За характером використання:

- персональні або Настільні (MS Access, Visual FoxPro, Paradox);
- розраховані на багато користувачів (Oracle, MySQL, MS SQL Server).

За ступенем доступності:

- загальнодоступні (Банк законодавчих документів http://zakon.rada.gov.ua);
- БД з обмеженим доступом (БД, що використовуються в системі МВС, БД дактилоскопічної інформації (АДІС «Папілон»).

За формою зберігання інформації:

- фактографічні (СКБД Oracle, PostgreSQL, MySQL, MS SQL Server);
- документальні (БД наукового цитування: Web of Science, Scopus Довідкові правові системи Ліга: Закон, Нормативні акти України, Експерт-Юрист).

За топологією зберігання:

- локальні (Dbase, MS Access);
- централізовані (Oracle, MySQL, MS SQL Server);
- розподілені (Google BigTable, БД NoSQL).

За характером обробки інформації:

- OLTP (OnLine Transaction Processing) – системи оперативної обробки інформації;
- OLAP (OnLine Analytical Processing) – системи для складної аналітичної обробки інформації.

За місцем зберігання БД:

- в зовнішній пам’яті. У цьому випадку сховище зовнішньої пам’яті доступне СКБД через системні виклики операційної системи і структуроване у вигляді блоків. Потрібний блок із зовнішньої пам’яті буферизується в оперативній пам’яті (ОП), де відбувається подальша робота;
- в ОП (in-memory). In-memory СКБД розташовують усю БД в ОП і використовують зовнішню пам’ять для забезпечення довготривалості транзакцій. За рахунок цього досягається дуже велика швидкість роботи з БД (на 4 порядки вища порівняно з магнітними дисками). Існує кілька підходів до організації in-memory СКБД. Найпопулярнішим є варіант, за якого БД зберігається цілком і у ОП, і в зовнішній пам’яті, однак операція читання відбувається з ОП, а запис – в обидві. Поява SSD-накопичувачів якісно змінила ситуацію, і тепер різниця у швидкості доступу між зовнішньою пам’яттю на SSD і ОП становить усього 2 порядки.

1.6. Функції СКБД

СКБД виконує кілька важливих функцій, які гарантують цілісність і узгодженість даних у базі даних. Більшість цих функцій є прозорими для кінцевих користувачів, і більшість з них можна реалізувати лише за допомогою СУБД. До них належать такі функції [1,6].

Управління даними в зовнішній пам’яті. Ця функція надає можливості виконання основних операцій, які здійснюються з даними, – зберігання, вилучення та оновлення інформації. Вона включає в себе забезпечення необхідних структур зовнішньої пам’яті як для зберігання даних, що безпосередньо входять до БД, так і для службових цілей, наприклад для прискорення доступу до даних.

Управління транзакціями. Транзакція – це послідовність операцій над БД, які СКБД розглядає як єдине ціле. Транзакція являє собою набір дій, що виконуються з метою доступу або зміни вмісту бази даних. Управління транзакціями забезпечує безконфліктний паралельний доступ до даних і узгодженість даних.

Відновлення бази даних. Під надійністю зберігання розуміється те, що СКБД має бути в змозі відновити останній узгоджений стан БД після будь-якого апаратного або програмного збою. Підтримка надійності зберігання даних у БД вимагає надмірності зберігання даних, причому та частина даних, яка використовується для відновлення, має зберігатися особливо надійно. Найпоширенішим методом підтримання такої надлишкової інформації є ведення журналу змін БД.

Підтримка мови БД. У сучасних СКБД зазвичай підтримується єдина інтегрована мова, що містить усі необхідні засоби для роботи з БД, починаючи від її створення, і забезпечує базовий користувацький інтерфейс із базами даних. Стандартною мовою найпоширеніших нині реляційних СКБД є мова SQL (Structured Query Language – мова структурованих запитів). Мова SQL дає змогу визначати схему реляційної БД і маніпулювати даними. Нереляційні СКБД підтримують власні мови програмування. Деякі з них схожі на SQL, інші суттєво відрізняються.

Організацій словника даних. Однією з основоположних ідей системи баз даних є наявність інтегрованого системного каталогу з даними про схеми, користувачів, застосунки тощо. Системний каталог, який ще називають словником даних, є сховищем інформації, що описує дані в базі даних. Передбачається, що каталог доступний як користувачам, так і функціям СКБД. Зазвичай у словнику даних: міститься така інформація:

- імена, типи та розміри елементів даних;
- зв'язки даних;
- умови підтримки цілісності, що накладаються на дані;
- користувачів, яким надано право доступу до даних;
- зовнішня, концептуальна і внутрішня схеми та відображення між ними;
- статистичні дані, наприклад частота транзакцій і лічильники звернень до об'єктів бази даних.

Управління паралельним доступом. Одна з основних цілей створення і використання СКБД полягає в тому, щоб всі зареєстовані користувачи могло здійснювати паралельний доступ до спільно оброблюваних даних. СКБД повинна гарантувати, що в разі одночасного доступу до бази даних багатьох користувачів конфліктів не відбудеться.

Управління буферами оперативної пам'яті. СКБД зазвичай працюють із БД значного розміру. Зрозуміло, що якщо при зверненні до будь-якого елемента даних буде здійснюватися обмін із зовнішньою пам'яттю, то вся система працюватиме зі швидкістю пристрою зовнішньої пам'яті. Практично єдиним способом реального збільшення цієї швидкості є буферизація даних в оперативній пам'яті. У розвинених СКБД підтримується власний набір буферів оперативної пам'яті з власною дисципліною заміни буферів.

Контроль доступу до даних. СКБД повинна мати механізм, що гарантує можливість доступу до бази даних тільки санкціонованих користувачів і що захищає її від будь-якого несанкціонованого доступу.

Підтримка цілісності даних. Термін цілісність використовується для опису коректності та несуперечливості збережених у БД даних. Реалізація підтримки цілісності даних передбачає, що СКБД має містити відомості про ті правила, які не можна порушувати під час роботи з даними, і володіти інструментами контролю за тим, щоб дані та їхні зміни відповідали заданим правилам.

1.7. Функціональні компоненти СКБД

Основні функціональні компоненти СКБД наведені на рис. 1.9.

Ядро СКБД (DBMS Engine). Ядро СКБД є центральної складовою і сполучною ланкою між усіма іншими підсистемами та фізичним пристроєм (комп'ютером) через операційну систему. Деякі важливі функції:

- надання прямого доступу до утиліт і програм операційної системи (наприклад, введення/виведення запитів, запити на стиснення даних, запити на зв'язок тощо);
- управління доступом до файлів (та управління даними) через операційну систему;

- управління передачею даних між пам’яттю та системним буфером для ефективного виконання запитів користувачів;
- обслуговування службових даних і метаданих, що зберігаються в словнику даних (системний каталог).

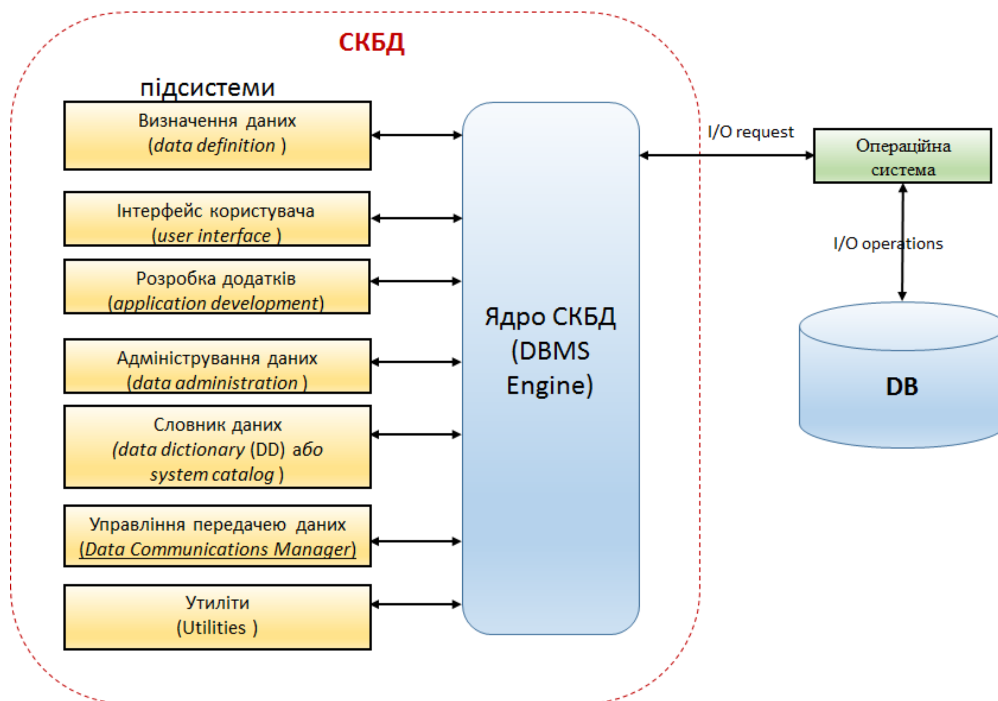


Рис. 1.9. Функціональні компоненти СКБД

Підсистема визначення і маніпулювання даними. Підсистема визначення даних складається з інструментів і утиліт для визначення і зміна структури бази даних. Структура включає реляційні таблиці, обмеження, профілі користувачів, службові структури даних тощо. DDL (мова визначення даних) є підмножиною мови SQL і використовується для визначення всіх об’єктів бази даних, які складають концептуальна схема (відношення, зв’язки, обмеження тощо). Інша підмножина це мова DML (мова маніпулювання даними) включає можливості створення запитів для вставки, видалення, оновлення та пошуку даних в базі даних. DCL (мова управління даними) використовується для налаштування середовища користувача, його можливостей доступу до даних.

Підсистема інтерфейсу користувача. Підсистема інтерфейсу користувача дозволяє користувачам і програмам отримувати доступ до бази даних через інтерактивну мову запитів, таку як SQL. Традиційний інтерфейс базувався на командах; однак поточна тенденція полягає у створенні графічних інтерфейсів користувача (GUI).

Інтерфейс користувача також може включати розширення мови програмування, специфічні для СКБД (наприклад, PL/SQL Oracle). Ці розширення мови стосуються лише тієї СКБД, у якій вони використовуються. Крім того, СКБД може підтримувати кілька мов високого рівня, наприклад C++ і Java.

Підсистема розробки застосунків. Підсистема розробки програм містить інструменти для розробки компонентів програми, таких як форми, звіти та меню. У

деяких випадках його можна об'єднати з підсистема інтерфейсу користувача. Як правило, ця підсистема забезпечує графічний інтерфейс користувача (GUI), який накладається на базову мову. Набір може містити автоматичний генератор коду (як у Team Developer) або безперешкодний доступ компілятора основної мови (як у Oracle).

Підсистема адміністрування даних. Підсистема адміністрування даних складається з набору утиліт, які полегшують ефективне управління базою даних. У цю підсистему входять засоби резервного копіювання та відновлення, налаштування бази даних і управління сховищем. Зазвичай він використовується адміністраторами баз даних або інженерами.

Підсистема словника даних. Словник даних – це традиційний термін для позначення системного каталогу. Системний каталог містить усі метадані – інформацію про структуру бази даних, зв'язки між об'єктами бази даних, привілеї системи та об'єктів, користувачів, обмеження цілісності тощо. Автоматично створюється і підтримується СКБД.

Менеджер передачі даних. Традиційно це окрема система, пов'язана з СКБД. Менеджер передачі даних виконує такі функції, як:

- обслуговування зв'язку з віддаленими користувачами в розподіленому середовищі;
- обробка повідомлень до та з СКБД;
- зв'язок з іншими компонентами СКБД.

Сучасні системи, як правило, мають цю підсистему як невід'ємну частину комплексу СКБД.

1.8. Архітектура баз даних ANSI / SPARC

Тісний зв'язок між базою даних і пов'язаними з нею програмами призводить до проблем у підтримці програмного забезпечення. У ранніх СКБД більшість змін у визначенні бази даних призводили до змін у комп'ютерних програмах. Ця робота з перевірки коду схожа на рішення проблеми 2000 року, коли формат дат було змінено на чотиризначний. Іноді для кожної зміни в структурі БД доводилося перекомпілювати сотні комп'ютерних програм.

Загалом, системи баз даних складаються зі складних структур даних. Щоб створити ефективну та зручну для користувача систему, розробникам потрібно приховати складність системи від кінцевих користувачів. Процес приховування складних деталей системи від користувача відомий як «абстрагування даних». Цей підхід спрощує проєктування бази даних. Існує в основному три рівні абстракції даних, а саме: фізичний, логічний та рівень відображення. Ці рівні в деякому сенсі незалежні один від одного.

Концепція незалежності даних з'явилася для того, щоб полегшити завдання з підтримки і обслуговування програм. Незалежність даних означає, що база даних повинна мати ідентичність окрему від застосунків, які її використовують. Окрема ідентичність дозволяє змінювати визначення бази даних, не впливаючи на пов'язаних з нею застосунків. Наприклад, якщо до таблиці додається новий стовпчик, це ні як не впливає на

програми, які не використовують цей стовпчик. Цей поділ має бути ще більш чітким, якщо зміна впливає лише на фізичну реалізацію бази даних.

В середині 1970-х років концепція незалежності даних призвела до пропозиції комітетом ANSI/SPARC так званої Архітектури трьох схем [3,4]. Схема – опис всіх елементів даних і зв'язку між ними із зазначенням необхідних умов підтримки цілісності даних. Архітектура трьох схем включає три рівні опису бази даних:

- **зовнішня модель даних** – відображає частину предметної області з позиції відношення до даних кожного існуючого в організації типу користувачів;
- **концептуальна модель даних** – відображає узагальнене (або логічне) представлення даних предметної області, незалежне від типу обраної СКБД;
- **внутрішня модель даних** – відображає концептуальну схему в інформаційні структури таким чином, що вони можуть бути реалізовані в обраній цільовій СКБД.

Архітектура трьох схем схематично наведена на рис. 1.10 і поділена на такі рівні:

- фізичний;
- логічний (концептуальний);
- рівень відображення.

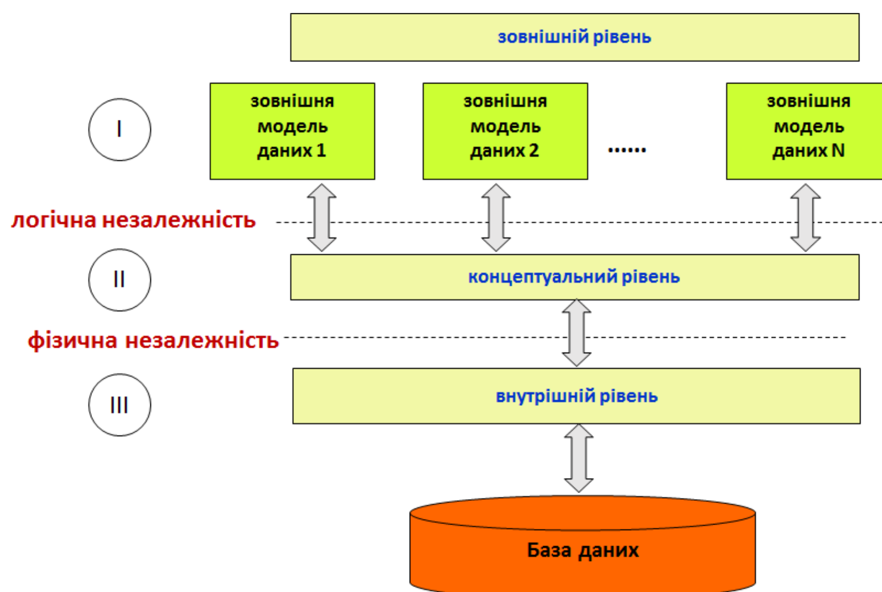


Рис. 1.10. Архітектура бази даних ANSI/SPARC

Фізичний рівень

Це найнижчий рівень абстракції даних. Фізичний рівень детально описує низькорівневі структури даних. Розробник бази даних може планувати структури даних з урахуванням майбутніх вимог до зберігання даних. Цей рівень описує наступні фактори, які розробники бази даних повинні знати під час проєктування бази даних, наприклад:

- як дані фактично зберігаються у фізичній пам'яті (наприклад, на магнітних стрічках, жорстких дисках тощо);
- який метод індексування файлів використовується (хешування, послідовний, дерево B+ тощо);

- яким буде розмір блоків зберігання в зовнішній пам'яті.

Логічний або концептуальний рівень

Це наступний рівень абстракції даних. Логічний рівень описує, які дані зберігаються в базі даних і зв'язки між ними. Він описує взаємопов'язані дані у вигляді набору простих відношень з відносно простою структурою. Розробник бази даних проєктує повну базу даних, використовуючи прості зв'язки, засновані на концепції нормалізації.

Розробник бази даних може планувати складну структуру фізичного рівня під час проєктування бази даних, але користувачеві не потрібно турбуватися про цю складність. Оскільки користувачі на логічному рівні не знають про складну структуру фізичного рівня, це називається фізичною незалежністю даних.

Рівень відображення

Це останній рівень абстракції даних. Він описує лише частину всієї бази даних. Кінцеві користувачі отримують доступ лише до частини бази даних на основі визначених ролей. Рівень відображення приховує складні деталі (зберігання та реалізація), а також логічну схему від кінцевого користувача. У випадку складних баз даних, складність зменшується шляхом створення/надання декількох відображень бази даних.

Приклад:

– Зовнішні відображення різних категорій користувачів

Таблиця 1.1. Зовнішнє відображення 1

ТабНомер	ПІБ	Вік	Зарплата

Таблиця 1.2. Зовнішнє відображення 2

ТабНомер	*ПІБ*	*Відділ*

– Концептуальна схема

Таблиця 1.3. Концептуальний рівень (концептуальна схема)

ТабНомер	ПІБ	Вік	Зарплата	Відділ

– Внутрішній рівень (внутрішня схема)

```
Struct Stuff {Int Stuff_No;
        Int Branch_No;
        Char Fname[15];
        Char Lname[15];
        Struct date Date_of_Birthday;
        Float Salary;
    }; index Stuff_no; Branch_No;
```

Незалежність даних в архітектурі трьох схем

Логічна незалежність – повна захищеність зовнішніх схем від змін, що вносяться у концептуальну схему.

Зміни концептуальної моделі (нові елементи даних, нові відношення) повинні не впливати на існуючі зовнішні представлення і не слід модифікувати існуючі прикладні програми при внесенні змін на логічному рівні.

Приклад:

Концептуальна схема університету (фрагмент)

```
Students(sid, fid, name, login, age, gpa)  --Студенти
Faculty(fid, fname, sal)                   --Викладачі
Courses(cid, cname, credits)               --Дисципліни
Enrolled(sid, cid, grade)                  --Студенти - Дисципліни
```

Зовнішня схема для студентів (подання)

```
Courseinfo(cid, fname, enrollment)    /*enrollment - к-ть зареєстрованих
студентів*/
```

Атрибут `enrollment` немає необхідності зберігати в концептуальній схемі, його можна розрахувати. Якщо концептуальну схему змінено, наприклад таблицю `Faculty` поділено на дві частини з міркувань безпеки персональної інформації:

```
Faculty_public(fid, fname)
Faculty_private(fid, sal)
```

то подання `Courseinfo` знов можна сформувати на підставі `Faculty_public`. Концептуальну схему змінено, зовнішню – ні. Це має назву логічна незалежність.

Фізична незалежність – захист концептуальної схеми від змін, що вносяться у внутрішню схему.

Незалежність фізичних даних відноситься до несприйнятливості логічної моделі до змін у внутрішній моделі. Внутрішні або фізичні зміни, такі як інша фізична послідовність записів, перехід від одного методу доступу до іншого, зміна алгоритму хешування, використання різних структур даних та використання нових пристроїв зберігання даних, не повинні впливати на логічну модель.

Прикладні програми не повинні залежати від використовуваних способів зберігання даних на носіях. На зовнішньому рівні єдиним ефектом, який може відчуватися, є зміна продуктивності. Насправді погіршення продуктивності є найпоширенішою причиною змін внутрішньої моделі.

Концептуальна схема ізолює користувачів від змін у фізичних деталях сховища. Ця властивість називається незалежністю від фізичних даних, або фізична незалежність.

Засіб, що забезпечує логічну і фізичну незалежність – СКБД.

1.9. Загальні властивості моделей даних

Загальновизнане визначення моделі даних як «абстрактне, самодостатнє, логічне визначення об’єктів, операторів та інших елементів, що в сукупності складають абстрактну машину доступу до даних, з якою взаємодіє користувач» належить К. Дейту. Таким чином, модель даних визначає логічні структури даних і способи організації зв’язків між ними; методи маніпулювання даними; правила обмеження цілісності даних. Можна також сказати, що модель даних задає спосіб опису схеми бази даних.

Незалежно від рівня абстракції, модель даних включає такі базові компоненти:

- способи опису даних: які базові (примітивні) типи можна використовувати, яким чином будувати складні структури даних із простіших (компонента структури);
- способи опису взаємозв'язків між об'єктами даних і засоби завдання обмежень цілісності (компонента цілісності);
- способи конструювання операцій, які можна використовувати в рамках моделі даних (компонента маніпуляції).

Конструктивні компоненти моделі даних

При побудові моделі повинні використовуватися тільки три типу конструктивних елементів: сутність, атрибут, зв'язок.

Кожен компонент інформації повинен моделюватися тільки одним з наведених конструктивних елементів для виключення надмірності і суперечливості опису:

– **сутність** – будь-який об'єкт (об'єкт, який ми можемо відрізнити від іншого), інформацію про який необхідно зберігати в базі даних. Сутність предметної області є результатом абстрагування реального об'єкта шляхом виділення і фіксації набору його властивостей;

– **атрибут** – пойменована характеристика сутності. З об'єктами пов'язано дві проблеми: ідентифікація і адекватний опис. Адекватний опис забезпечується коректним набором характеристик (атрибутів). Для ідентифікації використовують ім'я об'єкта. Ім'я – це прямий спосіб ідентифікації об'єкта. До непрямих методів ідентифікації об'єкта відносять визначення об'єкта через його властивості (характеристики або атрибути). Набір атрибутів, що однозначно визначають сутність називають ідентифікатором. Часто ідентифікатор сутності називають ключем. Завдання вибору ідентифікатора сутності є семантично суб'єктивним завданням.

– **Зв'язок** – це асоціювання двох або більше сутностей. Основне призначення зв'язків – можливість організації пошуку даних в базі даних і забезпечення цілісності даних.

Загальний процес побудови моделі даних

Процес побудови моделі включає послідовність завдань, які мають виконуватися ітеративним чином. Ці завдання зазвичай мають такий вигляд:

- визначення сутностей. Процес моделювання даних починається з ідентифікації об'єктів предметної області, яка має бути змодельована. Кожна сутність має бути логічно відокремленою від усіх інших;
- визначення властивостей кожної сутності. Кожен тип сутності можна відрізнити від усіх інших, оскільки він має кілька унікальних властивостей, які називаються атрибутами. Наприклад, сутність під назвою «клієнт» може мати такі атрибути, як ім'я, прізвище, номер телефону, тоді як сутність під назвою «адреса» може містити назву вулиці та номер будинку, місто, країну та поштовий індекс. Визначення атрибутів сутностей гарантує, що модель адекватно відображає предметну область;
- визначення відношення між сутностями. Модель даних має визначати характер відношення кожної сутності з іншими. У наведеному вище прикладі кожен клієнт «живе» за визначеною адресою;

- призначення ключів та вибір ступеню нормалізації, який врівноважує необхідність зниження надмірності з вимогами до продуктивності. Нормалізація – це метод організації моделей даних, у якому ідентифікатори, звані ключами, призначаються групам даних для представлення відношень між ними без повторення даних. Нормалізація зменшує обсяг дискового простору, який знадобиться базі даних, але це може коштувати продуктивності запитів. Детально нормалізація розглядається в розділі 4.

1.10. Класифікація моделей даних

Інфологічні моделі

Інфологічна модель – це семантична модель предметної області, тобто інформаційна модель найвищого рівня абстракції. Така модель створюється без орієнтації на будь-яку конкретну СКБД і модель даних. Терміни «концептуальна модель» та «інфологічна модель» є синонімами.

Конкретний вигляд і зміст концептуальної моделі бази даних визначається обраним для цього формальним апаратом. Зазвичай використовують графічні нотації, подібні до ER-діаграм.

Найчастіше концептуальна модель бази даних містить у собі:

- опис інформаційних об’єктів або понять предметної області та зв’язків між ними;
- опис обмежень цілісності, тобто вимог до допустимих значень даних.

Побудова концептуальної моделі є обов’язковим етапом проєктування бази даних і дозволяє однозначно окреслити межі предметної області, виділити її основні сутності, взаємозв’язки між ними.

Етап концептуального проєктування передбачає опис предметної області у термінах формальної мови. Широкого використання під час побудови концептуальної моделі бази даних набуло моделювання «Сутність-Зв’язок» (Entity-Relationship, ER). Елементи інформаційної моделі даних предметної області є вхідними даними для створення даталогічної моделі даних.

Даталогічні моделі

Під даталогічною розуміють модель, що відображає логічні взаємозв'язки між елементами даних безвідносно до їхнього змісту та фізичної організації. При цьому даталогічну модель розробляють з урахуванням конкретної реалізації СКБД, а також з урахуванням специфіки конкретної предметної області на основі її інфологічної моделі. Даталогічні моделі відносяться до наступних категорій:

а) Фактографічні моделі

Фактографічні даталогічні системи оперують фактичними відомостями, представленими у вигляді спеціальним чином організованих сукупностей записів даних визначеного формату. Характерною особливістю фактографічних систем є те, що вони працюють не з текстом, а з фактичними відомостями, які представлені у вигляді записів. Інформація, з якою працює фактографічна ІС, має чітку структуру, що дозволяє відрізняти

одне дане від іншого, наприклад, прізвище від посади людини, дату народження від зросту і т. і. Тому фактографічні системи, здатні однозначно вирішувати поставлені завдання і давати однозначні відповіді на запити. Фактографічні БД зберігають інформацію як факт, тобто як набір конкретних атрибутів, які цей факт цілком визначають. Це потрібує заздалегідь визначеній структури записів в БД.

б) Документальні моделі

Такі моделі використовуються в БД, у якій кожен запис відображає конкретний документ, містить його опис та іншу інформацію про нього. Документальні моделі оперують зі слабо структурованою інформацією, що орієнтована на вільний формат документа або тексту на його природній мові. За такими системами закріпився термін інформаційно-пошукові системи (ІПС). Інформаційний пошук – сукупність логічних і технічних операцій, що мають кінцевою метою знаходження документів, релевантних запитам користувача. Важливим показником пошуку є релевантність – відповідність змісту документа запиту в тому вигляді, в якому він був сформульований. Документальні моделі використовуються для завдань, які не передбачають однозначної відповіді на поставлене запитання.

Фізичні моделі

Фізична модель даних описує дані засобами конкретної СКБД. Обмеження, наявні в логічній моделі даних, реалізуються різними засобами СКБД, наприклад, за допомогою індексів, декларативних обмежень цілісності, тригерів, збережених процедур. При цьому рішення, ухвалені на рівні логічного моделювання, визначають деякі межі, у яких можна розвивати фізичну модель даних.

1.11. Еволюція моделей даних

На рис. 1.11 наведено історичний шлях еволюції моделей даних [1].

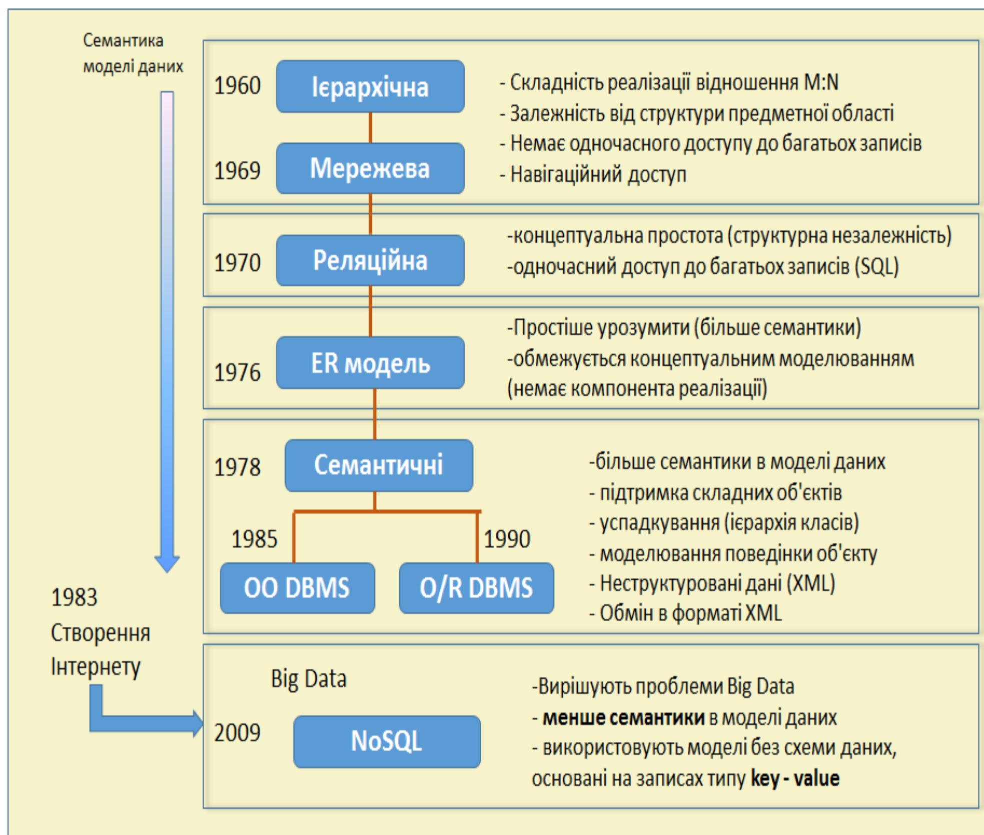


Рис. 1.11. Еволюція моделей баз даних

Як бачимо, моделі БД постійно розвиваються. Змінюються дані, змінюється характер обробки, створюються нові моделі. Спочатку з'явились ієрархічні моделі, які незабаром були замінені більш досконалими мережевими моделями. Завдяки роботам математика Едгара Ф. Кодда з'явились реляційні моделі, які стали домінуючими на ринку СКБД. Згодом з'явились об'єктно-орієнтовані і об'єктно-реляційні моделі. В зв'язку з розвитком технологій BigData з'явилось сімейство моделей NoSQL.

Якщо розглянути статистику популярності та поширеності СКБД (рис. 1.12), то побачимо, що на сьогодні безумовними лідерами є реляційні моделі. Тому в наступних розділах розглянемо саме реляційну модель даних.

423 systems in ranking, August 2024

Rank			DBMS	Database Model	Score		
Aug 2024	Jul 2024	Aug 2023			Aug 2024	Jul 2024	Aug 2023
1.	1.	1.	Oracle	Relational, Multi-model	1258.48	+18.12	+16.39
2.	2.	2.	MySQL	Relational, Multi-model	1026.86	-12.60	-103.59
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	815.18	+7.52	-105.64
4.	4.	4.	PostgreSQL	Relational, Multi-model	637.39	-1.52	+17.01
5.	5.	5.	MongoDB	Document, Multi-model	420.98	-8.85	-13.51
6.	6.	6.	Redis	Key-value, Multi-model	152.71	-4.06	-10.26
7.	7.	↑ 11.	Snowflake	Relational	135.97	-0.56	+15.34
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	129.83	-0.99	-10.09
9.	9.	↓ 8.	IBM Db2	Relational, Multi-model	123.00	-1.40	-16.23
10.	10.	10.	SQLite	Relational	104.79	-5.16	-25.13
11.	↑ 12.	↑ 12.	Apache Cassandra	Wide column, Multi-model	97.00	-2.12	-10.38
12.	↓ 11.	↓ 9.	Microsoft Access	Relational	96.37	-4.26	-33.97
13.	13.	↑ 14.	Splunk	Search engine	96.10	+3.18	+7.12

Рис. 1.12. Статистика поширеності СКБД і моделей даних[1]

Контрольні запитання

1. Визначте поняття «предметна область».
2. Які основні переваги та недоліки в опрацювання даних на основі технології баз даних?
3. Поясніть різницю між даними та інформацією. Наведіть приклади необроблених даних та інформації.
4. Дайте визначення поняттям: інформаційна система, предметна область.
5. В чому відмінність між даними і метаданими?
6. Опишіть та порівняйте інформаційні потреби на стратегічному, управлінському та оперативному рівнях управління.
7. Що являє собою трирівнева архітектура ANSI/SPARC?
8. В чому особливість рівня зовнішніх моделей?
9. Що означає логічна і фізична незалежність даних?
10. Що таке СКБД?
11. Які ви знаєте моделі даних для баз даних?
12. Назвіть етапи розвитку БД.
13. За якими критеріями можлива класифікація систем баз даних?
14. В чому особливість концептуального рівня абстракції даних?
15. В чому особливість фізичного рівня абстракції даних?

Тестові завдання

1. Які з наступних тверджень правильні?
   A. Ієрархічна модель була розроблена раніше реляційної моделі
   B. Об'єктно-орієнтована модель бази даних була розроблена раніше реляційної моделі
   C. Мережева модель бази даних була розроблена раніше реляційної моделі
   D. Модель «сутність-зв'язок» була розроблена раніше за реляційну модель
2. Що з наведеного нижче описує процес створення бази даних?
   A. Збір великого обсягу даних, створених цифровими процесами та пристроями
   B. Створення файлу для зберігання даних про одну сутність
   C. Створення сутностей, атрибутів і зв'язків між таблицями даних
   D. Перетворення даних у базі даних у формат, який неможливо розшифрувати звичайними програмами
3. Який із типів моделей даних не підтримують СКБД?
   A. Нелінійні
   B. Ієрархічні
   C. Мережеві

[1] https://db-engines.com/en/ranking

D. Реляційні

4. Що таке база даних? (виберіть правильний варіант)
   A. Реалізована за допомогою комп'ютера інформаційна структура, що відображає стани об'єктів та їхні відношення
   B. Мінімальна іменована структурна одиниця даних
   C. Сукупність нормалізованих відношень, що логічно взаємопов'язані та відображають деяку предметну область
   D. Універсальний програмний засіб, призначений для обробки інформації
5. _____ – це характеристика або властивість об'єкта.
   A. Метадані
   B. База даних
   C. Плоский файл
   D. Атрибут
6. Що з наведеного нижче визначає СКБД?
   A. Сукупність даних, які відповідають одному запису
   B. Файл, що використовується для зберігання даних про один об'єкт;
   C. Особа або група осіб, відповідальна за базу даних;
   D. Програма, за допомогою якої користувачі взаємодіють з даними, що зберігаються в базі даних.
7. Що таке база даних?
   A. Набір пов'язаних таблиць і даних
   B. Програмне забезпечення для управління файлами та папками
   C. Вебзастосунок для аналізу даних
   D. Мова програмування для візуалізації даних
8. Що таке РСКБД?
   A. Реляційна система керування базами даних
   B. Рекурсивна система керування базами даних
   C. Релятивна система керування базами даних
   D. Надійна система керування базами даних
9. Що вірно до властивостей бази даних
   A. Вона забезпечує меншу логічну незалежність даних, ніж файлові системи, які вона замінила
   B. Забезпечує як фізичну, так і логічну незалежність даних
   C. Елементи даних зберігаються саме так, як вони представлені користувачеві бази даних
   D. Забезпечує рівні абстракції бази даних
   E. База даних завжди управляється системою керування базами даних
10. Які функції можна виконувати в СКБД? (Виберіть усі правильні відповіді.)
    A. Пошук даних
    B. Продаж даних
    C. Сортування даних
    D. Відображення даних

11. Існує багато переваг використання підходу, що базується на базах даних. Що з наведеного нижче не вірно при використанні підходу на основі баз даних?
    A. Надлишковість даних може бути зменшена
    B. Можна уникнути неузгодженості даних
    C. Обчислювальні ресурси, необхідні для обробки даних, можуть бути скорочені
    D. Можна забезпечити дотримання стандартів обробки даних
12. Що таке OLTP?
    A. Online Transaction Processing
    B. Online Load Testing Process
    C. Offline Transaction Processing
    D. Offline Load Testing Process
13. Що не є характеристикою OLTP?
    A. OLTP фокусується на обробці транзакцій та оновленні даних в режимі реального часу
    B. OLTP фокусується на складних аналітичних операціях та аналізі історичних даних
    C. OLTP використовує реляційну модель бази даних
    D. OLTP використовує багатовимірну модель бази даних
14. Зважаючи на незалежність даних, що з наведеного нижче не відповідає дійсності?
    A. Користувачеві не потрібно знати розмір кожного запису файлу
    B. Користувачеві не потрібно знати, чи відсортовано файл
    C. Користувачеві не потрібно знати назву файлу, який містить відповідну інформацію
    D. Користувачеві не потрібно знати порядок полів у записах
    E. Все вищезазначене
15. ANSI/SPARC модель архітектури СКБД складається з трьох рівнів: фізичного, зовнішнього та концептуального. Яке з наступних тверджень не є правильним?
    A. Внутрішнє представлення бази даних має справу з фізичними записами та специфічними для пристрою питаннями, такими як розміри доріжок та секторів
    B. Концептуальне представлення бази даних – це уявлення спільноти про базу даних, тобто уявлення про базу даних такою, якою вона є насправді
    C. Зовнішнє представлення бази даних – це уявлення окремого користувача, тобто, часто це обмеження концептуального представлення
    D. Може існувати лише одне зовнішнє представлення, лише одне концептуальне представлення і лише одне внутрішнє представлення

2. КОНЦЕПТУАЛЬНЕ ПРОЄКТУВАННЯ БАЗ ДАНИХ

Класична методологія проєктування БД заснована на трирівневій моделі ANSI/SPARC. Процес проєктування бази даних можна представити у вигляді послідовності етапів (рис. 2.1):

1. З'ясовуються вимоги до інформаційної системи з боку окремих груп користувачів (побудова функціональної моделі).
2. Ці вимоги збираються в концептуальну модель бази даних (концептуальне проєктування).
3. Використовуючи конкретний тип СКБД (наприклад, реляційну) створюється логічна модель. Для кожної групи користувачів логічна модель перетворюється в зовнішні моделі.
4. На підставі логічної моделі, враховуючи особливості програмно-апаратного забезпечення, створюється внутрішня модель.

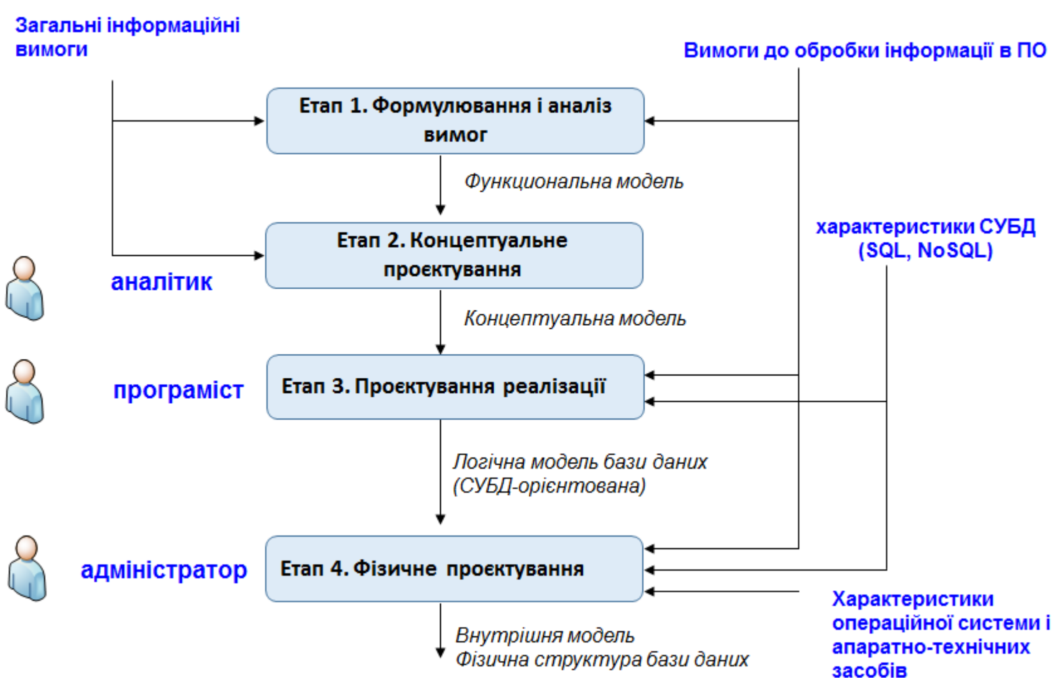


Рис. 2.1. Основні етапи проєктування баз даних

2.1. Побудова функціональної моделі

На етапі формулювання і аналізу вимог створюється функціональна модель. Завданням цього етапу є дослідження структури системи та логічних взаємозв'язків її елементів, причому тут не розглядаються питання, пов'язані з реалізацією на конкретній платформі.

Для цілей функціонального моделювання найчастіше й найефективніше застосовується діаграми потоків даних (Data Flow Diagrams, DFD), спільно зі словниками даних і специфікаціями процесів.

Діаграми потоків даних являють собою ієрархію функціональних процесів, пов'язаних потоками даних. Мета такого представлення – продемонструвати, як кожен процес перетворює свої вхідні дані на вихідні, а також виявити відносини між цими процесами. Джерела інформації (зовнішні сутності) породжують інформаційні потоки (потоки даних), що передають інформацію до підсистем або процесів. Ті, своєю чергою, перетворюють інформацію і породжують нові потоки, які передають інформацію до інших процесів чи підсистем, сховищ даних або зовнішніх сутностей – споживачів інформації. Структури потоків даних і визначення їхніх компонентів зберігаються й аналізуються в словнику даних. Кожна логічна функція (процес) може бути деталізована за допомогою DFD нижнього рівня. Коли подальша деталізація перестає бути корисною, переходять до вираження логіки функції за допомогою специфікації процесу.

Основними компонентами діаграм потоків даних є:

- зовнішні сутності (об'єкти за межею системи);
- системи та підсистеми (ієрархія підсистем);
- процеси (об'єкти що перетворюють інформацію);
- сховища даних (об'єкти для зберігання інформації);
- потоки даних (інформація, що циркулює в системі).

Процес перетворює вхідні потоки у вихідні відповідно до дії, заданою ім'ям процесу. Це ім'я має містити дієслово в невизначеною формі з подальшим доповненням (наприклад, ОБЧИСЛИТИ МАКСИМАЛЬНУ ВИСОТУ). Процес позначається прямокутником із заокругленими краями (рис. 2.2). Крім того, кожен процес повинен мати унікальний номер для посилань на нього всередині діаграми. Цей номер може використовуватися спільно з номером діаграми для отримання унікального індексу процесу у всій моделі. Інформація в полі фізичної реалізації показує, який підрозділ організації, програма чи апаратний пристрій виконує даний процес.

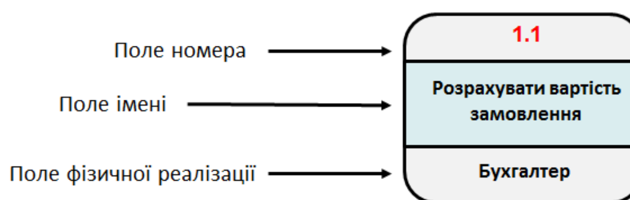


Рис. 2.2. Позначення процесу на діаграмі

Потоки даних є механізмами, які показують передачу інформації від одного процесу іншому (рис. 2.3). На діаграмі вони зазвичай відображаються спрямованою стрілкою, яка показує напрямок руху інформації.

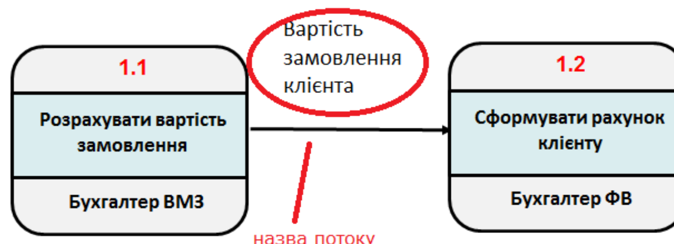


Рис. 2.3. Позначення потоку даних на діаграмі

Сховище (накопичувач) даних – це абстрактний пристрій для зберігання інформації, яку можна в будь-який момент помістити в сховище і через деякий час витягнути, причому способи переміщення і вилучення можуть бути будь-якими (рис. 2.4).

Сховище даних в загальному випадку є прообразом майбутньої бази даних, і опис даних, що зберігаються в ньому має відповідати моделі даних.

Рис. 2.4. Позначення сховища на діаграмі

Зовнішня сутність – це матеріальний об'єкт або фізична особа, що є джерелом або приймачем інформації, наприклад, замовники, персонал, постачальники, клієнти, склад (рис. 2.5). Визначення деякого об'єкту або системи в якості зовнішньої сутності вказує на те, що вона знаходиться за межами модельованої системи.

Рис. 2.5. Позначення зовнішньої сутності на діаграмі

У процесі побудови моделі створюється словник даних (Data Dictionary), у якому зберігається й аналізується склад потоків і накопичувачів даних, взаємозв'язок окремих елементів потоків і накопичувачів даних. Словник даних являє собою певним чином організований список усіх елементів даних системи з їхніми точними визначеннями, що дає змогу різним категоріям розробників (від системного аналітика до програміста) мати загальне розуміння всіх вхідних і вихідних потоків і компонентів сховищ. Наприклад, під час моделювання документообігу вводяться відомості про структуру і реквізитний склад документів. Словник даних використовується на наступному етапі проєктування бази даних.

Побудова ієрархії діаграм потоків даних

Спочатку створюється так звана контекстна діаграма, яка моделює систему найбільш загальним чином. Контекстна діаграма відображає інтерфейс системи з зовнішнім світом, а саме, інформаційні потоки між системою і зовнішніми сутностями, з якими вона повинна бути пов`язана. Вона ідентифікує ці зовнішні сутності в єдиному процесі, який відображає головну ціль системи (наскільки це можливо). Кожний проєкт повинен мати тільки одну контекстну діаграму, при цьому нема необхідності в нумерації її єдиного процесу. Після цього створюються діаграми першого, а за необхідністю і другого рівнів. Приклад контекстної діаграми наведено на рис. 2.6 .

DFD першого рівня будується як декомпозиція процесу, який показаний на контекстній діаграмі. Діаграма першого рівня містить множину процесів, які в свою чергу можу бути представлені в вигляді діаграм нижнього рівня. Процес декомпозиції триває до тих пір, поки процеси можуть бути ефективно описані за допомогою коротких (до однієї сторінки) специфікацій обробки (специфікацій процесів). Зв'язок між рівнями – за допомогою структурованої нумерації процесів.

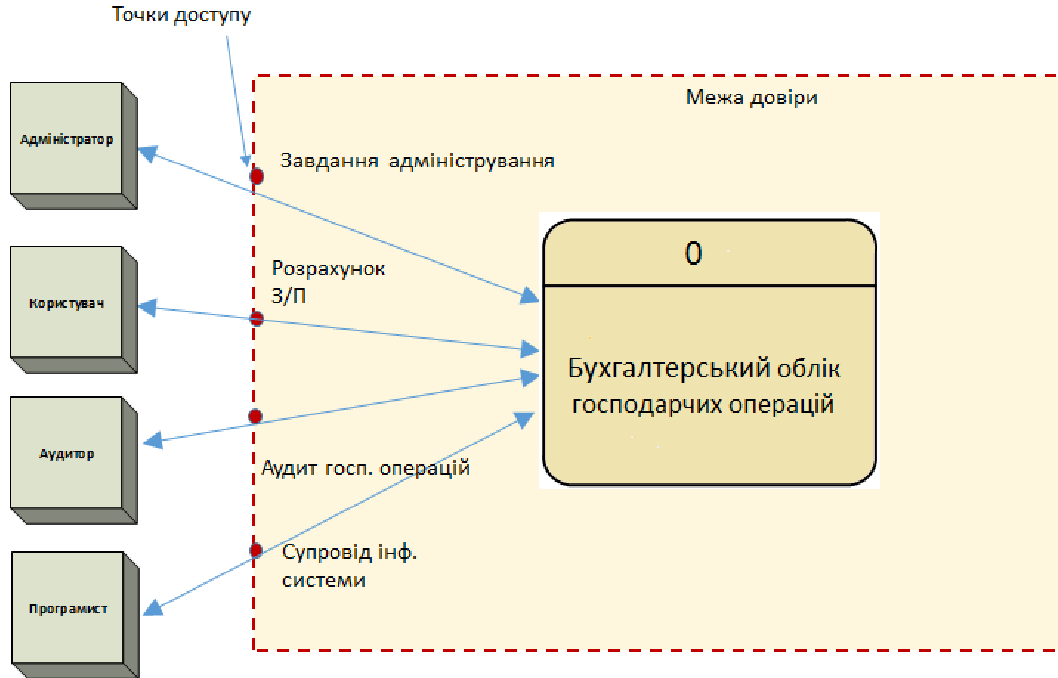


Рис. 2.6. Приклад контекстної діаграми

Приклад діаграми першого рівня наведено на рис. 2.7.

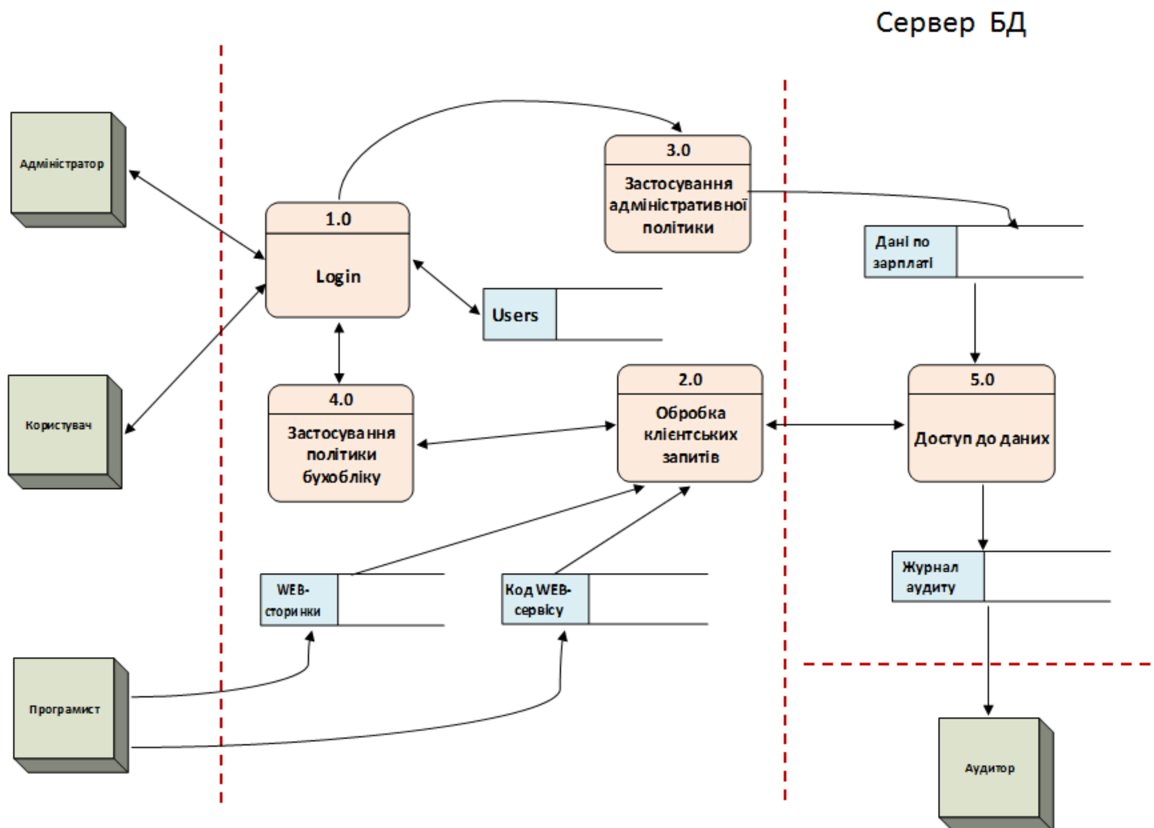


Рис. 2.7. Приклад DFD діаграми першого рівня

Ознаки закінчення процесу декомпозиції:

1. Наявність у процесу відносно невеликої кількості вхідних і вихідних потоків даних (2–3 потоки).
2. Можливість опису перетворення даних процесів у вигляді послідовного алгоритму.
3. Виконання процесом єдиної логічної функції перетворення вхідної інформації у вихідну.
4. Можливість опису логіки процесу за допомогою специфікацій невеликого об`єму (не більше 20–30 рядків).

2.2. Концептуальне проєктування та концептуальна модель

2.2.1. Завдання концептуального проєктування

Завданням етапу концептуального проєктування є формування абстрактної моделі незалежної від конкретної СКБД, тобто концептуальної моделі даних. Концептуальна модель – це абстрактна модель високого рівня, побудована шляхом об'єднання інформаційних вимог користувачів. Результатом цього етапу проєктування є формалізований опис предметної області.

Концептуальна модель має наступні чотири основні характеристики:

1. Вона може правдиво і повністю відображати реальний світ, включаючи зв'язок між об'єктами, як справжня модель реального світу.
2. Вона легка для розуміння, що дозволяє обговорювати її з користувачами, які не знайомі з базою даних.
3. Легко змінюється, коли змінюється середовище застосування і вимоги до застосування.
4. Її легко перетворити на реляційну модель даних.

Створена концептуальна модель даних є джерелом інформації для етапу логічного проєктування бази даних. Побудова концептуальної схеми БД дає змогу більш повно оцінити специфіку предметної області та уникнути можливих помилок на стадії проєктування схеми реляційної БД. Також, на етапі концептуального проєктування створюється важлива документація, яка може виявитися дуже корисною не тільки під час проєктування схеми реляційної БД, а й під час експлуатації, супроводу та розвитку вже заповненої БД.

У 1976 році був запропонований апарат формалізації концептуальної моделі – підхід **сутність-зв'язок** (E-R Entity-Relationship). Цей підхід швидко став одним з найпоширеніших і зараз є загальним підходом до опису інформаційних структур. Результатом моделювання є так звана ER-діаграма. ER-діаграма – інструмент семантичного моделювання даних, який використовується для досягнення мети формального опису або представлення моделі даних. Концепція бази даних, представлена за допомогою ER-діаграми, є дуже інтуїтивно зрозумілою і легко сприймається користувачами.

Створена концептуальна модель дозволяє визначити «схему». **Схема** – постійний опис структури даних. Тому, коли зафіксовано правильне відображення предметної області в концептуальній моделі, ER-діаграму можна назвати її схемою.

ER-діаграма також може бути використана для документування існуючої бази даних шляхом її зворотного проєктування.

2.2.2. Створення ER-діаграми

Основними поняттями ER-моделі є сутність, зв'язок та атрибут [4].

Сутність – це реальний або уявний об'єкт, інформація про який має зберігатися і бути доступною.

Атрибут сутності – це будь-яка деталь, яка слугує для уточнення, ідентифікації, класифікації, числової характеристики або вираження стану сутності

Зв'язок – це асоціація, що встановлюється між двома типами сутностей.

У побудові загальної концептуальної моделі даних виділяють низку етапів.

- Формулювання сутностей, що описують локальну частину предметної області БД що проєктується.
- Опис атрибутів, що становлять структуру кожної сутності і визначають її властивості.
- Виділення ключових атрибутів сутності.
- Аналіз, додавання або вилучення неключових атрибутів.
- Визначення зв'язків між сутностями. Видалення надлишкових зв'язків.

2.2.3. Створення діаграми «тільки сутності» (Entity-Only)

Сутність це реальний предмет або явище, що має суттєвої значення для розглянутої предметної області і про яку необхідно зберігати інформацію.

Ознаки сутності:

- Сутність – це «річ», яку можна чітко ідентифікувати.
- Сутність представляє щось, що існує в реальному світі, і ми хочемо зберігати певні дані про неї.

Оскільки сутність представляє об'єкт реального світу, слова «сутність» і «об'єкт» часто використовуються як взаємозамінні. Сутність може мати реалізацію в вигляді фізичного об'єкта (будинок, автомобіль, гравець) або логічною уявленням (робота, родина, сервер).

Об'єкти що можуть бути сутністю:

- об'єкт, який матиме багато екземплярів (тобто станів) у базі даних;
- об'єкт, який буде складатися з декількох атрибутів;
- об'єкт, стан якого ми намагаємося зберігати.

Приклади об'єктів, що не можуть бути сутністю:

- користувач системи бази даних;
- вивід системи бази даних (наприклад, звіт).

Існують різні системи нотації (позначень) ER-діаграми. Найбільш популярні нотації Чена і нотація Мартина (яку ще називають Crow's Foot – гусяча лапка) та нотація UML.

Деякі концептуальні поняття моделювання баз даних можуть бути виражені тільки за допомогою нотації Чена. Однак, оскільки основна увага приділяється проєктуванню та впровадженню баз даних, використовуються нотація Crow's. Foot. За допомогою такої нотації в моделі можна представляти тільки те, що може бути реалізовано. Крім того, ER-діаграма в нотації Мартина більш компактна (рис. 2.8).

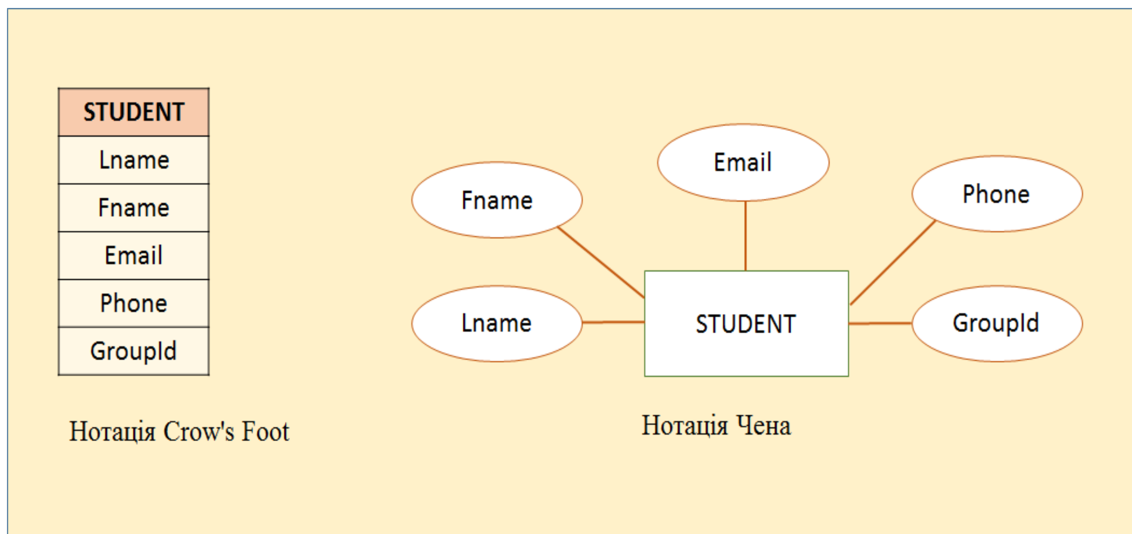


Рис. 2.8. Порівняння розміру моделі в різних нотаціях

Сучасні інструментальні засоби проєктування як правило використовують саме нотацію Crow's Foot. У цій нотації сутність зображується прямокутником, у який вписано назву сутності. На етапі проєктування атрибутів, вони записуються прямо всередині символу прямокутника, в окремій секції (рис. 2.9).

Рис. 2.9. Відображення сутностей моделі

Екземпляри сутності – це набір сутностей одного типу, які мають однаковий набір властивості.

Приклад екземплярів сутностей: на рис. 2.10 це набір записів відповідних таблицях.

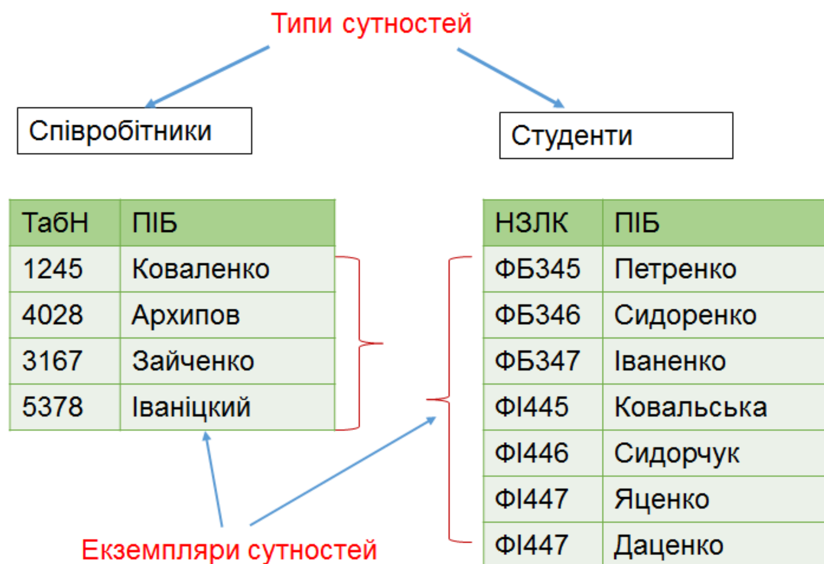


Рис. 2.10. Приклади типів і екземплярів сутностей

Сильні і слабкі сутності

Незалежна (сильна) сутність - не залежить від інших сутностей і може бути визначена без зв’язку з іншими сутностями. Її часто називається "батьківською", оскільки в неї в підпорядкуванні зазвичай перебувають слабкі сутності. Незалежні сутності мають власний набір атрибутів, який дає змогу ідентифікувати кожен екземпляр сутності.

Слабка (залежна) сутність – це сутність, яка залежить від існування екземпляра сильної сутності. Вона не може бути ідентифікована за своїми власними атрибутами. Прикладом цього є така сутність, як запис у результатах сесії. Без зазначення екземплярів інших сутностей (Студент, Дисципліна) сам запис буде безглуздим.

Визначення кандидатів в сутності

Важливим етапом побудови концептуальної моделі є обстеження предметної області що автоматизується, під час якого спеціалісти проводять опитування співробітників установи. Під час обстеження вирішується завдання функціонального моделювання (DFD діаграми), де визначені потоки даних надають уявлення щодо сутностей. Питання під час обстеження сформульовані таким чином, щоб визначити основні характеристики предметної області. Ось приклад відповіді співробітника:

> «Як **представник по роботі з клієнтами**, я відповідаю за десять **клієнтів**. Кожен із моїх клієнтів домовляється про **зустріч**, щоб прийти до **виставкової зали**, щоб переглянути **товари**, які ми можемо запропонувати на поточний **сезон**. Частина моєї **роботи** полягає в тому, щоб відповідати на будь-які питання про наші **товари** та давати рекомендації щодо найпопулярніших товарів. Як тільки вони ухвалюють рішення про товари, які хотіли б придбати, я оформляю **замовлення на продаж** для клієнта. Потім я передаю замовлення на продаж своєму **помічнику**, який оперативно оформляє замовлення та надсилає його клієнту».

З відповіді можна скласти список сутностей:

- представник по роботі з клієнтами,

- клієнти,
- зустріч,
- виставкова зала
- товари,
- сезон,
- робота (функціональні обов’язки),
- замовлення на продаж,
- помічник.

Далі треба створити діаграму «тільки сутності» (рис. 2.11).

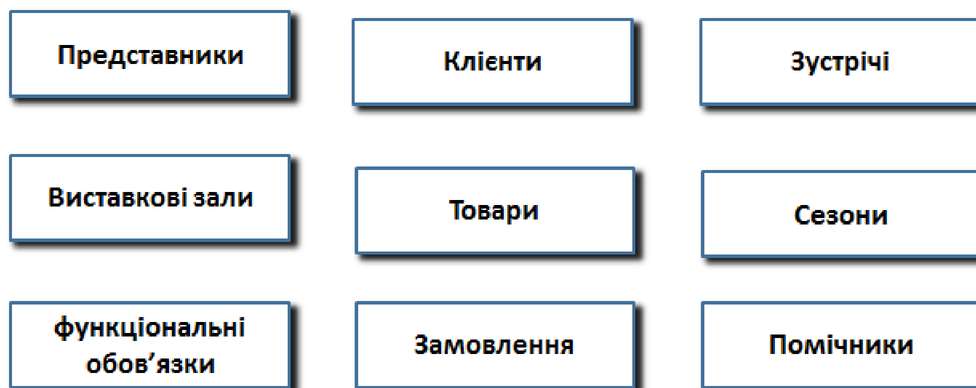


Рис. 2.11. Діаграма «тільки сутності»

Не завжди простий список сутностей дозволяє адекватно відобразити властивості предметної області. Розробник повинен провести більш детальний аналіз.

Приклад. Предметна область Бібліотека (облік читачів).

Основні предметно-значимі сутності:

- Книги. Атрибути сутності – автор книги, назва, рік видання, ціна, чи являється новим виданням, коротка анотація.
- Читачі. Атрибути сутності – номер читацького квитка, ПІБ, адреса і телефон читача.

На перший погляд достатньо двох сутностей Книги і Читачі. Однак аналіз вимог до функцій системи:

- вибрати книги, які знаходяться у читачів, чи окремого читача;
- вибрати читачів, які брали ту чи іншу книгу з вказанням дати взяття книги та дати здачі книги читачем;
- вибрати авторів, книги яких користуються найбільшим попитом.

змушує визначити ще такі сутності:

- Журнал реєстрації.
- Автори книг.

2.2.4. Визначення атрибутів сутностей

Сутність представлена набором атрибутів; тобто властивостями, якими володіють усі екземпляри сутності і які значимі для розглянутої предметної області.

В загальному випадку, атрибути призначені для класифікації, ідентифікації, кількісної характеристики і вирази стану сутності. Окремий атрибут визначається типом (числовий, текстовий, логічний, тимчасової і ін.), А також значенням, яке він приймає з деякої множини, званої доменом.

Набір атрибутів, що однозначно ідентифікує кожен екземпляр сутності, називають ключовим. В ідеалі, ідентифікатор сутності складається лише з одного атрибута. Однак можна використовувати складений ідентифікатор, що складається з більш ніж одного атрибуту.

Правила для атрибутів сутності:

- Ім'я атрибута – це іменник або словосполучення в однині (наприклад, Код постачальника, Вік, Мінімальна ціна товару, Ім'я, Прізвище).
- Ім'я атрибуту має бути унікальним. Жодні два атрибути одного і того ж типу сутності не можуть мати однакове ім'я. Бажано, щоб жодні два атрибути у всіх типах сутностей не мали однакового імені. Наприклад атрибут Адреса для різних сутностей Співробітники і Клієнти: EmployeeAddress, ClientAddress.
- Щоб зробити ім'я атрибута унікальним і для ясності, ім'я кожного атрибута має відповідати стандартному формату, який встановлюється кожною організацією (Customers.CustomerID).
- Значення кожного атрибута належить визначеному домену.

Домени

Домен – це множина можливих значень для даного атрибута. Наприклад, домен для атрибута Оцінка має вигляд (0,100), оскільки найнижче можливе значення дорівнює 0, а найвище – 100. Для атрибута «стать» має лише два значення: Ч або Ж (або інший еквівалентний код). Домен для атрибута «дата найму на роботу» складається з усіх дат, які вписуються в діапазон (наприклад, від дати заснування компанії до поточної дати). Атрибути можуть мати спільний домен. Наприклад, адреса студента та адреса професора мають один і той самий домен з усіх можливих адрес.

Типи атрибутів

В концептуальній моделі визначають такі типи атрибутів [7]:

- **Обов'язковий атрибут** – повинен мати значення для кожного екземпляра сутності. Значення необов'язкового атрибута може бути невідомим.
- **Атомарний атрибут** – атрибут, який не має складових частин. Наприклад, вік, стать і сімейний стан класифікуються як прості атрибути. Щоб полегшити виконання запитів, доцільно замінити складні атрибути на набір простих атрибутів.
- **Складений атрибут** – це атрибут, який можна розбити на складові частини (рис. 2.12). Наприклад, ПІБ особи можна розбити на частини: ім'я, по батькові та прізвище. Атрибут АДРЕСА можна розбити на вулицю, місто та поштовий індекс.

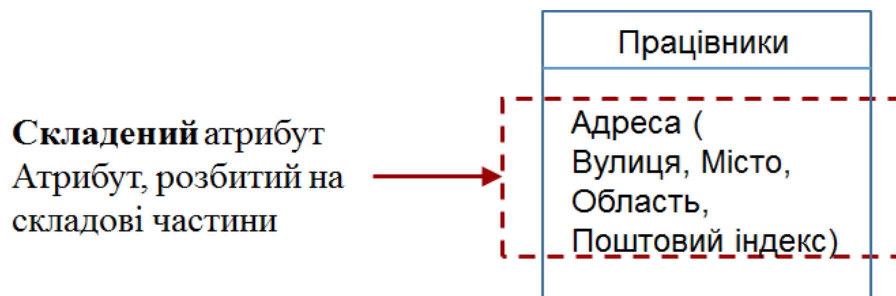


Рис. 2.12. Приклад складеного атрибуту

- **Однозначний атрибут** – це атрибут, який може мати лише одне значення. Наприклад, особа може мати лише один індивідуальний податковий номер, а виготовлена деталь може мати лише один серійний номер. Однозначний не обов'язково є простим атрибутом. Наприклад, серійний номер деталі (на зразок SE-08-02-189935) є однозначним, але він є складеним атрибутом, оскільки його можна розбити на регіон, в якому деталь було вироблено (SE), завод у цьому регіоні (08), зміну на заводі (02) та номер деталі (189935). Як правило однозначний атрибут може ідентифікувати екземпляр сутності.
- **Багатозначні атрибути** – це атрибут, який може мати кілька значень для одного екземпляра сутності. Наприклад, особа може мати декілька дипломів про вищу освіту, а установа може мати кілька різних телефонів, кожен з яких має свій номер. Аналогічно, колір автомобіля може бути поділятися на багато кольорів для даху, кузова та оздоблення. На рис. 2.13 атрибут «Нагороди» багатозначний – працівник може мати кілька нагород. Нотація Crow's Foot не ідентифікує багатозначні атрибути, оскільки в реляційній моделі значення кожного атрибуту повинно бути атомарним. Отже, якщо існують багатозначні атрибути, проєктувальник повинен прийняти рішення про один з двох можливих варіантів дій:
  - Створити кілька нових атрибутів сутності, по одному для кожного компонента багатозначного атрибута.
  - Створіть нову сутність, що складається з компонентів багатозначного атрибута. Потім ця нова сутність буде пов'язана з оригінальною сутністю у зв'язком «один до багатьох» (1:N).
- **Обчислюваний (похідний) атрибут** – це атрибут, значення якого можна обчислити на основі значень інших атрибутів. На рис. 2.13 це атрибут «Років стажу», значення якого можна обчислити за допомогою атрибута «Дата прийому».

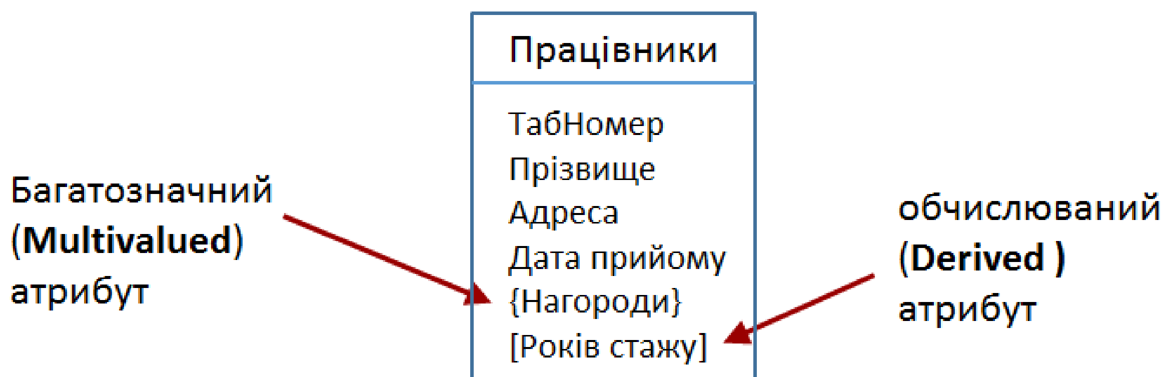


Рис. 2.13. Приклади багатозначних і похідних атрибутів

Похідний атрибут не обов'язково повинен фізично зберігатися в базі даних; натомість його можна отримати за допомогою алгоритму. Наприклад, вік працівника може бути знайдений шляхом обчислення різниці між поточною датою і датою народження. Аналогічно, загальну вартість замовлення можна отримати, помноживши кількість на ціну за одиницю. Нотація Crow's Foot не має способу відрізнити похідний атрибут від інших атрибутів. Рішення зберігати похідні атрибути залежить від вимог до обробки та обмежень, накладених на конкретну програму, наприклад, на швидкість обробки. Розробник повинен мати можливість збалансувати схему відповідно до таких обмежень. У табл. 2.1 показано переваги та недоліки зберігання (або не зберігання) похідних атрибутів у базі даних.

Таблиця. 2.1. Переваги та недоліки зберігання похідних атрибутів

Критерій	Зберігати похідний атрибут	Не зберігати похідний атрибут
Переваги	• Зменшення навантаження на процесор. • Зменшення часу доступу до даних. • Значення даних легко доступні. • Можна відстежувати історію змін	• Заощаджує місце на диску. • Обчислення завжди дає поточне значення
Недоліки	• Потребує постійної актуалізації, особливо якщо будь-які значення що використовуються в розрахунку, змінюються	• Додаткове навантаження процесору. • Збільшення часу доступу до даних. • Додає складності кодуванню запитів

Продовження прикладу обстеження предметної області. Визначення атрибутів сутності.

Приклад відповіді співробітника:

"Ну, спочатку я вводжу всю інформацію про **клієнта**, таку як його **назва**, **адреса**, **номер телефону** та **адреса електронної пошти**. Потім я вводжу **товари**, які клієнт хоче придбати. Після того, як я ввів усі товари, я підбиваю **підсумки** і все готове". О, я забув згадати, що я вводжу **номер факсу** клієнта та **адресу доставки** - якщо він має".

З такої відповіді можна визначити характеристики цієї сутності (рис. 2.14) – це іменники, вони зазвичай стоять в однині («номер телефону», «адреса»). На відміну від них, іменники, що позначають **інші сутності**, зазвичай знаходяться в присвійній формі («номер телефону **клієнта»**, «адреса **компанії**»).

Клієнти

назва	Name
адреса	**Address**
номер телефону	**Phone_Number**
адреса електронної пошти	**Email**
номер факсу	**Fax**
адресу доставки	**Shipping_Address**
підсумки	???

Рис. 2.14. Атрибути сутності Клієнти

Для розглянутої раніше предметної області Бібліотека основні предметно-значимі сутності: Книги, Читачі, Журнал, Автори.

Основні предметно-значимі атрибути сутності книги – автор книги, назва, рік видання, ціна, чи являється новим виданням, коротка анотація.

Аналіз сутності Книги дозволяє визначити похідний атрибут – ознака, що це нове видання. Його можна визначити за роком видання. Крім того, з часом його треба буде змінити. Тому краще зберігати рік видання, а цю ознаку розраховувати. Більш детальний аналіз предметної області дозволяє визначити атрибут, за яким обліковуються книги в бібліотеці – це номер, що записується на 17 сторінці книги. Також важливою характеристикою книги є ISBN – її номер за міжнародною класифікацією.

Аналіз вимоги до функції системи «вибрати читачів, які брали ту чи іншу книгу з вказанням дати взяття книги та дати здачі книги читачем» змушує створити додатковий атрибут в сутності Журнал – дата, коли читач повинен повернути книгу. Це дозволяє визначити «боржників». Значення атрибуту є не обов'язковим.

Отримуємо такий набір сутностей і атрибутів предметної області Бібліотека:

- Книги: {назва, рік видання, ціна, анотація, автор, рік видання, Id конкретного екземпляру книги, ISBN}
- Читачі: {номер читацького квитка, ПІБ, адреса, телефон}
- Журнал: {книга, читач, дати взяття, дати здачі, дата до якої повинен повернути книгу}
- Автори: {ПІБ, ….. }

Уточнення повного списку атрибутів сутностей

Аналіз повного набору атрибутів дозволяє знайти ті, назві яких збігаються. Якщо такі збіги є, визначте, чи всі вони представляють одну й ту саму характеристику. Якщо так, залиште одну назву. Наприклад, припустимо, що у списку характеристик присутні

«Product #», «Product No.» та «Product Number». Очевидно, що всі ці елементи представляють ту саму характеристику, і вам потрібна лише одна з них у вашому списку. Виберіть той, який ясно, повно і недвозначно передає задуманий сенс, і викресліть пункти, що залишилися, зі списку характеристик.

В іншому випадку визначте, що є кожним екземпляром назви. Часто можна виявити, що дублююче ім'я представляє той самий тип характеристики, але має бути пов'язаним з іншою сутністю. У цьому випадку ви перейменовуватимете дублікат, щоб відобразити його відношення до відповідної сутності.

Приклад: ви виявляєте, що перше входження «Назва» є характеристикою об'єкта «Клієнти», друге – характеристикою об'єкта «Співробітники», а третє – характеристикою об'єкта «Контакти». Усуньте дублювання, перейменувавши кожне входження «Назва»: «Назва клієнта», «Назва співробітника» та «Назва контакту». (Client Name, Employee Name, Contact Name).

Популярні назви, що дублюються: «Ім'я», «Адреса», «Місто», «Поштовий індекс», «Номер телефону» та «Адреса електронної пошти». Необхідно перейменувати кожен такий атрибут, щоб відобразити його зв'язок із конкретним типом сутності, забезпечуючи цим максимально точний список атрибутів.

Визначення ключових атрибутів

Дані, що зберігаються в БД, в процесі обробки треба знаходити. Атрибут, який використовується для пошуку конкретного екземпляра сутності, називається ключем. Коли створюється ER-модель, можна знайти деякі атрибути, які природно виглядають як ключі.

Ключі мають вирішальне значення для структури таблиці :

- Вони забезпечують точну ідентифікацію кожного запису в таблиці.
- Вони допомагають встановлювати та забезпечувати різні типи цілісності. Ключі є основним компонентом цілісності на рівні сутності та цілісності на рівні зв'язку сутностей.
- Вони служать для встановлення зв'язку між сутностями.

Для кожної сутності треба визначити ключі. Це допоможе гарантувати, що надлишкові дані в кожній таблиці будуть мінімальними, і що зв'язки між таблицями будуть коректними.

Можлива ситуація, коли для сутності можна визначити кілька різних ключів. Всі вони мають назву ключі-кандидати. Вимоги до ключа-кандидата:

- Це не може бути багатозначний атрибут, але він може складатись з кількох атомарних атрибутів (складений ключ).
- Атрибут повинен містити унікальні значення.
- Його значення не може бути необов'язковим повністю або частково. Необов'язкове значення означає, що в певний момент воно може бути Null.
- Його значення не може призвести до порушення правил безпеки або конфіденційності організації. Такі значення, як паролі та індивідуальний податковий номер не підходять для використання в якості ключа-кандидата.

- Можна використовувати комбінацію атрибутів в якості ключа-кандидата, якщо кожен атрибут вносить свій внесок у визначення унікального значення. Однак він повинен містить мінімальну кількість атрибутів, необхідну для визначення унікальності.

У деяких випадках недоцільно використовувати семантично визначений первинний ключ, наприклад:

- Коли складовий первинний ключ стає занадто громіздким для використання. Стає складно створювати відповідний зовнішній ключ, ускладнюються процедури пошуку.
- Атрибут первинного ключа може мати занадто багато описового вмісту, щоб його можна було безпечно використовувати.

В таких випадках доцільно використовувати так званий сурогатний ключ [1]. Сурогатний ключ — це первинний ключ, створений розробником бази даних для спрощення ідентифікації екземплярів сутності.

Практичною перевагою сурогатного ключа є те, що, оскільки він не має семантичного сенсу, його значення можуть генеруватися СКБД, щоб завжди забезпечувати унікальність значень. Зазвичай визначений системою сурогатний ключ є числовим, і його значення автоматично збільшується для кожного нового рядка. Поширеним прийомом є створення сурогатного ключа з додаванням до назви слова «ID» (наприклад, EMPLOYEE_ID, DEPARTMENT_ID, CATEGORY_ID і так далі), який є лічильником, що збільшується на 1 при додаванні запису. Він цілком відповідає вимогам до ключа-кандидата.

Приклад аналізу атрибутів сутності з метою виявлення ключів-кандидатів наведено на рис. 2.15 та 2.16

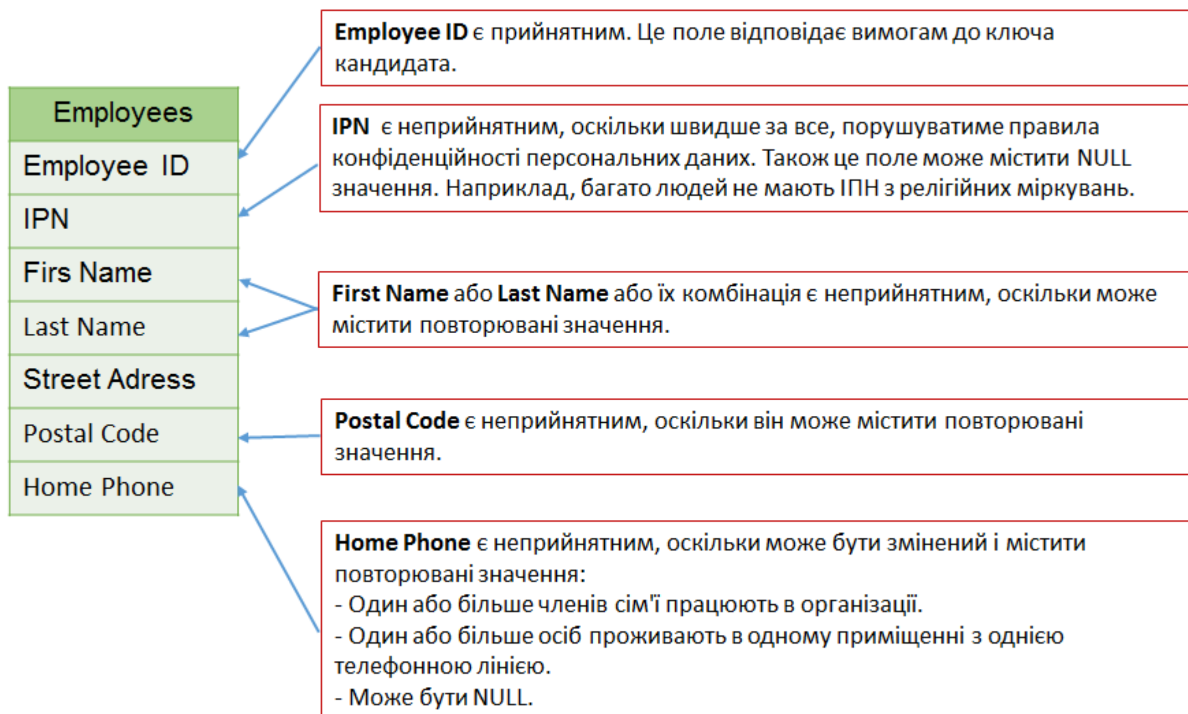


Рис. 2.15. Аналіз ключових атрибутів сутності

. Для таблиці Empoyees, виходячи з семантичних особливостей сутності, вдалося знайти атрибут, що можна використовувати як первинний ключ. Для іншої таблиці Parts, такого атрибуту, або набору атрибутів, обрати не вдається, краще створити сурогатний ключ.

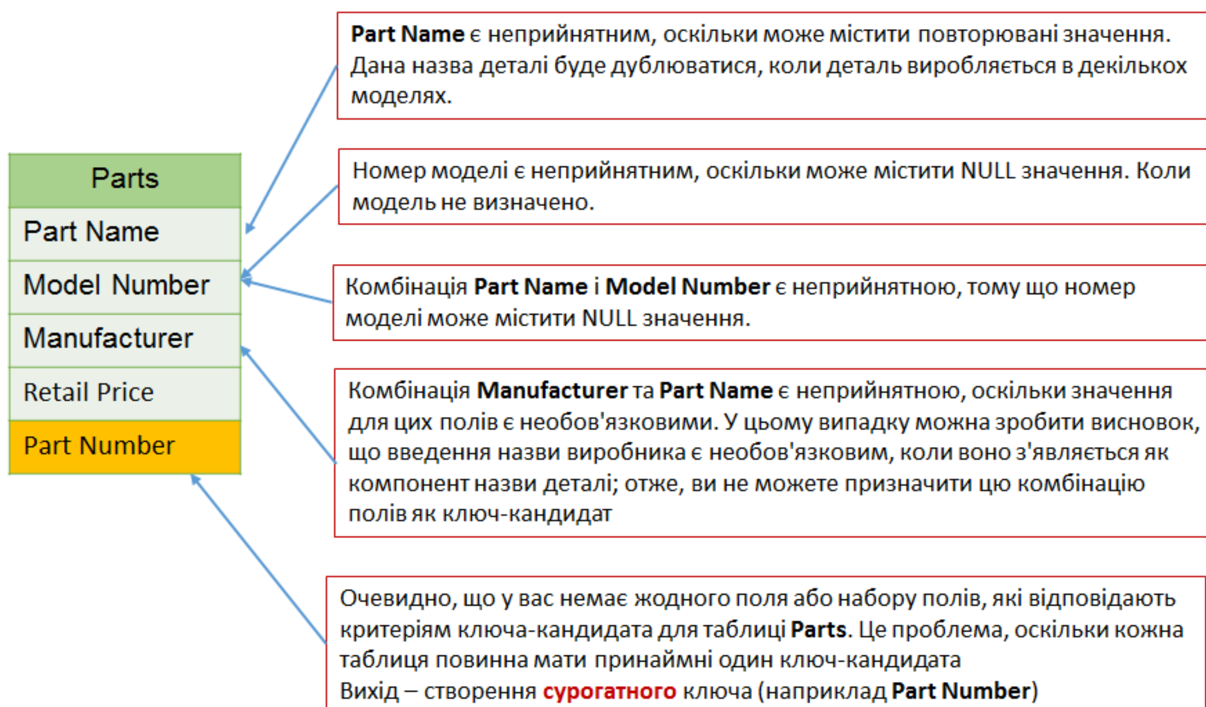


Рис. 2.16. Приклад аналізу атрибутів сутності Деталі

В випадку сурогатного ключа зазвичай створюють атрибут з постфіксом ID (наприклад, EMPLOYEE ID, VENDOR ID, DEPARTMENT ID, CATEGORY ID і так далі). Він завжди відповідає вимогам до ключа-кандидата, є чудовим первинним ключем.

Після визначення ключів-кандидатів, один з них призначаєм первинним ключем:

- Якщо є один ключ-кандидат, він стає первинним ключем
- Якщо у вас є простий ключ-кандидат і складений ключ-кандидат, виберіть простий ключ-кандидат. Завжди краще використовувати ключ-кандидат, який містить найменшу кількість полів.

Для розглянутого нами прикладу предметної області Бібліотека сутності і атрибути виглядатиме так, як зображено на рисунку 2.17.

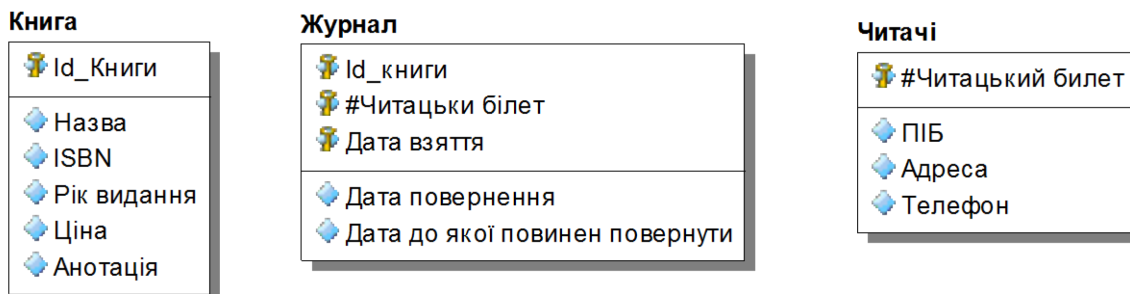


Рис. 2.17. Ключові атрибути сутностей

2.2.5. Створення зв'язку між сутностями

Якщо між двома сутностями існує функціональна залежність, на діаграмі вона відображається як зв'язок. Зв'язок – бінарна асоціації, що показують, яким чином сутності співвідносяться або взаємодіють між собою. Зв'язок може існувати між двома різними сутностями або між сутністю і нею ж самою (рекурсивний зв'язок). Якщо зв'язок встановлюється між двома сутностями, то він визначає взаємозв'язок між екземплярами однієї та другої сутності.

Зв'язку дається ім'я, яке виражається граматичної формою дієслова. Ім'я зв'язку завжди формується з точки зору батьківської сутності, так, що може бути утворено речення, якщо з'єднати ім'я сутності батька, ім'я зв'язку і ім'я сутності-нащадку (наприклад «СТУДЕНТИ - *складають* – ІСПИТИ»).

Ступінь зв'язку

Ступінь зв'язку – це кількість типів сутностей, які в ньому беруть участь (рис. 2.18) [4]. Зв'язок, що зв'язує два набори сутностей, має ступінь два і називається бінарним. Зв'язок, що зв'язує три набори сутностей, має ступінь три і називається тернарним, а зв'язок, що зв'язує більше трьох наборів сутностей, називається n-арним. Також можливий унарний зв'язок, який зв'язує набір сутностей із самим собою і має ступінь один. Найпоширенішим типом зв'язку є бінарний. Зв'язок, що зв'язує сутність із самою собою (один екземпляр сутності з іншим екземпляром тієї ж сутності) називається рекурсивним. Він має ступінь 1. В ER-моделі, що використовує нотації Crow's Foot, не підтримуються зв'язку ступеню більше 2. Це стосується і реляційної моделі.

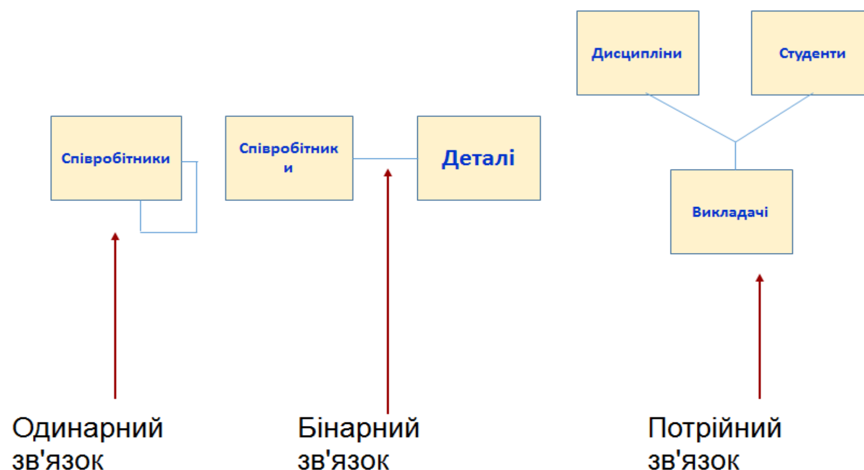


Рис. 2.18. Приклади зв'язків різного ступеня

Кардинальність зв'язку

Кардинальність бінарного зв'язку – це кількість екземплярів сутностей, на які може посилатися інший екземпляр сутності в межах цього зв'язку в будь-який момент часу [4]. Нехай X та Y – набори сутностей, а R – бінарний зв'язок набору сутностей X до набору сутностей Y. Якби на R не було обмежень кардинальності, то будь-яка кількість сутностей у X могла б бути пов'язана з будь-якою кількістю сутностей в Y. Зазвичай, однак, існують обмеження на кількість відповідних сутностей. Розрізняють три типи бінарних відношень: «один до одного», «один до багатьох», «багато до багатьох» (рис. 2.19).

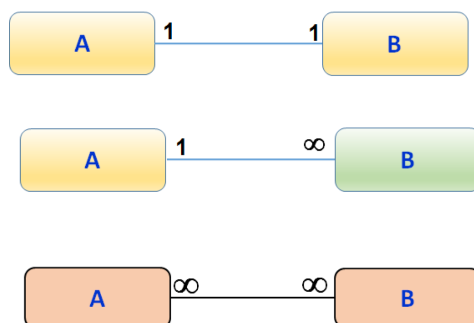


Рис. 2.19. Типи бінарних відношень

Зовнішні ключі

Атрибути, які успадковуються через зв'язок з батьківською сутністю, є зовнішніми ключами і служать для організації зв'язку між сутностями. Якщо зовнішній ключ сутності використовується як її первинний ключ або як частина складеного первинного ключа, то сутність є **залежною** від батьківської сутності, інакше– **незалежною** сутністю. Залежні сутності ще називають слабкими сутностями, а незалежні – сильними.

Якщо сутність є залежною, то зв'язок її з батьківською сутністю має назву **ідентифікуючий**, інакше – **не ідентифікуючий**. В випадку ідентифікуючого зв'язку пов'язаний атрибут батьківської сутності входить до складу первинного ключа підлеглої сутності.

З точки зору реалізації сутність визначається як залежна, якщо вона має обов'язковий зовнішній ключ, тобто атрибут зовнішнього ключа, який не може бути Null. Наприклад, якщо працівник хоче вказати одного або кількох утриманців для цілей утримання податків, виникає необхідність у сутності «УТРИМАНЦІ». У цьому випадку сутність «УТРИМАНЦІ» явно залежить від сутності «ПРАЦІВНИК», оскільки неможливо, щоб залежна сутність існувала окремо від ПРАЦІВНИКА в базі даних.

Зв'язок може бути **обов'язковим**, якщо у зв'язку з цим повинен брати участь кожен екземпляр сутності, **не обов'язковим** – якщо не кожен екземпляр сутності може брати участь у цих зв'язках. При цьому зв'язок може бути обов'язковим з одного боку і необов'язковим з іншого боку.

Позначення зв'язків

Як і раніше, використовуємо нотацію Crow's Foot для визначення типів зв'язку (рис. 2.20).

Зв'язок «один до одного» (1:1). Зв'язок R між множиною сутностей X та множиною сутностей Y має назву «один до одного», якщо кожен екземпляр сутності в X асоціюється не більше ніж з одним екземпляром сутності в Y і навпаки, кожен екземпляр сутності в Y асоціюється не більше ніж з одним екземпляром сутності в X.

Рис. 2.20. Позначення зв’язків в нотації Crow’s Foot

Прикладом такого зв’язку є відношення між сутностями СТУДЕНТ і ОБЛІКОВИЙ ЗАПИС студента в інформаційній системі вишу (рис. 2.21).

Login

Login_Name	User_Id	Pass_word
awoods	A97567	tryagain
kolesnik	k17459	smart144
goody	G23961	seeNow
Druz_k	D77345	Druz_k91
terica	T21374	Excellent!

Students

User_Name	User_Id	Class
Петренко	A97567	ФБ-11
Колесник	k17459	ФБ-12
Скворцов	G23961	ФБ-11
Друзь	D77345	ФБ-12
Шевченко	T21374	ФБ-11

Рис. 2.21. Сутності Студенти та Облікові записи

Позначення зв’язку сутностей наведено на рис. 2.22[2].

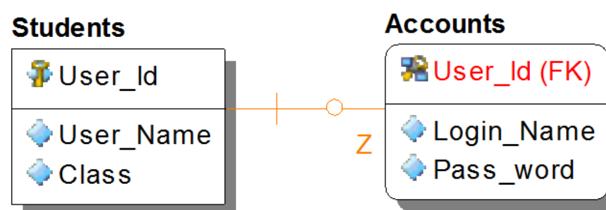


Рис. 2.22. Позначення зв’язку «один до одного»

Зв’язок ідентифікуючий, оскільки первинний ключ батьківської сутності входить до складу первинного ключа підлеглої сутності. Не для кожного студента може бути створений обліковий запис, тому зв’язок не обов’язковий з боку Студенти.

Ще один приклад – зв’язок між сутностями КАФЕДРА і ЗАВІДУВАЧ КАФЕДРИ. У будь-який момент часу кожен завідувач очолює щонайбільше одну кафедру, а кожна кафедра має щонайбільше одного завідувача.

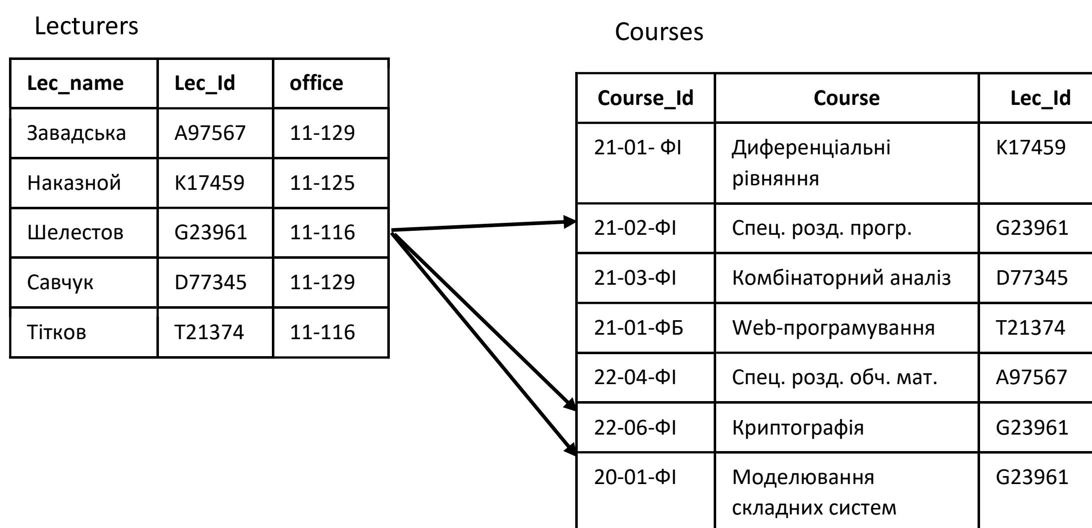
В випадку зв’язку «один до одного», необхідно перевірити, чи можна об’єднати дві сутності в одну сутність для спрощення. Якщо так, то атрибути кожної сутності слід

[2] Приклади для ілюстрації елементів ER-діаграм створені за допомогою ER/studio

об’єднати в одну. Однак іноді існують вагомі причини для збереження двох окремих таблиць, які мають зв’язок «один до одного». Наприклад, для імпорту таблиці деякі атрибути якої є конфіденційними, їх потрібно захистити спеціальним чином і не імпортувати. Загалом, додаткової складності ведення двох сутностей, кожна з яких має однаковий первинний ключ слід уникати, якщо це можливо.

Зв’язок «один до багатьох». Зв’язок між множиною сутностей X та множиною сутностей Y є зв’язком «один до багатьох», якщо кожен екземпляр сутності у X може бути пов’язаний з багатьма екземплярами сутностей у Y, але кожен екземпляр сутності у Y пов’язаний не більше ніж з одним екземпляром сутності у X.

Приклад. Сутності Викладач і Дисципліни (рис. 2.23). Викладач може читати кілька дисциплін, а може не читати жодної, тому зв’язок «один до багатьох», необов’язковий з боку викладача. Зв’язок не ідентифікуючий, первинний ключ батьківської сутності не входить до складу первинного ключа підлеглої сутності. Відображення такого зв’язку наведено на рис. 2.24.



Lecturers

Lec_name	Lec_Id	office
Завадська	A97567	11-129
Наказной	K17459	11-125
Шелестов	G23961	11-116
Савчук	D77345	11-129
Тітков	T21374	11-116

Courses

Course_Id	Course	Lec_Id
21-01- ФІ	Диференціальні рівняння	K17459
21-02-ФІ	Спец. розд. прогр.	G23961
21-03-ФІ	Комбінаторний аналіз	D77345
21-01-ФБ	Web-програмування	T21374
22-04-ФІ	Спец. розд. обч. мат.	A97567
22-06-ФІ	Криптографія	G23961
20-01-ФІ	Моделювання складних систем	G23961

Рис. 2.23. Сутності Викладачі і Дисципліни

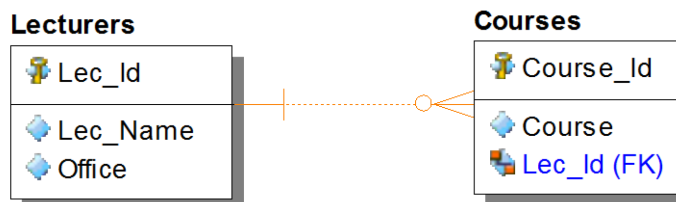


Рис. 2.24. Зв’язок між сутностями Викладачі і Дисципліни

Приклад. Сутності Покупець і Замовлення покупця (рис. 2.25). Якщо є замовлення, обов’язково є покупець, що зробив замовлення. Але покупець може і не робити замовлень. Тому зв’язок обов’язковий з одного боку і не обов’язковий з іншого. Зв’язок ідентифікуючий, первинний ключ батьківської сутності входить до складу первинного ключа підлеглої сутності. Зв’язок «один до багатьох», один покупець може зробити багато замовлень.

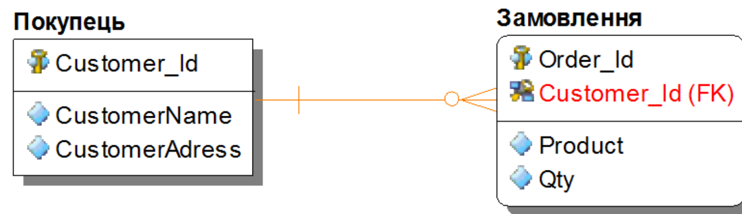


Рис. 2.25. Зв’язок між сутностями Покупець і Замовлення

Ще один приклад – зв’язок між сутностями Мати і Діти. Біологічна мати може мати багато дітей, але кожна дитина має лише одну біологічну матір.

Зв’язок «багато до багатьох». Зв’язок з множини сутностей X до множини сутностей Y є «багато до багатьох», якщо кожен екземпляр сутності в X може бути пов’язаний з багатьма екземплярами сутності в Y і кожен екземпляр сутності в Y може бути пов’язаний з багатьма екземплярами сутності в X.

Приклад: у сім’ї стосунки між дідусем і бабусею та онуками ілюструють зв’язок «багато до багатьох». Бабуся і дідусь можуть мати багато онуків, а онуки можуть мати багато бабусь і дідусів.

Зв’язки такого ступені є неідентифікуючими за своєю суттю.

Приклад. Сутності Співробітник і Проєкт (рис. 2.26). Це також зв’язок «багато до багатьох», оскільки один Співробітник може брати участь в кількох проєктах, а в кожному проєкті бере участь багато співробітників.

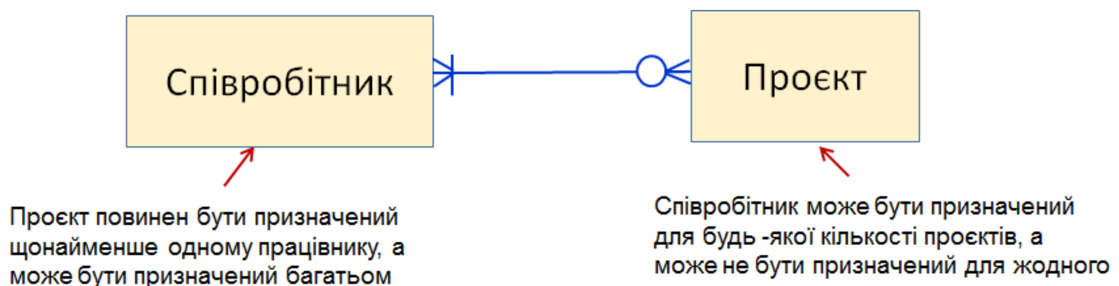


Рис. 2.26. Зв’язок між сутностями Співробітник і Проєкт

Якщо існує зв’язок «багато до багатьох», то виникають деякі проблеми – як встановити зв’язок записів з першої таблиці із записами в другій таблиці? Якщо не встановите зв’язок належним чином, а створити додаткові атрибути в однієї з таблиць:

- отримання інформації з однієї з таблиць буде складним;
- одна з таблиць буде містити велику кількість зайвих даних;
- в обох таблицях існуватимуть повторювані дані;
- вставляти, оновлювати та видаляти дані буде складно.

Зв’язок такого ступеня реалізується шляхом додавання ще однієї сутності, яка розташовується між двома початковими. Нова сутність успадковує первинні ключі обох сутностей, які стають зовнішніми ключами і разом створюють первинний ключ нової сутності.

Якщо повернутись до розглянутого прикладу предметної області Бібліотека, то можна з’ясувати, що у одного автора може бути багато книжок, а у однієї книги може бути кілька авторів. Тобто маємо типовий приклад зв’язку «багато до багатьох». На рис. 2.27 показано як можна реалізувати його в ER-діаграмі.

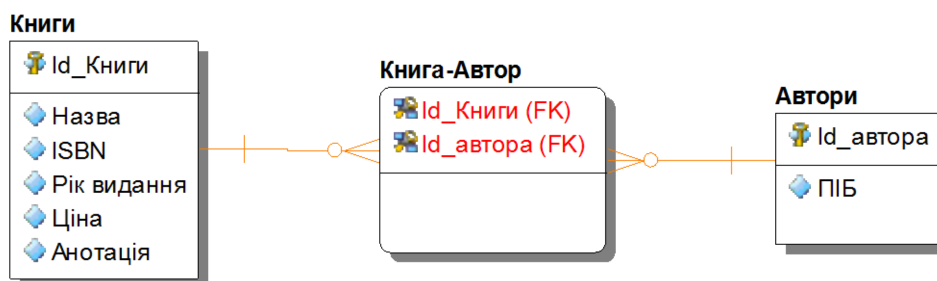


Рис. 2.27. Реалізація зв’язку між сутностями Книги та Автори

В ER-діаграмі зв’язки демонструють функціональні залежності між об’єктами предметної області. Якщо існує більш ніж одна функціональна залежність, то і в ER-діаграмі між сутностями може бути більш одного зв’язку різного змісту.

Рекурсивний зв’язок

Рекурсивний зв’язок означає, що батьківська сутність і підлегла сутність є однією і тією ж сутністю, утворюючи рекурсивний або вкладений зв’язок, а первинний ключ сутності також стає її власним зовнішнім ключем. Рекурсивний зв’язок виникає, коли сутності самі утворюють ієрархічний зв’язок. У практичних застосуваннях такі сутності рекурсивного зв’язку є дуже поширеними. Наприклад, структура організації включає в себе вищі відділи та підпорядковані їм відділи. Один відділ може мати один або кілька підлеглих відділів, найнижчий відділ не має підлеглих відділів, а вищий відділ не має батьківського відділу.

Рекурсивний зв’язок – це зв’язок ступеня 1, що показує, як пов’язані різні екземпляри однієї і тієї ж сутності між собою.

Приклад. В сутності Працівники атрибут Manager ID – це посилання на інший екземпляр сутності, в якому первинний ключ Employee Id збігається з таким Manager ID (рис. 2.28). Цей екземпляр сутності пов’язаний з керівником конкретного працівника.

EmpId	EmpName	EmpSpec	Manager Id
0234	Чуприна	директор	Null
1205	Петренко	бригадир	0234
3478	Міщук	нач. дільниці	0234
2715	Скуратов	Маляр	1205
7654	Захарченко	штукатур	1205
1409	Литвин	Слюсар	3478
6317	Друзь	водій	3478

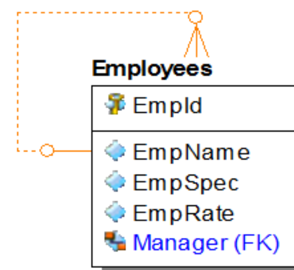


Рис. 2.28. Рекурсивний зв’язок Керівник - Підлеглий

Очевидно, що зв'язок є необов'язковим з обох сторін. В протилежному випадку, виникає нескінченна ієрархія. Кардинальність зв'язку – «один до багатьох». В одного керівника може бути багато підлеглих, у працівника тільки один керівник.

Наведемо інші приклади рекурсивного зв'язку різних ступенів.

Рекурсивний зв'язок «багато до багатьох»: Кожна деталь може складатись з інших деталей, кожна деталь може бути частиною інших (рис. 2.29).

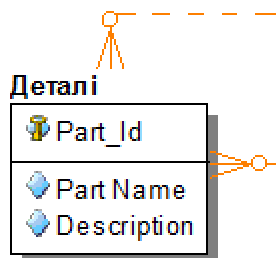


Рис. 2.29. Рекурсивний зв'язок Деталі - Компоненти

Рекурсивний зв'язок «один до одного»: Людина перебуває у шлюбі з однією іншою особою або взагалі неодружена (рис. 2.30).

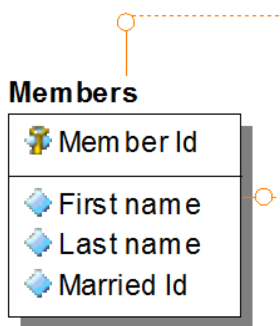


Рис. 2.30. Рекурсівний зв'язок Шлюбні стосунки

Контрольні запитання

1. Назвіть етапи проєктування бази даних.
2. Опишіть, які типи сутності можуть бути в ER-моделі.
3. Опишіть, які типи зв'язків можуть бути в ER-моделі і наведіть приклади унарних, бінарних зв'язків.
4. Який зв'язок називається рекурсивним?
5. Яка кардинальність зв'язку між двома типами сутностей «Проєкт» та «Службовець», якщо відомо, що на підприємстві існує один проєкт и 21 співробітник бере участь у цьому проєкті?
6. Які завдання вирішуються на етапі концептуального проєктування?
7. Як визначається процес на DFD діаграмі, його властивості?
8. Що таке слабка сутність, її властивості?
9. Що таке ідентифікуючий зв'язок, і як він відображається в ERD.
10. Що називається сутністю та екземпляром сутності?
11. Що називається атрибутом сутності та екземпляром атрибута?

12. Що називається зв’язком між сутностями?
13. Яка роль первинного та зовнішнього ключів у схемі бази даних?.
14. Що таке правила цілісності даних і чому вони важливі в схемі бази даних?
15. Які існують рекомендації щодо іменування зв’язку?

Тестові завдання

1. Прямокутник у DFD представляє:
   A. Процес
   B. Сховище даних
   C. Зовнішній об’єкт
   D. Вхідні дані
2. Під зовнішнім об’єктом розуміємо:
   A. Об’єкт за межами системи, що проєктується, який може контролюватися розробником
   B. Об’єкт за межами системи, поведінка якого не залежить від системи, що проєктується
   C. Сховище, зовнішнє по відношенню до системи, що проєктується
   D. Блок, який не є частиною DFD
3. Потік даних може:
   A. Тільки входити в сховище даних
   B. Тільки виходить зі сховища даних
   C. Входити або виходити зі сховища даних
   D. або Входити, або виходити зі сховища даних, але не одночасно
4. Яке з наступних тверджень є правильним щодо зв’язку «один до багатьох»?
   A. Кожен атрибут первинного ключа бере участь на стороні «один»
   B. Кожен атрибут зовнішнього ключа бере участь на стороні «один»
   C. Кожен атрибут первинного ключа бере участь на стороні «багато»
   D. Кожен атрибут зовнішнього ключа бере участь з боку «багато»
5. Що з наведеного нижче не входить до класичної ER-моделі?
   A. Сутності
   B. Зв’язки
   C. Обмеження цілісності
   D. Атрибути
6. Яке з наступних тверджень є неправильним?
   A. Кожна сутність повинна мати атрибути
   B. Кожен зв’язок повинен мати атрибути
   C. Деякий атрибут або атрибути повинні бути здатні однозначно ідентифікувати кожен екземпляр сутності
   D. Ідентифікаторів сутностей, що беруть участь у зв’язку, достатньо для однозначної ідентифікації кожного екземпляра зв’язку
7. Які з наведених нижче тверджень правильні?
   A. Зв’язки представляють реальні асоціації між однією або кількома сутностями
   B. Зв’язки описуються в термінах їхнього ступеня та кардинальності

   C. Кожна сутність, що бере участь у зв’язку, відіграє певну роль
   D. Всі вищезазначені твердження є правильними
8. Залежно від природи атрибуту, він може належати до однієї з наведених нижче пар типів. Який з наведених нижче типів не можна вважати типом атрибута?
   A. Простий або складений
   B. Однозначний або багатозначний
   C. Обов’язковий або похідний
   D. NULL або NOT NULL
   E. Всі вищезазначені твердження є правильними
9. [Множинний вибір] Зв’язок між сутностями має вигляд:
   A. A. Зв’язок «один до одного» (1:1)
   B. B. Зв’язок «один до нуля» (1:0)
   C. C. Зв’язок «один до багатьох» (1:N)
   D. D. Зв’язок «багато до багатьох» (M:N)
10. Яке з наступних речень пояснює, як сутності, атрибути та записи працюють разом?
   A. Запис містить один атрибут з багатьма сутностями
   B. Сутність може мати лише один атрибут у записі
   C. Запис містить всі пов’язані сутності та всі їхні атрибути
   D. Запис – це всі значення атрибутів для одного елемента сутності
11. Поле _____ містить значення, які однозначно ідентифікують кожен запис у таблиці, і слугує полем-зв’язком у таблиці на батьківській стороні зв’язку «один до багатьох».
   A. Поле первинного ключа
   B. Поле зовнішнього ключа
   C. Обчислюване поле
   D. Природне поле
12. Для чого потрібне поле зовнішнього ключа?
   A. Це зв’язуюче поле в таблиці на батьківській стороні зв’язку «один до багатьох»
   B. Воно встановлює порядок записів у таблиці
   C. Це зв’язуюче поле в таблиці на стороні «багато» (підлеглому) зв’язку «один до багатьох»
   D. Зв’язує дві таблиці, які не мають спільного поля
13. Яке з наступних тверджень є правильним щодо зв’язку «один до багатьох»?
   A. Кожне атрибут первинного ключа бере участь на стороні «один»
   B. Кожне атрибут зовнішнього ключа бере участь на стороні «один»
   C. Кожне атрибут первинного ключа бере участь на стороні «багато»
   D. Кожне атрибут зовнішнього ключа бере участь з боку «багато»
14. _____ існує між двома сутностями, коли кожен рядок у першій сутності може відповідати багатьом рядкам у другій сутності, і кожен рядок у другій сутності відповідає лише одному рядку у першій сутності.
   A. Запис
   B. Зв’язок «багато до багатьох»
   C. Зв’язок «один до багатьох»

D. Атрибут

15. Яке з наступних тверджень правильне?
    A. Всі відношення можуть бути перетворені у бінарні відношення
    B. Всі відношення можуть бути перетворені у відношення 1:1
    C. Всі атрибути відношення можуть бути прикріплені до однієї з сутностей-учасниць
    D. Всі зв’язки можуть бути представлені у вигляді таблиці в базі даних

3. РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ

3.1. Загальні поняття реляційного підходу до організації даних

Принципи реляційної моделі даних (РМД) були закладені в 1970 році доктором Е.Ф. Коддом, Він вперше сформулював основні поняття і представив реляційну модель для опису бази даних, яка не має недоліків ієрархічної та мережевої моделей, що мали навігаційний інтерфейс, заснований на фізичних зв'язках та обмеженнями, пов'язаних з апаратним забезпеченням. Ідея Кодда полягала в тому, що якщо ігнорувати спосіб зв'язку між даними і впорядкувати дані у прості двовимірні невпорядковані таблиці, то можна зосередитися на даних як на даних, а не на фізичній реалізації логічної моделі.

Реляційна база даних зберігає дані у відношеннях (relations), які користувач сприймає як таблиці. Кожне відношення складається з кортежів, або записів, та атрибутів, або полів. Фізичний порядок записів або полів у таблиці абсолютно неважливий, і кожен запис у таблиці ідентифікується полем, яке містить унікальне значення. Це дві характеристики реляційної бази даних, які дозволяють даним існувати незалежно від того, як вони фізично зберігаються в комп'ютері. Таким чином, користувачеві не потрібно знати фізичне місцезнаходження запису, щоб отримати його дані.

Як і інші моделі даних, реляційна модель складається з трьох частин: структурної, маніпуляційної і цілісної (рис. 3.1).

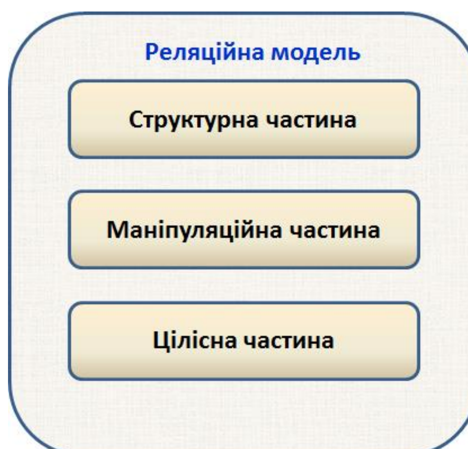


Рис. 3.1. Складові реляційної моделі

Структурна частина визначає, які базові (примітивні) типи можна використовувати, яким чином будувати складні структури даних із простіших. Маніпуляційна частина визначає які операції над даними можна використовувати в рамках моделі. Цілісна частина визначає способи опису взаємозв'язків між об'єктами даних і засоби завдання обмежень цілісності.

Розглянемо кожну з частин реляційної моделі.

3.2. Структурна частина РМД

Визначення та позначення

У математиці заведено елементи множин позначати маленькими літерами, але в теорії баз даних цієї традиції немає, тобто як самі множини, так і їхні елементи позначатимуться як великими так і маленькими літерами. Позначення, що використовуються в реляційної моделі наведені в табл. 3.1.

Таблиця 3.1. Позначення, що використовуються в реляційної моделі

Символ	*Опис*
Дужки { і}	*Перелік елементів множини,* $X = \{a, b, c, d, e\}$
$\varnothing$	*Порожня множина* $N = \varnothing$
$\in$	*Елемент входить до множини,* $a \in X$
$\notin$	*Елемент не входить до множини,* $a \notin Y$
$=$	*Множини містять однакові набори елементів,* $X = Z$
$\times$	*Добуток множин. Множина всіх можливих комбінацій значень доменів,* $A \times B$
$\cap$	*Перетин. Набір таких елементів множин, які належать обом множинам,* $X \cap Y$
$\cup$	*Об'єднання. Сукупність елементів обох множин,* $X \cup Y$
$\subseteq$	*Підмножина. Одна множина входить до складу іншої,* $K \subseteq X$
$\subset$	*Строга підмножина. Одна множина входить до складу іншої, при цьому вони не дорівнюють одна одній,* $K \subset X$
$\not\subset$	*Не підмножина . Одна множина не входить до складу іншій,* $L \not\subset X$

Щоб зрозуміти справжнє значення реляційної моделі, потрібно розглянути деякі поняття з математики [13].

Домен. У реляційній моделі даних доменами (D) називаються множини деяких значень. При цьому припускається, що всі значення, які містяться в доменах, є атомарними:

$$D = \{ d_1, d_2, .. d_n \}$$

На будь-якому домені визначено бінарне відношення рівності, для будь-яких двох значень із домену можна визначити, збігаються ці значення чи ні. Крім цього, на доменах можуть бути визначені інші властивості, операції і функції. Наприклад, на доменах, що містять числові значення, можна розглядати:

- відношення впорядкування $<$, $\geq$ $\leq$, $>$, $>$;
- арифметичні операції, наприклад $+$, $-$, $\times$, $\div$.

Прикладами доменів можуть бути множина всіх цілих чисел, множина всіх рядків, множина всіх номерів деталей і т. д. Обмеження на типи даних доменів відсутні, обов'язковий тільки тип «булево».

Повний декартів добуток. Повний декартів добуток D_1 x D_2 x ... D_n – множина усіх можливих комбінацій значень доменів [13]:

$$d_1 \times d_2 \times .. d_n = \prod_{i=1}^{n} D_i ,$$

$$\text{де } d_1 \subseteq D_1,\ d_2 \subseteq D_2,\ ...\ d_n \subseteq D_n \, .$$

Приклад:

- множина всіх студентів групи D_1 = {Іваненко, Петренко, Степанов};
- множина всіх дисциплін D_2 = (Теорія систем, Бази даних};
- множина допустимих оцінок D_3 = {3, 4, 5}

Повний декартів добуток множин D_1 x D_2 x D_3 наведено в табл. 3.2.

Таблиця 3.2. Приклад повного декартового добутку множин D_1, D_2, D_3

Прізвище	Дисципліна	Оцінка
Іваненко	Теорія систем	3
Іваненко	Теорія систем	4
Іваненко	Теорія систем	5
Петренко	Теорія систем	3
Петренко	Теорія систем	4
Петренко	Теорія систем	5
Степанов	Теорія систем	4
Степанов	теорія систем	5
Степанов	Теорія систем	3
Іваненко	Бази даних	3
Іваненко	Бази даних	5
Іваненко	Бази даних	4
Петренко	Бази даних	4
Петренко	Бази даних	3
Степанов	Бази даних	3
Петренко	Бази даних	5
Степанов	Бази даних	5
Степанов	Бази даних	4

Відношення. Відношення (R) – підмножина повного декартова добутку

$$R \subseteq \prod_{i=1}^{n} D_i \quad (n > 1).$$

Приклад відношення наведено в табл. 3.3.

Таблиця 3.3. Відношення – підмножина повного декартового добутку

Прізвище	Дисципліна	Оцінка
Іваненко	Теорія систем	4
Іваненко	Бази даних	3
Петренко	Теорія систем	5
Степанов	Теорія систем	5
Степанов	Бази даних	4

Домени, що входять в відношення не обов’язково різні. В табл. 3.4 атрибути HomeTeam і VisitingTeam належать одному домену – множини команд, що беруть участь в чемпіонаті. Атрибути HomeGoals і VisitorGoals також належать одному домену – множині цілих чисел.

Таблиця 3.4. Приклад відношення визначеного на однакових доменах

HomeTeam	VisitingTeam	HomeGoals	VisitorGoals
Динамо	Карпати	3	1
Карпати	Ворскла	2	0
Динамо	Шахтар	1	2
Шахтар	Ворскла	0	1

Атрибут. Атрибут – входження домена у відношення. Ім’я атрибута показує його роль в відношенні. Атрибут може приймати тільки значення, що належить домену.

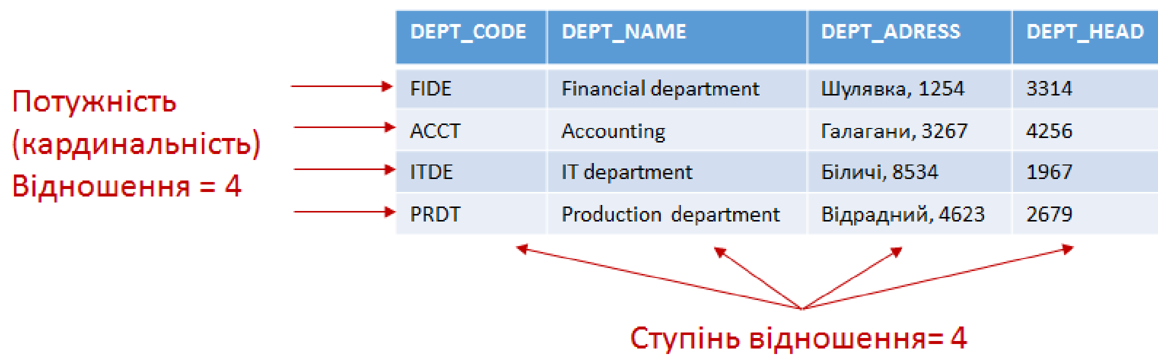
Кортеж. Кортеж t-рядок відношення. Значення атрибута A деякого кортежу позначається t [A] (або t.A) . t [A, B] – значення атрибутів A і B деякого кортежу.

Ступінь. Ступінь (ранг) відношення – кількість атрибутів відношення.

Кардинальність (потужність). Потужність відношення (кардинальне число) – кількість кортежів відношення (рис. 3.2).

Екземпляр. Екземпляр відношення – стан об’єкта що моделюється, в поточний момент часу. Це набір кортежів, фіксованих у базі даних у певний момент часу.

Порівнянні атрибути. Θ-порівнянні атрибути – атрибути що приймають значення з одного і того ж домена, де Θ – множина допустимих операцій порівняння, заданих для даного домену.



DEPT_CODE	DEPT_NAME	DEPT_ADRESS	DEPT_HEAD
FIDE	Financial department	Шулявка, 1254	3314
ACCT	Accounting	Галагани, 3267	4256
ITDE	IT department	Біличі, 8534	1967
PRDT	Production department	Відрадний, 4623	2679

Рис. 3.2. Кількісні характеристики відношення

Схема відношення. Схема відношення S_R – це перелік атрибутів відношення із зазначенням домену, до якого вони належать

$$S_R = (A_1, A_2, \dots A_n), \quad A_i \subseteq D_i$$

Наприклад, схема розглянутого відношення має вигляд

$$S_{сесія} = (\text{Студенти, Дисципліни, Оцінки})$$

Еквівалентні схеми. Схеми двох відношень еквівалентні, якщо ступені відношень збігаються і можливе таке впорядкування атрибутів, коли на однакових позиціях перебуватимуть атрибути, що належать одному домену

$$S_{R1} = (A_1, A_2, \dots A_n);$$

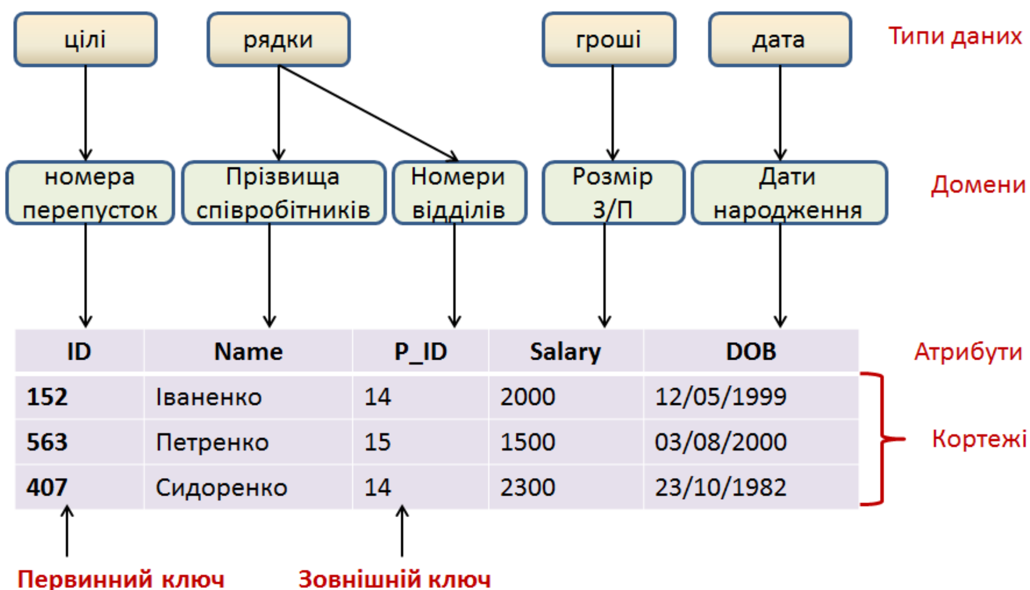
$$S_{R2} = (B_1, B_2, \dots B_m);$$

$$S_{R1} = S_{R2}, \longrightarrow \begin{cases} n = m \\ A_i, B_i \subseteq D_i. \end{cases}$$

Схема бази даних – сукупність схем відношень

$$S = \{S_{R1}, \dots, S_{Rn}\}$$

Зазвичай, коли записуємо відношення у вигляді таблиці, то перераховуємо імена атрибутів як заголовки стовпців і просто записуємо кортежі, використовуючи значення, вибрані з відповідних доменів. Таблиця – це просто відображення такого відношення (рис. 3.3).



ID	Name	P_ID	Salary	DOB
152	Іваненко	14	2000	12/05/1999
563	Петренко	15	1500	03/08/2000
407	Сидоренко	14	2300	23/10/1982

Рис. 3.3. Представлення відношення в вигляді таблиці

Властивості відношень

Більшість характеристик, визначених для реляційних таблиць, випливають з властивостей математичних відношень:

- оскільки відношення є множиною, порядок кортежів не має значення. Тому в таблиці порядок рядків не має значення;
- у декартовому добутку доменів всі комбінації різні, тому і відношення всі кортежі різні. Отже, в реляційній таблиці немає рядків, що повторюються.
- оскільки домени, на яких визначено відношення, мають атомарні значення, кожен елемент у кожному кортежі є атомарним. Аналогічно, кожна клітинка таблиці містить лише одне значення;
- у відношенні можливі значення для певного атрибуту визначаються областю, в якій цей атрибут визначений. У таблиці значення в кожному стовпчику повинні походити з одного домену;
- у математичному відношенні важливим є порядок елементів у кортежі, що визначається схемою відношення. Наприклад, впорядкована пара (1,2) значно відрізняється від впорядкованої пари (2,1). Однак у таблицях порядок стовпців не має значення. Причина в тому, що заголовки стовпців повідомляють нам, до якого атрибуту належить значення. Однак після того, як структура таблиці обрана, порядок елементів у рядках повинен відповідати порядку назв стовпців;
- обмеженість відношень. Відношення включає не всі можливі кортежі з декартова добутку доменів. Для кожного відношення є критерій, який пов'язаний з семантикою відношення, є логічним виразом і називається предикатом відношення.

3.3. Цілісна частина реляційної моделі

Важливо зберегти цілісність, тобто правильність і внутрішню узгодженість даних у базі даних, не дозволяючи користувачам вводити дані, які можуть зробити дані в БД

некоректними. Реляційна модель дозволяє визначити обмеження цілісності, які є правилами, що застосовуються до всіх екземплярів відношень БД. Коректний стан бази даних – це стан, який відповідає всім обмеженням цілісності. Частиною роботи системи управління базами даних є забезпечення дотримання обмежень цілісності – щоб гарантувати, що будь-які введені дані створюють коректний стан бази даних.

3.3.1. Обмеження домену

Схема відношення визначає домен кожного атрибута у відношенні. Це обмеження домену, які необхідно перевіряти в кожному екземплярі відношення. Крім того що значення кожного типу атрибута має бути атомарним, обмеження домену гарантує, що значення атрибута відповідає його домену. Оскільки існують обмеження на набір значень домену, так існує і набір дозволених для атрибутів значень. Вони називаються доменними обмеженнями, і їх можна застосувати, визначивши домен для атрибута під час створення таблиці.

Наприклад, якщо домен має вбудований тип даних Int, атрибут відношення може приймати тільки цілі значення, він не може приймати символьні значення або значення дати чи часу.

Іншими обмеженнями домену є:

- Обмеження NOT NULL. Якщо атрибут має значення NULL, це означає, що значення або не застосовується, або невідоме. Як тільки оголошено атрибут як NOT NULL, це означає, що ми обмежуємо атрибут у прийнятті значень, і за відсутності значень запис не буде прийнятий.
- Обмеження значення за замовчуванням (DEFAULT). За допомогою цих обмежень встановлюється значення за замовчуванням для атрибута. Якщо додається кортеж у відношення і немає значень для таких атрибутів, в кортежі атрибутам буде надано значення за замовчуванням.
- Обмеження з умовою перевірки CHECK. Це гарантує, що відношення має задовольняти заданому предикату, вказаному в умові перевірки. Якщо значення відповідають умовам, то воно буде вставлене, в іншому випадку його буде відхилене.

3.3.2. Цілісність сутностей

З тієї властивості, що у відношенні немає кортежів-дублікатів, випливає, що існує набір атрибутів, який унікальним чином ідентифікує кожен кортеж. У крайньому випадку, такий набір атрибутів може складатися з усіх атрибутів кортежу. Такий набір атрибутів називається ключем відношення. В відношенні можливо існування кількох потенційних ключів. Ключі відношень бувають атомарними і складеними. Атомарний ключ складається з єдиного атрибуту, а складений ключ – із набору атрибутів.

Формально, набір атрибутів K

$$K \subseteq \{A_1, ..., A_m\}$$

є ключем відношення

$$R(A_1, A_2, ..., A_n)$$

якщо:

- для будь-якого стану відношення R, не існує двох різних кортежів зі значенням ключа K, що збігається

$$t_1 [K] \neq t_2 [K] \Rightarrow t_1 \neq t_2;$$

- набір атрибутів ключа K є мінімально можливим. Тобто при видаленні будь-якого з атрибутів ключа втрачається властивість унікальності.

Обмеження первинного ключа, яке називається цілісність сутності, стверджує, що у відношенні жоден атрибут первинного ключа не може мати Null значення. За визначенням, первинний ключ – це мінімальний ідентифікатор, який використовується для унікальної ідентифікації кортежів. Це означає, що жодна підмножина первинного ключа не є достатньою для забезпечення унікальної ідентифікації кортежів. Якби можна було дозволити Null-значення для будь-якої частини первинного ключа, це означало б, що не всі атрибути потрібні для розрізнення кортежів, що суперечило б визначенню.

Можливі ключі відношення

В теорій реляційних баз даних існує наступна система ключів, що ідентифікують кожен кортеж відношення [6,7]:

- Суперключ – це ключ, який однозначно ідентифікує кортежі у відношенні/таблиці. Це набір одного або декількох атрибутів з відношення, який однозначно ідентифікує кортежі. Суперключ може мати додаткові атрибути, які не є обов’язковими для унікальної ідентифікації рядків.
- Ключ-кандидат (потенційний ключ) – це ключ, який однозначно ідентифікує кортежі з відношення / таблиці. Ключ-кандидат – це мінімальний суперключ без надлишкових атрибутів.
- Первинний ключ. Первинний ключ – це ключ, який однозначно ідентифікує кортеж з відношення / таблиці. Первинний ключ – це вибраний розробником бази даних ключ-кандидат. Таблиця не може мати більше одного первинного ключа.
- Альтернативний ключ. Ключ-кандидат, який не вибрано як первинний ключ, називається альтернативним ключем.

Обмеження первинного ключа встановлюється під час створення таблиці. Для ключів-кандидатів СКБД дозволяє вказати як обмеження унікальності, так і обмеження на відмінність від Null.

Вибір ключа не можна визначити тільки з аналізу структури і змісту таблиці. Наприклад, аналіз змісту таблиці (рис. 3.4) показує, що немає двох робітників, що мають однакове прізвище і ім’я. Здається, звідси ми можемо зробити висновок, що ці атрибути можна використовувати як первинний ключ відношення? Ні, аналіз предметної області доводить, що комбінація таких атрибутів не буде унікальною.

Робітники

EmpId	First Name	Last Name	salary
4001	Петренко	Костянтин	65000
4002	Антоненко	Максим	90000
4003	Черкаський	Дмитро	87000
4004	Яценко	Дарина	65000
4005	Сергєєв	Данило	80000

Рис. 3.4. Відношення Робітники

Інший приклад (рис. 3.5) демонструє складність вибору атрибутів, що будуть ідентифікувати кожен рядок відношення. А якщо такий набір атрибутів знайдено, ключ буде мати дуже велику довжину.

title	genre	year	director	minutes	budget	gross
The Company Men	drama	2010	John Wells	104	15,000,000	4,439,063
Lincoln	biography	2012	Steven Spielberg	150	65,000,000	181,408,467
War Horse	drama	2011	Steven Spielberg	146	66,000,000	79,883,359
Argo	drama	2012	Ben Affleck	120	44,500,000	135,178,251
Fire Sale	comedy	1977	Alan Arkin	88	1,500,000	NULL
Lincoln	biography	1992	Peter W. Kunhardt	240	NULL	NULL
Life	comedy	1999	Ted Demme	108	75,000,000	63,844,974
Life	drama	1999	Eun-Ryung Cho	19	NULL	NULL

Рис. 3.5. Приклад відношення зі складеним ключем

Ще один приклад – наявність кількох ключів-кандидатів (рис. 3.6). В наведеному прикладі це може бути як номер держ. реєстрації так і VIN-код автомобіля.

№ авто	VIN	Марка	Модель	Рік
AA 4570 КИ	A7364926	Ford	Mustang	2014
AA 1243 КХ	W9237461	Opel	Corsa	2021
PE 6543 AE	C8436105	Toyota	Camry	1998
AP 1476 KE	Y4562846	Volkswagen	Passat	2020
HB 6490 PE	U1230987	Opel	Astra	2018
AA 6457 KE	X0483726	Toyota	RAV4	2024

Рис. 3.6. Наявність кількох потенційних ключів відношення

Звідси можна зробити висновок – поняття ключа є семантичним, потенційні ключі визначається семантикою предметної області, а не існуючим набором даних.

В випадках, коли первинний ключ має дуже велику довжину (це основна причина) або ключ важко визначити (як на рис. 3.5), використовують «сурогатний» ключ, наприклад унікальний номер запису. Але при цьому необхідно зберегти атрибути, що складають природний первинний ключ і самостійно контролювати цілісність сутностей.

3.3.3. Цілісність посилань

Посилальна цілісність – це набір обмежень і правил, які забезпечують узгодженість і логічну цілісність даних у всіх пов'язаних таблицях, гарантуючи, що дані, які зберігаються в цих таблицях, є дійсними, узгодженими і відповідають встановленій схемі. Підтримка посилальної цілісності необхідна для збереження точності та надійності даних у базі даних, запобігання неузгодженості даних, що можуть виникнути через неправильне управління відносинами між сутностями в моделі реляційної бази даних.

Посилальна цілісність контролюється використанням обмежень первинного ключа і зовнішнього ключа. Атрибут відношення, що посилається на первинний ключ іншого або того ж самого відношення, стає зовнішнім ключем.

Приклад. Відношення (рис. 3.7) моделює постачання деталей на підприємство.

номер постачальника	**Найменування постачальника**	*Номер деталі*	**Найменування деталі**	**Обладнання, кількість**
1	Іваненко	*1*	Болт	100
1	Іваненко	*2*	гайка	200
1	Іваненко	*3*	гвинт	300
2	Петренко	*1*	Болт	150
2	Петренко	*2*	гайка	250
3	Сидоренко	*3*	гвинт	1000

Рис. 3.7. Відношення Постачання деталей

Зовнішні ключі

Основним недоліком відношення (рис. 3.7) є те, що в одному відношенні зібрана інформація стосовно різних сутностей – Постачальники, Деталі і, власне, факт постачання. Щоб усунути ці недоліки, зробимо декомпозицію відношення на 3 відношення (рис. 3.8).

номер постачальника	**Найменування постачальника**
1	Іваненко
2	Петренко
3	Сидоренко

Номер деталі	**Найменування деталі**
1	Болт
2	гайка
3	гвинт

номер постачальника	*Номер деталі*	**Обладнання кількість**
1	*1*	100
1	*2*	200
1	*3*	300
2	*1*	150
2	*2*	250
3	*3*	1000

Рис. 3.8. Зв'язки відношень після декомпозиції

Сутності Постачальники і Деталі зараз моделюються окремими відношеннями. Третє відношення моделює постачання деталей, але постачальники і деталі представлені посиланнями на первинні ключі в відповідних відношеннях. Такі посилання мають назву зовнішні ключі. Значення зовнішнього ключа кортежу повинно представляти існуючий кортеж у зв'язаному (батьківському) відношенні.

Формально підмножина атрибутів відношення R називається зовнішнім ключем (foreign key) F_K, якщо підтримується цілісність посилань:

- існує відношення S з первинним ключем P_K (батьківське відношення);
- зовнішній ключ F_K визначений на тих же доменах, що і первинний ключ P_K;
- кожне значення F_K у відношенні R завжди збігається зі значенням P_K для деякого кортежу з S, або є Null-значення.

Зовнішній ключ не повинен обов'язково бути компонентом первинного ключа батьківського відношення. У цьому випадку зовнішній ключ може прийматися значення Null. Зовнішній ключ та посилальний первинний ключ повинні бути визначені на одному домені. Однак імена атрибутів можуть бути різними.

Якщо відношення R_n, R_{n-1}, R_{n-2} .... R_1 такі, що

$R_n \rightarrow R_{n-1} \rightarrow R_{n-2} \rightarrow ....R_2 \rightarrow R_1$, ($\rightarrow$ - зв'язок за зовнішнім ключем)

то ланцюжок від R_n до R_1 утворює посилальний шлях (referential path).

У посилальному шляху відношення може бути як батьківським, так і підлеглим. Наприклад R_2 в посилальному шляху $R_3 \rightarrow R_2 \rightarrow R_1$.

Якщо зовнішній ключ посилається на первинний ключ того ж самого відношення, такий зовнішній ключ називається рекурсивним зовнішнім ключем.

Приклад. Відношення «Співробітник», де кожен кортеж працівника також містить посилання на керівника працівника:

схема відношення:

`Employee {Emp, Emp_FName, Emp_LName, Emp_Dept, . . ., Emp_MgrEmp}`

первинний ключ: `PK[Emp]`

зовнішній ключ: `FK [Emp_MgrEmp]` посилання на `Employee.Emp.`

Цілісність посилань контролює СКБД. Якщо використовуються операції, що можуть порушити цілісність посилань (оновлення або видалення кортежу в батьківському відношенні.), СКБД приймає рішення щодо обробки таких операцій використовуючи такі політики підтримки цілісності посилань:

а) Загальні:

- RESTRICT (обмежити). Операцію не буде дозволено;
- CASCADE (каскадувати). Будуть оновлені або видалені всі кортежі в пов'язаних відношеннях, на які посилається первинний ключ, що оновлюється.

б) Додаткові:

- SET NULL (встановити в Null). Всі значення зовнішніх ключів встановлюються в Null;
- SET DEFAULT (встановити за умовчанням). В батьківській таблиці створюється спеціальний запис і зовнішні ключі посилаються на нього.

3.3.4. Null-значення і тризначна логіка

Null – спеціальне позначення відсутності значення атрибуту. На практиці часто доводиться ставити обмеження на присутність Null-значення в кортежі.

Незважаючи на те, що використання Null-значень було предметом теоретичних суперечок, зараз всі СКБД підтримують використання Null-значень. Наприклад, без Null неможлива операція зовнішнього з'єднання. Але використовуючи Null треба враховувати деякі правила. Такі правила мають назву тризначна логіка.

Приклад. Як знайти студентів, старших за 18 років, що мешкають в гуртожитку № 6 (рис. 3.9). В нашому випадку вік одного зі студентів невідомий, а іншому гуртожиток не потрібний (N/A – не застосовується)

Студент	Дата народження	№ гуртожитку
Іваненко	12.09.2002	5
Петренко	(відсутня)	6
Сидоренко	23.12.2001	N/A

Рис. 3.9. Відношення з Null-значеннями

Для пошуку відповіді необхідно використовувати операції, одним з операндів яких є Null. Ось такі правила треба враховувати.

Для будь-якої арифметичної операції (+, –, *, /,%) , коли хоча б один операнд Null, результат буде Null:

$$X + \text{Null} \rightarrow \text{Null};\ X/X \rightarrow \text{Null};\ X * \text{Null} \rightarrow \text{Null}$$

Для операцій порівняння (=,>,<,<>, >=, <=), коли хоча б один операнд Null, результат буде Unknown (або NULL в деяких системах) :

$$X > \text{Null} \rightarrow \text{Unknown};\ \ X = \text{Null} \rightarrow \text{Unknown}$$

Результат логічних операцій, коли хоча б один операнд Null, наведено на рис. 3.10.

AND	F	T	Null
F	F	F	F
T	F	T	Unknown
Null	F	Unknown	Unknown

OR	F	T	Null
F	F	T	Unknown
T	T	T	T
Null	Unknown	T	Unknown

	NOT
F	T
T	F
Null	Unknown

Рис. 3.10. Логічні операції тризначної логіки

3.3.5. Основні правила перетворення моделі «сутність – зв'язок» в реляційну

Створена на етапі концептуального моделювання ER-модель використовується для створення реляційної моделі. Існують правила перетворення ER-моделі в об'єкти реляційної моделі [3,4,8]:

1. Сутність $\Rightarrow$ Відношення.

Кожній сутності ER-моделі ставиться у відповідність відношення реляційної моделі даних. При цьому необхідно розуміти, що імена сутностей і відношень можуть бути різні. В концептуальної моделі на імена сутностей не накладаються ніякі обмеження, за винятком їх унікальності в рамках моделі. В реляційної моделі імена відношень можуть бути обмежені вимогами конкретної СКБД. Найчастіше імена відношень обмежені по довжині, алфавітом що використовується і не можуть містити пробілів та інших спеціальних символів.

Наприклад, сутність може називатися «Студенти університету», а відповідне відношення – Students.

2. Атрибут сутності $\Rightarrow$ Атрибут відношення.

Кожному атрибуту сутності ставиться у відповідність атрибут відношення. Перейменування атрибутів проводиться відповідно до тих же правил, що і перейменування сутностей. Потім кожному атрибуту відношення задається конкретний (допустимий в використовуваної СКБД) тип даних і обов'язковість або необов'язковість даного атрибута (необов'язковість атрибута означає, що він може прийняти спеціальне значення Null).

В ER-моделі атрибут сутності може бути складеним. В реляційній моделі це не припустимо, тому для кожної складової такого атрибуту створюється окремий атрибут відношення. Крім того, в ER-моделі атрибут сутності може бути багатозначним. В реляційної моделі для багатозначних атрибутів створіть окрему таблицю з двома стовпчиками, перший стовпчик буде мати значення первинного ключа екземпляру сутності, а другий стовпчик – значення багатозначного атрибуту. Первинним ключем таблиці багатозначних атрибутів завжди є складений ключ (тобто комбінація обох стовпців).

3. Первинний ключ сутності $\Rightarrow$ PRIMARY KEY відношення.

Первинний ключ сутності стає первинним ключем (PRIMARY KEY) відповідного відношення. Всі атрибути, які входять в первинний ключ сутності, автоматично отримують властивість NOT NULL.

4. Зовнішній ключ сутності $\Rightarrow$ FOREING KEY відношення

У кожне відношення, що відповідає підпорядкованої сутності, додається набір атрибутів, що є первинним ключем батьківського відношення. У підпорядкованому відношенні цей набір атрибутів стає зовнішнім ключем (FOREIGN KEY), за допомогою якого і реалізуються зв'язок між відношеннями.

5. Обов'язковий зв'язок $\Rightarrow$ NOT NULL.

Для моделювання необов'язковою зв'язку у атрибутів, відповідних зовнішньому ключу, встановлюється можливість приймати невизначені значення (Null). Для моделювання обов'язкової зв'язку атрибутам зовнішнього ключа встановлюється неприпустимість невизначених значень (NOT NULL).

6. Реалізація зв'язку «багато до багатьох».

У реляційної моделі підтримується тільки один тип зв'язку з множинністю – зв'язок «один до багатьох». Разом з тим, ER-модель підтримує і зв'язку «багато до багатьох» (випадок «один до одного» не розглядається, тому що він є окремим випадком зв'язку «один до багатьох»). Тому потрібен спеціальний механізм перетворення зв'язку «багато до багатьох» при переході до реляційної моделі. У цьому випадку вводиться додаткове відношення, атрибутами якого є первинні ключі двох пов'язаних відношень. Дані ключі є в новому відношенні зовнішніми (FOREIGN KEY), утворюючи разом вже первинний ключ нового відношення (PRIMARY KEY). Саме нове відношення пов'язано з кожним з вихідних відношень зв'язком «один до багатьох».

7. Перетворення пов'язаних сутностей.

- Якщо зв'язок «один до одного» обов'язковий для обох сутностей, то потрібно тільки одне відношення. Первинним ключем цього відношення може бути ключ будь-якої з двох сутностей (рис. 3.11).

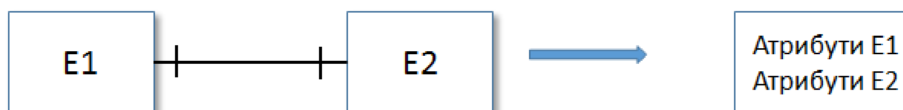


Рис. 3.11. Перетворення обов'язкового зв'язку «один до одного»

- Якщо зв'язок «один до одного» для обох сутностей необов'язковий (рис. 3.12) і ми впевнені, що це різні сутності, можливі кілька варіантів перетворення в реляційні відношення.

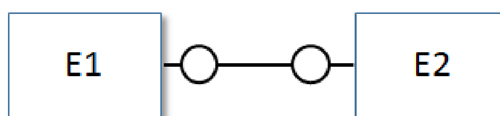


Рис. 3.12. Необов'язковий зв'язок «один до одного»

Перший варіант – помістити ключ будь-якого з цих відношень в іншу таблицю, щоб показати зв'язок (тобто або помістити ключ E1 в таблицю E2, або навпаки, але не обидва, як на рис. 3.13).

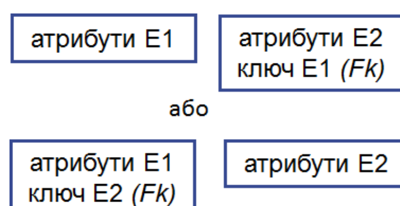


Рис. 3.13. Перший варіант перетворення

Другий варіант – створити додаткове відношення, що містить ключі обох відношень (рис. 3.14).

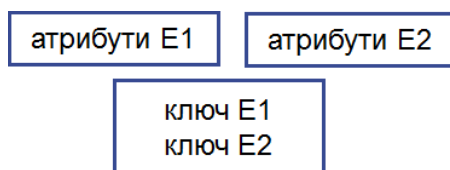


Рис. 3.14. Другий варіант перетворення

- Якщо зв'язок для однієї сутності є обов'язковим, а інший –необов'язковим, то необхідно побудова двох відношень (рис. 3.15). Під кожну сутність виділяється відношення, при цьому ключ сутності повинен служити первинним ключем для відповідного відношення. Крім того, ключ сутності, з необов'язковою приналежністю додається як атрибут в відношення з обов'язковою приналежністю.

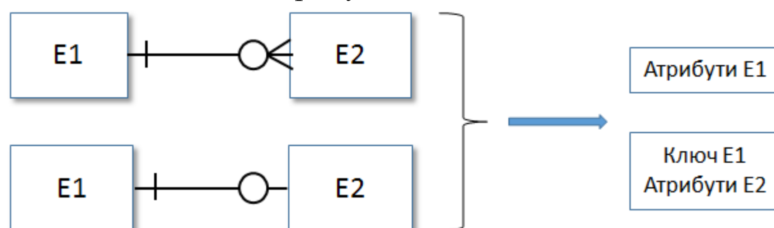


Рис. 3.15. Перетворення зв'язку не обов'язкового з одного боку

- Якщо зв'язок для обох сутностей необов'язковий, то необхідно використовувати три відношення : по одному для кожної сутності і одне відношення для зв'язку (рис. 3.16). Відношення зв'язку повинно мати в числі своїх атрибутів ключі кожної сутності.

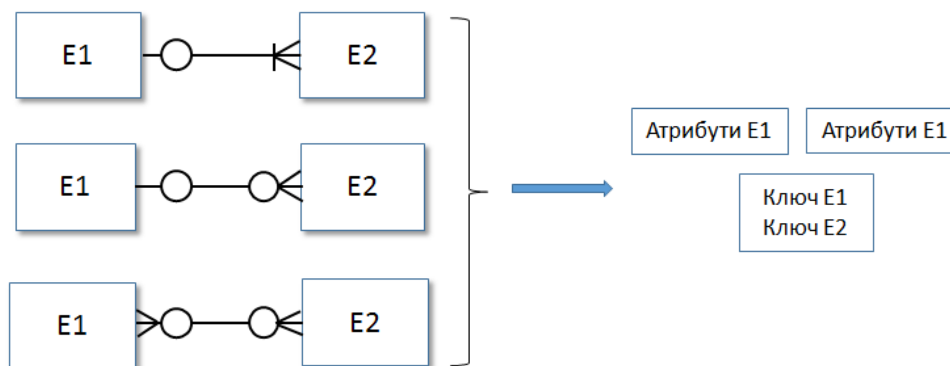


Рис. 3.16. Перетворення зв'язку не обов'язкового з обох сторін

3.4. Маніпуляційна частина реляційної моделі

Реляційна модель має наступні дві формальні мови:

- Реляційна алгебра,
- Реляційне числення.

Оскільки відношення є множинами, до них застосовуються всі звичайні операції над множинами. Кодд запропонував реляційну алгебру як основу маніпулювання вмістом відношення / таблиці. Алгебра – це множина об'єктів із заданою на ньому сукупністю

операцій, замкнутих відносно цієї множини, званого основною множиною. Реляційна алгебра – множина відношень із замкнутим на ньому множиною операцій над відношеннями. Отже, реляційна алгебра є базовим набором операцій, що використовуються в реляційній моделі. Реляційні операції мають одну важливу властивість: вони замкнені відносно поняття відношення. Це означає, що вирази реляційної алгебри визначаються над відношеннями реляційних БД і результатом обчислення також являються відношення. Оскільки результатом будь-якої реляційної операції є деяке відношення, можна створювати реляційні вирази, в яких замість відношення – операнду деякої реляційної операції знаходиться вкладене реляційне вираження.

Реляційна алгебра складається з двох груп операцій (рис. 3.17):

- теоретико-множинні операції;
- спеціальні операції.

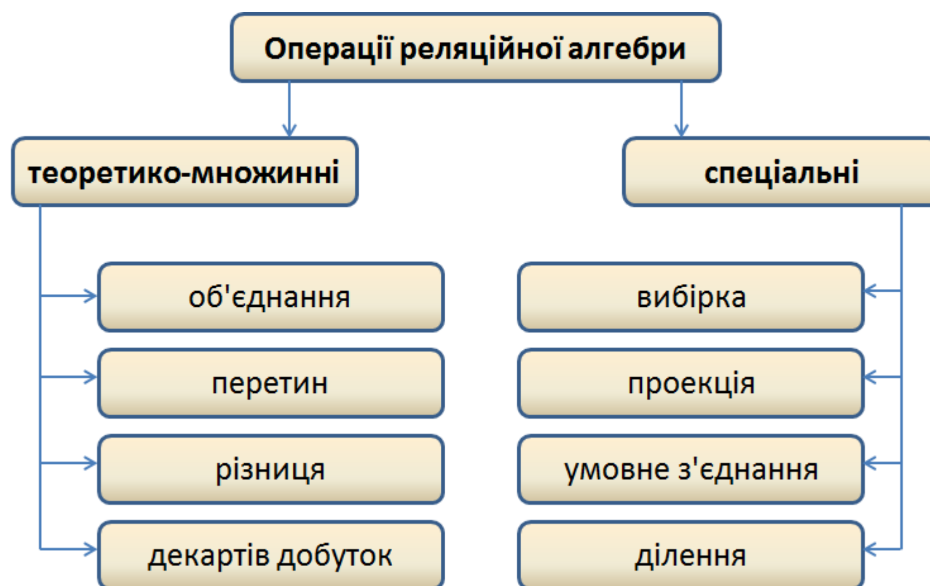


Рис. 3.17. Операції реляційної алгебри

Всі операції мають умовне позначення (рис. 3.18). В таблиці додано операцію перейменування. Хоча вона не відноситься безпосередню до реляційної алгебри, вона використовуються в складних вираз для запобігання дублювання імен атрибутів.

Позначення	Назва операції
∪	Об'єднання відношень
∩	Перетин відношень
-	Різниця відношень
×	Розширений декартів добуток відношень
σ	Вибірка з відношення
π	Проекція відношення
⋈	З'єднання відношень
÷	Ділення двох відношень
ρ	Перейменування атрибутів відношень

Рис. 3.18. Позначення операцій реляційної алгебри

3.4.1. Об'єднання відношень

Операція об'єднання є бінарною операцією і позначається символом «∪». Результатом операції об'єднання є відношення з такою ж схемою, як і у вхідних відношень.

Для застосування операції об'єднання до відношень R_1 і R_2 обидва відношення повинні бути сумісними за типом (або сумісними за об'єднанням). Сумісність за типом означає, що обидва відношення R_1 і R_2 повинні мати однакову кількість атрибутів, а також кожна відповідна пара атрибутів повинна належати до одного домену, інакше кажучи відношення повинні мати еквівалентні схеми:

$$R_1 = \{R_n\},\ R_2 = \{R_m\}$$

$$S_{R1} = S_{R2} \ \rightarrow \ n = m \ \text{ і } \ r_{1i}, r_{2i} \subseteq D_i$$

де r_1 і r_2 – відповідно кортежі відношень R_1 і R_2,

Результатом $R_1 \cup R_2$ буде відношення, яке включає всі кортежі, що є в R_1 або R_2, або в обох. Дублікати кортежів усуваються.

Приклад. Об'єднання відношень $R_3 = R_1 U\ R_2$ (рис. 3.19)

R1 - Деталі	
Шифр деталі	Назва деталі
11073	Гайка М1
11075	Гайка М2
11076	Гайка М3
11003	Болт М1
11006	Болт М3
13063	Шайба М1
13066	Шайба М3

R2 - Деталі	
Шифр деталі	Назва деталі
11073	Гайка М1
11076	Гайка М3
11077	Гайка М4
11004	Болт М2
11006	Болт М3

$R_3 = R_1 U\ R_2$

R3 - Деталі	
Шифр деталі	Назва деталі
11073	Гайка М1
11075	Гайка М2
11076	Гайка М3
11003	Болт М1
11006	Болт М3
13063	Шайба М1
13066	Шайба М3
11077	Гайка М4
11004	Болт М2

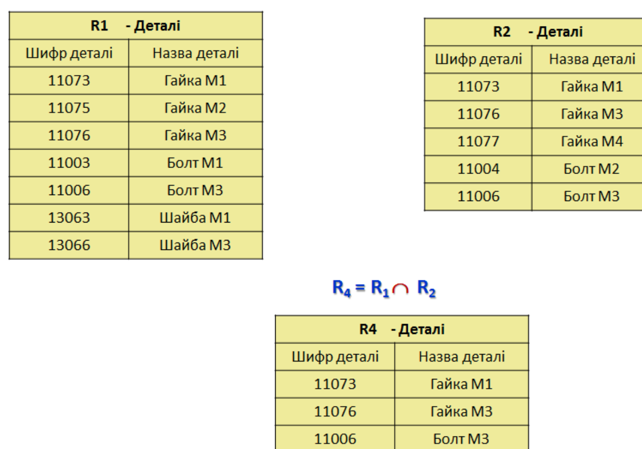
Рис. 3.19. Приклад об'єднання відношень

3.4.2. Перетин відношень

Операція перетину є бінарною операцією і позначається символом «∩». Для застосування операції перетину до відношень R_1та R_2 вони повинні бути сумісними за об'єднанням.

Результатом операції $R_1 \cap R_2$ буде відношення, яке включає всі кортежі, що з'являються в обох відношеннях R_1 і R_2. Дублікати кортежів усуваються.

Приклад. Перетин відношень $R_3 = R_1 U\ R_2$ (рис. 3.20).



R1 - Деталі	
Шифр деталі	Назва деталі
11073	Гайка М1
11075	Гайка М2
11076	Гайка М3
11003	Болт М1
11006	Болт М3
13063	Шайба М1
13066	Шайба М3

R2 - Деталі	
Шифр деталі	Назва деталі
11073	Гайка М1
11076	Гайка М3
11077	Гайка М4
11004	Болт М2
11006	Болт М3

$R_4 = R_1 \cap R_2$

R4 - Деталі	
Шифр деталі	Назва деталі
11073	Гайка М1
11076	Гайка М3
11006	Болт М3

Рис. 3.20. Приклад перетину відношень

3.4.3. Різниця відношень

Операція ризниця є бінарною операцією і позначається символом «–».

Для того, щоб застосувати цю операцію до відношень R_1і R_2, вони повинні бути сумісними по об’єднанню.

Результатом цієї операції, що позначається $R_1 - R_2$, є відношення, яке включає всі кортежі, що є у відношенні R_1, але не є у відношенні R_2.

Операція не є комутативною операцією, тобто $R_1 - R_2 \neq R_2 - R_1$.

Зауважте, що перетин можна виразити через об’єднання та різницю множин, як показано нижче:

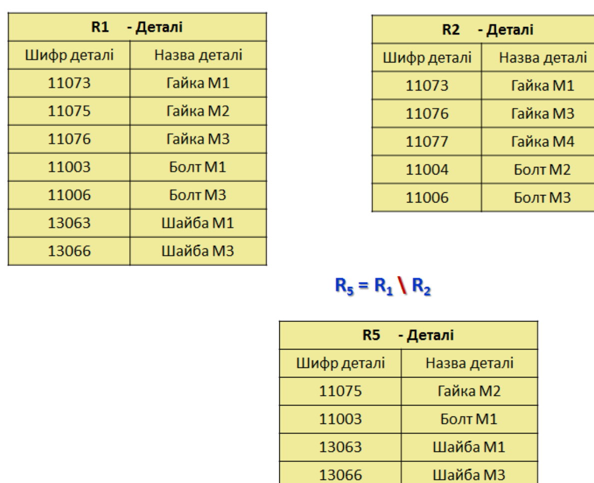
$$R_1 \cap R_2 = ((R_1 \cup R_2) - (R_1 - R_2)) - (R_2 - R_1)$$

$$R_1 \cap R_2 = R_1 - (R_1 - R_2)$$

$$R_1 \cap R_2 = R_2 - (R_2 - R_1)$$

Операція перетину є надлишковою, але історично зберігається в списку операцій реляційної алгебри.

Приклад. Різниця відношень $R_5 = R_1 - R_2$ (рис. 3.21).



R1 - Деталі	
Шифр деталі	Назва деталі
11073	Гайка М1
11075	Гайка М2
11076	Гайка М3
11003	Болт М1
11006	Болт М3
13063	Шайба М1
13066	Шайба М3

R2 - Деталі	
Шифр деталі	Назва деталі
11073	Гайка М1
11076	Гайка М3
11077	Гайка М4
11004	Болт М2
11006	Болт М3

$R_5 = R_1 \setminus R_2$

R5 - Деталі	
Шифр деталі	Назва деталі
11075	Гайка М2
11003	Болт М1
13063	Шайба М1
13066	Шайба М3

Рис. 3.21. Приклад різниці відношень

Розглянуті бінарні операції мають такі властивості (OP – операція):

- Асоціативність
  (A OP B) OP C = A OP (B OP C)
- Комутативність (крім операції різниці)
  A OP B = B OP A.

3.4.4. Розширений декартів добуток

Зчепленням, або конкатенацією, кортежів

$$a = <a_1, a_2, ..., a_n> \text{ і } b = <b_1, b_2, ..., b_m>$$

називається кортеж, отриманий додаванням значень другого в кінець першого. Зчеплення кортежів a і b позначається як (a, b).

$$(a, b) = <a_1\ a_2, ..., a_n, b_1, b_2, ... b_m>$$

Розширеним декартовим добутком відношення R_1 ступеня n зі схемою $S_{R1}= (A_1, A_2..., A_n)$ і відношення R_2 ступеня m зі схемою $S_{R2}= (B_1, B_2, ..., B_m)$ називається відношення R_3 ступеня n + m зі схемою

$$S_{R3} = (A_1, A_2, ..., A_n, B_1, B_2, ..., B_m),$$

що містить кортежі, отримані зчепленням кожного кортежу відношення R_1 з кожним кортежем відношення R_2. Декартів добуток змінює ступінь результуючого відношення.

Декартів добуток є бінарною операцією, тобто вимагає наявності двох відношень і не вимагає сумісності відношень з об’єднанням. Для виконання добутку кожен рядок першого відношення з’єднується з кожним рядком другого відношення. Це може призвести до конфлікту, якщо ім’я атрибута збігається в обох відношеннях. Щоб уникнути конфлікту необхідно перейменувати атрибут, щоб уникнути конфлікту.

Декартів добуток змінює ступінь результуючого відношення.

Приклад. Декартів добуток відношень $R_3 = R_1 \times R_2$ (рис. 3.22).

R1 - Деталі

Шифр деталі	Назва деталі
11073	Гайка M1
11076	Гайка M3
11006	Болт M3

R2 - Працівники

Таб_Номер	Прізвище
4713	Петренко
2214	Ноєвий
0077	Сидоренко

$R_3 = R_1 \times R_2$

Шифр деталі	Назва деталі	Таб_Номер	Прізвище
11073	Гайка M1	4713	Петренко
11073	Гайка M1	2214	Ноєвий
11073	Гайка M1	0077	Сидоренко
11076	Гайка M3	4713	Петренко
11076	Гайка M3	2214	Ноєвий
11076	Гайка M3	0077	Сидоренко
11006	Болт M3	4713	Петренко
11006	Болт M3	2214	Ноєвий
11006	Болт M3	0077	Сидоренко

Рис. 3.22. Приклад декартового добутку відношень

Декартові добутки можуть давати досить великі результати залежно від кардинальності відношень, що беруть участь у них. Загалом, результат має кардинальність **n** ∗ **m**, де **n** і **m** кардинальність початкових відношень. Операція зазвичай використовується не як самостійна, а як проміжна операція перед спеціальними операціями реляційної алгебри.

3.4.5. Вибірка

Операція вибірки позначається символом «**σ**» (сигма). Це унарний оператор, тобто він приймає в якості операнду тільки одне відношення. Синтаксис операції вибірки наступний:

σ <умова вибору> (R)

Тут символ **σ** (сигма) позначає оператор SELECT, R – відношення бази даних, а умова відбору – булевий вираз, заданий на атрибутах відношення R. Булевий вираз, заданий в <умові відбору>, може мати декілька речень наступного вигляду

<ім’я атрибуту><операція порівняння><константне значення

або

<ім’я атрибуту><оператор порівняння><ім’я атрибуту>

Тут <ім’я атрибута> – ім’я атрибута R, <оператор порівняння> може бути одним з операторів {=, <, <=, >, >= ≠}, а <константне значення> – статичне значення, що зберігається в атрибуті. Речення можуть бути з’єднані стандартними булевими операторами {AND, OR і NOT} для комбінування умов відбору.

Вибірка повертає підмножину кортежів з відношення, які задовольняють умові. Це можна уявити як горизонтальне розбиття відношення на дві множини кортежів – одну, що задовольняє умову і відбирається в результуюче відношення, і іншу, що не задовольняє умову і не входить до результуючого відношення.

Приклад. Вибірка за умовою $R_2 = \boldsymbol{\sigma}_{\text{Зарплата} <= 2000} (R_1)$ (рис. 3.23).

R_1

Табельний номер	Прізвище	Зарплата
1	Іваненко	1000
2	Петренко	2000
3	Сидоренко	3000

R_2

Табельний номер	Прізвище	Зарплата
1	Іваненко	1000
2	Петренко	2000

Рис. 3.23. Приклад вибірки з відношення

3.4.6. Проекція

Операція проекції позначається символом «**π**» (pi). Проекція є унарним оператором, тобто приймає в якості операнду лише одне відношення.

Операція проекції також діє як фільтр і вибирає певні стовпці з відношення, яке перераховане в списку атрибутів запиту. Її можна уявити як вертикальне розбиття

відношення на два набори стовпців – один, який потрібен у результуючому відношенні, і інший, який не потрібен і не є частиною результуючого відношення.

Синтаксис операції проекції наступний:

π <список атрибутів> (R)

Тут символ **π** (pi) позначає операцію проекції, а <список атрибутів> – бажаний набір атрибутів з атрибутів вхідного відношення (R). Список атрибутів визначає схему вихідного відношення.

Оскільки схема відношення змінюється, можливо поява кортежів-дублікатів. Їх потрібно видалити.

Приклад. Проекція відношення (рис. 3.24).

$R_3 = \pi_{\text{Місто постачальника}} (R_1)$

R1

номер постачальника	Найменування постачальника	місто постачальника
1	Іваненко	Київ
2	Петренко	Одеса
3	Сидоренко	Одеса
4	Сидоренко	Житомир

R₃

місто постачальника
Київ
Одеса
Житомир

Рис. 3.24. Приклад проекції відношення

3.4.7. Умовне з’єднання

З’єднання є операцією реляційної алгебри і похідною декартового добутку і позначається символом «⋈».

Операцію з’єднання можна розглядати як операцію декартового добутку, за якою слідує операція вибірки. Ця операція дуже корисна для будь-якої реляційної бази даних, коли потрібно обробити зв’язки між відношеннями, тобто вона використовується для з’єднання двох відношень, як декартов добуток, але при цьому видаляє дублікати атрибутів і спрощує операції відбору та проекції. Іншими словами, вона з’єднує різні відношення, використовуючи стовпці з порівнянною інформацією.

Загальна форма операції з’єднання над двома відношеннями $A(a_1, a_2, ..., a_n)$ і $B(b_1,b_2,..., b_m)$ така:

A<умова об’єднання> B

Результатом об’єднання є відношення C з n + m атрибутами $C(a_1, a_2, ..., a_n, b_1, b_2, ... , b_m)$ у цьому порядку. Відношення C містить всі комбінації кортежів з A та B, якщо вони задовольняють умові з’єднання.

Оскільки комбінація декартова добутку і вибірки у запитах є поширеним явищем, було введено спеціальний оператор з’єднання JOIN. Як і в операції декартова добутку за необхідністю виконується перейменування атрибутів.

Типи з’єднань

Операція з’єднання має кілька варіацій використання, які поділяються на дві категорії:

1. Внутрішні з’єднання. Внутрішнє з’єднання включає тільки ті кортежі, які задовольняють умову збігу, тоді як решта кортежів виключаються. Нижче наведено різні типи внутрішніх з’єднань:
- природне з’єднання;
- еквіз’єднання;
- тета-з’єднання.
2. Зовнішні з’єднання. Зовнішнє з’єднання корисне в тих випадках, коли ми хочемо включити інформацію з одного або обох відношень, навіть якщо вони не задовольняють умові з’єднання. Якщо кортежі збігаються, то з’єднуються кортежі з обох відношень, а якщо не збігаються, то все одно з’єднуються з підстановкою Null-значень для значень, що не збігаються. Зовнішні з’єднання бувають трьох типів:
- ліве зовнішнє з’єднання,
- праве зовнішнє з’єднання,
- повне зовнішнє з’єднання.

Природне з’єднання

Природне з’єднання – це бінарна операція. При природному з’єднанні нам не потрібно явно вказувати оператор порівняння в умові. Воно об’єднує два різних відношення на основі спільного стовпця. Найважливіше, що спільні атрибути повинні мати однакове ім’я та домен в обох відношеннях.

Природне з’єднання діє на ті спільні атрибути, де значення атрибутів в обох відношеннях збігаються і поміщаються в результуюче відношення.

Для відношень R_1 і R_2 зі схемами

$$R_1(A_1, A_2, \dots .A_n, X_1, X_2, .. X_m)$$

$$R_2(B_1, B_2, \dots B_r, X_1, X_2, .. X_m)$$

$X_1, X_2, .. X_m$ – атрибути що збігаються, результатом природнього з’єднання

$$R_3 = R_1 \bowtie R_2 \text{ або } R_3 = R_1 \text{ JOIN } R_2$$

є відношення зі схемою

$$S_{R3} = (A_1, A_2, \dots .A_n, X_1, X_2, .. X_m, B_1, B_2, \dots B_r)$$

що містить кортежі R_1 і R_2 де значення атрибутів в обох відношеннях збігаються.

Приклад. Природне з’єднання відношень R_1 і R_2 (рис. 3.25).

R_1

Батько	Дитина
Петро	Наталя
Михайло	Олександр
Володимир	Андрій

R_2

Мати	Дитина
Галина	Олександр
Тетяна	Наталя
Інга	Андрій

$R_3 = R_1$ JOIN R_2

Батько	Дитина	Мати
Петро	Наталя	Тетяна
Михайло	Олександр	Галина
Володимир	Андрій	Інга

Рис. 3.25. Приклад природнього з'єднання

Еквi-з'єднання

Еквi-з'єднання – це спеціальний тип природного з'єднання двох відношень R і S за вказаним атрибутом. Атрибути повинні належати до одного домену. Воно виконує лише умови рівності між парою атрибутів. Результуюче відношення містить ті кортежі, у яких значення атрибутів будуть рівними і в схемі вихідного відношення цей атрибут з'явиться тільки один раз. Назви атрибутів в умові з'єднання не обов'язково однакові.

Приклад. Еквi-з'єднання відношень R_1 і R_2 за умовою рівності прізвища працівника і проекції за атрибутами Прізвище і Кількості днів що відпрацьовано (рис. 3.26).

Прізвище працівника і кількість днів що відпрацьовано

R1 - Працівники

Таб_Номер	Прізвище	Ставка
4713	Петренко	100
2214	Ноєвий	150
0077	Сидоренко	120
1674	Пащенко	120

R2 – Табель робочого часу

Таб_Номер	Кількість днів
4713	25
2214	19
0077	24

$R_3 = R_1 \bowtie_{(R1.Таб_Номер\ =\ R2.Таб_Номер)} R_2$

Таб_Номер	Прізвище	Ставка	Таб_Номер1	Кількість днів
4713	Петренко	100	4713	25
2214	Ноєвий	150	2214	19
0077	Сидоренко	120	0077	24

$R_4 = \pi_{Прізвище,\ Кількість\ днів}(R_3)$

Прізвище	Кількість днів
Петренко	25
Ноєвий	19
Сидоренко	24

Рис. 3.26. Приклад еквi-з'єднання відношень

Тета-з'єднання

Найбільш загальною формою внутрішнього з'єднання є тета-з'єднання. Тета-з'єднання позначається як «$\bowtie\theta$», де Θ – формула, що визначає предикат з'єднання. Це загальне з'єднання, яке часто використовується, оскільки воно дозволяє вказати умову при виконанні операції з'єднання. Тета-з'єднання створює кортежі з обох операндів відношення, які задовольняють умову з'єднання. Але умова з'єднання відрізняється від умови рівності і включає в себе інші оператори порівняння, такі як { > , >= , <, <=, ≠ }.

Предикат з’єднання задається подібно до предиката вибору, за винятком того, що атрибути залучаються з обох відношень, що беруть участь у з’єднанні.

Наприклад, тета-з’єднання можна задати як $R.A \bowtie_{\theta} S.B$, де A і B – атрибути відношень R і S відповідно. З’єднання двох відношень еквівалентне виконанню вибірки, використовуючи предикат з’єднання як формулу вибору над декартовим добутком двох відношень. Таким чином, маємо наступну еквівалентність:

$$R \bowtie_{\theta} S = \sigma_{\Theta} (R \times S)$$

Слід зазначити, що якщо **Θ** включає атрибути двох відношень, які є спільними для них обох, необхідно переконатися, що ці атрибути не з’являються двічі в результаті і виконати перейменування за необхідністю.

Зовнішнє з’єднання

При внутрішньому з’єднанні відбувається втрата інформації, не можна відновити вихідні відношення. В зовнішнім з’єднанні двох відношень в обов’язковому порядку входять всі рядки або одного, або обох відношень.

Ліве зовнішнє з’єднання (Left Outer Join ($R ⟕ S$)

$$R_3 = R_1 ⟕ R_2$$

1. В результат включається еквi-з’єднання лівого і правого відношень.
2. потім в результат додаються ті кортежі лівого відношення, які не ввійшли в з’єднання на кроці 1. Для таких записів поля, відповідні правому відношенню, заповнюються значеннями Null.

Приклад. Для відношень R_1 і R_2 створити ліве зовнішнє з’єднання $R_3 = R_1 ⟕ R_2$ (рис. 3.27).

R_1

Студент	Місто
Петренко	Львів
Сидоренко	Одеса

R_2

Місто	Відстань
Одеса	500
Рівне	300

R_3

Студент	Місто	Відстань
Сидоренко	Одеса	500
Петренко	Львів	Null

Рис. 3.27. Ліве зовнішє з’єднання відношень R_1 і R_2

Праве зовнішнє з’єднання (Right Outer Join ($R ⟖ S$))

$$R_3 = R_1 ⟖ R_2$$

1. В результат включається еквi-з’єднання лівого і правого відношень.
2. Потім в результат додаються ті кортежі правого відношення, які не ввійшли в з’єднання на кроці 1. Для таких записів поля, відповідні лівому відношенню, заповнюються значеннями Null.

Приклад. Для відношень R_1 і R_2 створити праве зовнішнє з’єднання $R_3 = R_1 ⟖ R_2$ (рис. 3.28).

R_1

Студент	Місто
Петренко	Львів
Сидоренко	Одеса

R_2

Місто	Відстань
Одеса	500
Рівне	300

R_3

Студент	Місто	Відстань
Сидоренко	Одеса	500
Null	Рівне	300

Рис. 3.28. Праве зовнішнє з’єднання відношень R_1 і R_2

Повне зовнішнє з’єднання (Full Outer Join (R ⟗ S))

$$R_3 = R_1 ⟗ R_2$$

Усі кортежі з обох відношень, що беруть участь, включаються до результуючого відношення. Якщо для обох відношень немає кортежів, що збігаються, їхні відповідні атрибути, що не збігаються, заповнюються значеннями Null.

Приклад. Для відношень R_1 і R_2 створити повне зовнішнє з’єднання $R_3 = R_1 ⟗ R_2$ (рис. 3.29).

R_1

Студент	Місто
Петренко	Львів
Сидоренко	Одеса

R_2

Місто	Відстань
Одеса	500
Рівне	300

R_3

Студент	Місто	Відстань
Сидоренко	Одеса	500
Петренко	Львів	Null
Null	Рівне	300

Рис. 3.29. Повне зовнішнє з’єднання відношень R_1 і R_2

3.4.8. Ділення відношень

Остання операція реляційної алгебри – ділення, позначається як «÷». Якщо є два відношення зі схемами:

$R_1(A_1, A_2, \dots .A_n, X_1, X_2, .. X_m)$ – ділене;

$R_2(X_1, X_2, \dots X_m)$ – дільник;

причому, атрибути дільника $X_1, X_2, \dots X_m$ мають бути в схемі діленого, результатом ділення відношення R_1 на R_2

$$R_3 = R_1 \div R_2$$

буде відношення зі схемою

$$S_{R3} = (A_1, A_2, \dots .A_n) ,$$

що містить кортежи, декартов добуток яких з дільником цілком присутній в діленому.

Приклад. Які постачальники постачають ВСІ деталі

R_1=(PNUM, DNUM) де PNUM – номер постачальника, DNUM – номер деталі. Відношення містить інформацію, які деталі хто постачає.

R_2=(DNUM) де DNUM – номер деталі. Відношення містить перелік деталей.

Результатом ділення $R_3 = R_1 \div R_2$ буде відношення зі схемою R_3=(PNUM), що містить постачальників, які постачають всі деталі (рис. 3.30). Це характерна особливість операції ділення, вона дозволяє знайти відповідь на питання, в якому є речення «всі».

R_1

Номер постачальника PNUM	Номер деталі DNUM
1	1
1	2
1	3
2	1
2	2
3	3

R_2

Номер деталі DNUM
1
2
3

R_3

Номер постачальника PNUM
1

Рис. 3.30. Приклад ділення відношення R_1 на R_2

3.4.9. Перетворення операцій реляційної алгебри

Комбінування операцій

Вирази РА будуються з використанням операторів і дужок. Порядок виконання операцій задається пріоритетом операцій (рис. 3.31). В випадку рівного пріоритету операції виконуються з ліва на право. Порядок виконання може змінюватися за допомогою дужок.

операція	пріоритет
перейменування	4
вибірка	3
проекція	3
декартів добуток	2
з'єднання	2
перетин	2
ділення	2
об'єднання	1
різниця	1

Рис. 3.31. Пріоритет операцій реляційної алгебри

Тотожні перетворення

Як було сказано раніше, набір операцій реляційної алгебри має надлишковість. Ось приклади заміни операцій іншими.

$$A - (A - B) = A \cap B$$

$$B - (B - A) = A \cap B$$

$$(A \cap B) - (A - B) = A \cap B$$

$$(A \cap B) \cap (B - (A - B)) = A \cap B$$

$$(A \cap B) \cup (B - A) = B$$

Висновок: вирішити завдання можна по-різному, важливо знайти оптимальне рішення.

Приклад. Відношення R_1 і R_2, що містять по 1000 кортежів, причому тільки 10 кортежів в кожному відношенні задовольняють умові **Θ**. Відношення не містять однакових кортежів. Треба знайти об'єднання відношень що задовольняє умові Θ.

1. Якщо виконати операції в такої послідовності $R_3 = \sigma_{(\Theta)} (R_1 \cup R_2)$ то після виконання об'єднання вийде 2000 кортежів, а після вибірки залишиться 20 записів.
2. Якщо ж виконати операції в такої послідовності $\sigma_{(\Theta)} (R_1) \cup \sigma_{(\Theta)} (R_2)$, то після вибірки залишиться по 10 записів з кожного відношення, об'єднання яких дасть 20 необхідних кортежів.

Оскільки об'єднання виконується шляхом сортування даних (для видалення однакових кортежів) і проміжний результат треба зберігати, то виграш і за обсягом пам'яті, і за часом очевидний: набагато швидше впорядкувати 20 кортежів, а не 2000.

Існують правила, які дозволяють виконувати еквівалентні перетворення виразів реляційної алгебри. Ось деякі з них:

1. Закон комутативності для декартових добутків :

$$R_1 \times R_2 = R_2 \times R_1$$

2. Закон комутативності для з'єднань (Θ – умова з'єднання):

$$(R_1 \bowtie_{(\Theta)} R_2) = (R_2 \bowtie_{(\Theta)} R_1)$$

3. Закон асоціативності для декартових добутків:

$$(R_1 \times R_2) \times R_3 = R_1 \times (R_2 \times R_3)$$

4. Закон асоціативності для з'єднань (F1, F2 – умови з'єднання):

$$(R1 \bowtie (F1)\ R2) \bowtie (F2)\ R3 = R1 \bowtie (F1)(R2 \bowtie (F2)\ R3)$$

5. Перестановка селекції з об'єднанням:

$$\sigma_{(F)}(R_1 \cup R_2) = \sigma_{(F)}(R_1) \cup \sigma_{(F)}(R_2)$$

Контрольні запитання

1. З яких структурних елементів складається відношення?
2. Що називають записом і полем в структурі реляційної БД?
3. Чи впливає поряд рядків у реляційній таблиці на зміст даних?
4. Що таке ключ? Поясніть різні типи ключів у реляційних моделях даних
5. Назвіть унарні операції реляційної алгебри.
6. Коли можливо виконати операцію об'єднання відношень?
7. Яка максимальна кількість відношень може брати участь в операції різниці?
8. Що таке Null в реляційній базі даних? Навіщо він потрібен?

9. Поясніть сенс фрази – «Дані, що зберігаються в таблицях, повинні бути атомарні».
10. Що ви розумієте під обмеженням домену?
11. Які переваги застосування обмежень цілісності в процесі проєктування та реалізації бази даних (замість того, щоб робити це в процесі проєктування застосунків)?
12. За яких умов зовнішній ключ не повинен бути Null?
13. Поясніть умови, за яких первинним ключем відношення слід обрати сурогатний ключ.
14. Перерахуйте три компоненти реляційної моделі даних.
15. Детально пояснити типи обмежень цілісності

Тестові завдання

1. Яке з наступних тверджень не є правильним для поля первинного ключа таблиці?
   A. Воно не може містити Null
   B. Воно повинно містити унікальні дані для кожного запису
   C. Має містити числові дані
   D. Може використовуватися у зв’язку з іншою таблицею
2. В яких випадках не можна забезпечити посилальну цілісність для зв’язку «один до багатьох»?
   A. Таблиця на стороні «один» не має відповідного запису (записів) у таблиці на стороні «багато»
   B. Таблиця на стороні «багато» не має поля первинного ключа
   C. Таблиця на стороні «багато» має значення у полі зовнішнього ключа, якому не відповідає значення у полі первинного ключа таблиці на стороні «один»
   D. Таблиця на стороні «багато» має багато значень у полі зовнішнього ключа з одним відповідним значенням у полі первинного ключа таблиці на стороні «один»
3. Що з наведеного нижче не є обмеженням для поля первинного ключа таблиці?
   A. Не може бути Null значень
   B. Усі значення повинні бути унікальними
   C. Всі значення повинні мати відповідне значення в полі зовнішнього ключа таблиці, в якій вони мають зв’язок «один до багатьох»
   D. Усі значення повинні містити дані в межах типу даних, призначеного для цього поля
4. Яка з наведених нижче змін в базі даних, швидше за все, потребуватиме найбільшої переробки існуючих програм та запитів?
   A. Додавання нового поля в таблицю
   B. Створення нового індексу
   C. Зміна зв’язків між таблицями
   D. Додавання нового подання (view)
5. Яке з наступних тверджень є правильним? При використанні реляційної бази даних, що складається з декількох таблиць, користувачеві потрібно знати лише_______
   A. Кількість рядків у кожному відношенні

B. Кількість стовпців у кожному відношенні
C. Шляхи доступу до даних у кожному відношенні
D. Назви таблиць та назви відповідних стовпців

6. Поняття домену є фундаментальним для реляційної моделі даних. Яке з наступних тверджень щодо доменів є неправильним?
   A. Концептуально, домени відіграють роль, подібну до типів даних у мовах програмування
   B. Домени визначаються розробником, в той час як типи даних вбудовані в СКБД
   C. Домен – це набір скалярних або атомарних значень одного типу
   D. У кожному відношенні, визначеному на кількох доменах, може бути не більше одного атрибута
7. Що найкраще описує Null значення?
   A. Рядок нульової довжини, «»
   B. Пробіл або символ табуляції
   C. Відсутнє значення
   D. Рядок «Null»
8. Яке з наступних тверджень не є правильним?
   A. Операція Вибірка вибирає один або декілька рядків відношення
   B. Проекція вибирає один або кілька стовпців відношення
   C. З'єднання склеює кожен рядок одного відношення з усіма рядками іншого
   D. Різниця дає всі рядки першого відношення, яких немає в другому
9. Яка з наведених нижче операцій не завжди дає той самий список атрибутів, що й операнди?
   A. Проекція
   B. Вибірка
   C. Об'єднання
   D. Різниця
10. Який з операторів не вимагає, щоб два операнди були сумісними за об'єднанням (тобто мали однакову кількість і типи атрибутів)?
    A. Перетин
    B. Різниця
    C. Об'єднання
    D. Ділення
11. Які дві з наступних властивостей повинні належати первинному ключу відношення?
    A. Він унікально ідентифікує кожен рядок
    B. Це атрибут або набір атрибутів, на основі яких будується індекс
    C. Це атрибут або набір атрибутів, для яких не допускаються відсутні значення
    D. Власна підмножина не може бути ключем
12. Яке з наступних тверджень правильне?
    A. Декартів добуток двох відношень дорівнює їх об'єднанню
    B. З'єднанням двох відношень є вибірка з їх декартова добутку
    C. З'єднання завжди є еквіз'єднанням або природним з'єднанням

D. Ступінь еквi-з'єднання двох відношень дорівнює ступені природного з'єднання цих відношень

13. Які два з наступних операторів вимагають виключення дублікатів?

A. Проекція

B. Перетин

C. Різниця

D. Об'єднання

E. З'єднання

14. Яка з наведених нижче дій завжди призводить до отримання принаймні стільки ж рядків, скільки у відношенні R? Припустимо, що жодне з цих двох відношень не пусте.

A. $R \bowtie S$

B. $R \cap S$

C. $R \cup S$

D. $R - S$

E. $R \div S$

15. Які з наступних тверджень правильні?

A. $R \bowtie S = S \bowtie R$

B. $R - S = S - R$

C. $R \div S = S \div R$

D. $S \cup R = (S - R) \cup (S \cup R) \cup (R - S)$

4. ЛОГІЧНЕ ПРОЄКТУВАННЯ БАЗ ДАНИХ

4.1. Засоби логічного проєктування реляційної бази даних

Відповідно до конкретних вимог сервісу, проєктування бази даних повинно чітко визначити структуру бази даних, що відповідає вимогам, скільки сутностей потрібно створити, з яких атрибутів ці сутності складаються і який зв’язок між сутностями. Саме такі питання необхідно вирішити на етапі логічного проєктування реляційної бази даних. Коректної назвемо схему БД, в якій відсутні небажані залежності між атрибутами відношенні. Процес розробки коректної схеми реляційної БД називається логічним проєктуванням БД.

Реляційна модель базується на строгій математичній теорії, тому проєктування реляційної моделі на основі теорії нормалізації реляційної бази даних дозволяє побудувати оптимальну реляційну модель. На етапі проєктування логічної моделі бази даних процес правильного розміщення атрибутів у сутності називається нормалізацією.

У 1971–1972 роках Е.Ф. Кодд запропонував концепцію нормальних форм (normal form, скорочено NF) від 1NF до 3NF, в якій повністю обговорювалися питання нормалізації моделі. Пізніше інші дослідники поглибили і запропонували більш високі стандарти нормальних форм, але для реляційних баз даних достатньо досягти 3NF в практичних додатках.

Теорія нормалізації надає логічну основу для аналізу схем відношень і набір нормальних форм як ознак глибини нормалізації та інструментів процесу нормалізації. З практичної точки зору, нормалізація являє собою «прочісування» схеми бази даних з метою виявлення та усунення аномалій операцій з даними і надмірності. Схеми відношень, що мають виявлені недоліки, будуть переоблятися (як правило – декомпозуватися) у нові схеми, позбавлені цих недоліків.

Реляційна модель даних, розроблена відповідно до теорії нормалізації, має наступні наслідки:

- уникнення генерації надлишкових даних, економія обсягу використовуваної пам’яті;
- зменшення ризику неузгодженості даних через зберігання в різних місцях відомостей про одні й те ж об’єкті;
- хороша масштабованість отриманої моделі,
- гнучкість в налаштуванні для відображення змін в предметній області.

4.2. Надмірність даних і аномалії поновлення

Під час проєктування бази даних необхідно перевіряти, чи не вийшла така ситуація, за якої в одне відношення потрапляє інформація про кілька незалежних сутностей реального світу.

Аномалії операцій з даними – виникнення побічних ефектів операцій з даними, що стало результатом порушення вимоги адекватності бази даних предметної області.

Традиційно виділяють такі види аномалій операцій з даними [3,7]:

- аномалії включення;
- аномалії видалення;
- аномалії модифікації.

Продемонструємо кожну з аномалій на прикладі, для якого використовуємо наведене на рис. 4.1. відношення СТУДЕНТИ:

- **аномалії включення.** Неможливість додання записів, викликана відсутністю інших даних. Виражається в необхідності використовувати вигадані значення або Null-значення для частини атрибутів. Немає можливості створити нову групу, поки немає інформації про студентів, оскільки номер залікової книжки є первинним ключем.
- **аномалії видалення.** Ненавмисна втрата даних, викликана видаленням інших даних. Виражається у втраті не призначеної для видалення інформації, остання копія якої зберігалася в атрибутах запису, що видаляється. При видаленні запису №3 пропадає інформація про групу ФІ-82.
- **аномалії модифікаці**ї. Суперечливість даних, викликана їх надмірністю та часткової модифікацією. Виражається в необхідності з метою збереження цілісності бази даних оновлювати значення деякого атрибута в декількох записах, крім того, з яким зараз виконується операція оновлення. При заміні куратора в групі ФІ-81 необхідно змінити усі кортежі з ФІ-81.

№ заліковки	ПІБ	Група	Староста	Куратор
ФІ2101	Степаненко А.А	ФІ-81	Петренко М.А	Фокін Н.Е
ФІ2102	Петровський А.Б	ФІ-81	Петренко М.А	Фокін Н.Е
ФІ2201	Федченко М.Д	ФІ-82	Панченко А.С	Шевченко В.В
ФІ2103	Свиридов А.Д	ФІ-81	Петренко М.А	Фокін Н.Е

Рис. 4.1. Відношення СТУДЕНТИ

Очевидно, необхідно вносити певні зміни до схеми даних, і саме цей процес називається нормалізація. В нашому прикладі допоможе декомпозиція відношення Студенти на два нових, Студенти і Групи, як показано на рис. 4.2.

студенти

№ заліковки	ПІБ	Група
ФІ2101	Степаненко А.А	ФІ-81
ФІ2102	Петровський А.Б	ФІ-81
ФІ2201	Федченко М.Д	ФІ-82
ФІ2103	Свиридов А.Д	ФІ-81

Групи

Група	староста	куратор
ФІ-81	Петренко М.А	Фокін Н.Е
ФІ-82	Панченко А.С	Шевченко В.В

Рис. 4.2. Декомпозиція відношення Студенти

4.3. Функціональні залежності

4.3.1. Визначення функціональної залежності

Основою аналізу коректності схеми бази даних є функціональні залежності між атрибутами відношень.

Функціональна залежність (ФЗ) – це формальна назва основної концепції того, як атрибути залежать від інших атрибутів у відношенні або пов'язані з ними. Ідея ФЗ полягає у визначенні одного поля як якоря, з якого завжди можна знайти єдине значення для іншого поля. Фактично, таку ж ідею реалізує первинний ключ. Тому, основна задача ФЗ полягає в тому, щоб знайти первинні ключі таким чином, щоб всі дані в записі залежали тільки від первинного ключа.

Функціональна залежність

Нехай **R** – відношення зі схемою **R(X, Y, ...)**, . **X, Y** – непорожні множини атрибутів $\mathbf{X, Y} \subseteq \mathbf{R}$.

Відношення **R** задовольняє функціональній залежності $\mathbf{X} \rightarrow \mathbf{Y}$, якщо для будь-яких двох кортежів $\mathbf{t_1}$ і $\mathbf{t_2}$ в **R** за умови, що

$$\mathbf{t_1.X = t_2.X}$$

випливає, що

$$\mathbf{t_1.Y = t_2.Y}$$

Тобто, в будь-який момент часу кожному значенню **X** відповідає не більше одного значення атрибута **Y**.

Функціональна залежність позначається стрілкою $\mathbf{X} \rightarrow \mathbf{Y}$, де **X** – детермінант залежності, **Y** – залежна частина.

Приклад: № заліковки → ПІБ студента.

Взаємно-незалежні атрибути – атрибути, що не залежать функціонально один від іншого.

Якщо для визначення іншого атрибута в сутності необхідно більше одного атрибута, то такий детермінант називається **складеним** детермінантом.

Обмеження первинного ключа є окремим випадком ФЗ. Зауважте, однак, що визначення ФЗ не вимагає, щоб множина X була мінімальною; додаткова умова мінімальності повинна виконуватися для того, щоб X був ключем.

Приклад. Розглянемо визначення ФЗ для схеми відношення Графік польотів:

ГРАФІК_ПОЛЬОТІВ (Пілот, Рейс, Дата вильоту, Час відльоту)

Пілот	Рейс	Дата вильоту	Час відльоту
Іваненко	100	8.07	10:20
Іваненко	102	9.07	13:30
Ісаєв	90	7.07	6:00
Ісаєв	100	11.07	10:20
Ісаєв	103	10.07	19:30
Петренко	100	12.07	10:20
Петренко	102	11.07	13:30
Фролов	90	8.07	6:00
Фролов	90	12.07	6:00
Фролов	104	14.07	13:30

Рис. 4.3. Розклад польотів

Після аналізу предметної області, можна визначити такі ФЗ:

Рейс $\rightarrow$ Час відльоту; (кожному рейсу відповідає певний час вильоту);

(Пілот, Дата вильоту, Час відльоту) $\rightarrow$ Рейс; (для кожного пілота, дати і часу вильоту можливий тільки один рейс);

(Рейс, Дата вильоту) $\rightarrow$ Пілот (на певний день і рейс призначається певний пілот).

4.3.2. Типи функціональних залежностей

Притаманні предметної області взаємозв'язки об'єктів, можуть породжувати різні типи ФЗ. Розглянемо приклади.

Функціональна взаємозалежність

Якщо одночасно існують функціональні залежності виду **A → B** і **B → A**, тоді має місце функціональна взаємозалежність, яка зображується **A ↔ B**.

Приклад функціональної взаємозалежності: Табельний номер ↔ Номер паспорта

Повна функціональна залежність.

В випадку, коли детермінант залежності складається з кількох атрибутів, ФЗ називається повною, якщо існує ФЗ від всього набору атрибутів А і не існує функціональної залежності від будь-якої підмножини А

$$\mathbf{A \rightarrow B}$$

$$\mathbf{A_1 \subset A \quad \Rightarrow \quad A_1 \nrightarrow B}$$

інакше залежність називається неповною.

Приклад. Відношення Замовлення:

Замовлення (Номер замовлення, Код товару, Кількість Товару, Найменування товару)

- повна функціональна залежність:

(Номер замовлення, Код товару) $\rightarrow$ Кількість Товару

- неповна функціональна залежність:

(Номер замовлення, Код товару) $\rightarrow$ Найменування товару.

Надлишкова функціональна залежність

Надлишкова функціональна залежність – залежність, що містить у собі таку інформацію, яка може бути отримана на основі інших залежностей, наявних в базі даних. Наявність таких залежностей у схемі бази даних вважається небажаною, оскільки вони є надмірними і можуть призводити до проблем при модифікації даних.

Транзитивна функціональна залежність

Транзитивна функціональна залежність **A → B** виникає якщо виконуються умови:

- існує набір атрибутів **C** такий, що:

$$\mathbf{C \not\subset A}$$

$$\mathbf{B \not\subset C}$$

- існує функціональна залежність

$$\mathbf{A \rightarrow C}$$

- не існує функціональної залежності **C → A** (А не є підмножиною С)
- існує функціональна залежність

$$\mathbf{C \rightarrow B.}$$

Приклад транзитивної залежності

СПІВРОБІТНИКИ (Номер_співробітника, Код_відділу, Найменування_відділу, ...)

Транзитивна функціональна залежність:

Номер_співробітника $\rightarrow$ Найменування_відділу

Тривіальна функціональна залежність

Функціональна залежність **X → Y** є тривіальною тоді і тільки тоді, коли її права (залежна) частина **Y** є підмножиною її лівої частини (детермінанта) **X** .

$$\mathbf{Y \subseteq X.}$$

Наприклад, **(A, B) → B** є тривіальною функціональною залежністю, оскільки **B** є підмножиною (**A, B**),

(Employee_ID, Employee_Name) $\rightarrow$ Employee_ID також є тривіальною функціональною залежністю, оскільки Employee_ID є підмножиною (Employee_ID, Employee_Name).

Багатозначна функціональна залежність

Вона виникає, коли в одній таблиці існує більше одного взаємонезалежного атрибута з декількома значеннями. Це можна представити наступним чином:

$$A \rightarrow B \qquad B \not\leftrightarrow C$$
$$A \rightarrow C \qquad B \not\leftrightarrow D$$
$$A \rightarrow D \qquad C \not\leftrightarrow D$$

Тут A, B, C і D – атрибути однієї таблиці, де A – первинний ключ, а B, C і D – неключові атрибути. Причому, B, C і D функціонально залежать від A, але не залежать один від одного (взаємо незалежні). У цьому прикладі кажуть, що ці три атрибути (B, C, D) є багатозначно залежними від атрибуту A.

4.4. Аксіоми виведення (Армстронга)

В процесі проєктування бази даних у процесі обстеження спочатку визначають функціональні залежності сутностей і атрибутів. В подальшому функціональна залежність використовується для:

- перевірки створених відношень на сумісність із заданим набором функціональних залежностей. Якщо відношення **r** є допустимим для набору функціональних залежностей **F**, ми говоримо, що **r** задовольняє **F**;
- встановлення обмеження на множину допустимих відношень **R**. Будемо говорити, що **F** виконується на **R**, якщо всі відношення на **R** задовольняють множині функціональних залежностей **F**.

Замикання функціональних залежностей

Часто **F** – це набір найбільш очевидних і найважливіших функціональних залежностей, а **F+** замикання, тобто це набір усіх функціональних залежностей, включаючи **F**, які можна вивести з **F**. Замикання є важливим і, наприклад, може бути потрібні для пошуку одного або кількох ключів -кандидатів таблиці.

Щоб визначити **F+**, існують правила для виведення всіх функціональних залежностей, які передбачені **F**. Набір правил, які можуть бути використані для визначення додаткових залежностей, був запропонований Армстронгом у 1974 р. Ці правила (або аксіоми) є повним набором правил в тому, що з них можуть бути виведені всі можливі функціональні залежності.

Правила Армстронга

Для відношення **R (A, B, C,....)** з атрибутами **U= (A, B, C,....)** і множиною функціональних залежностей **F**, заданих на множині U правила мають вигляд:

основні правила

- F_1. Рефлексивність: $B \subseteq A, \Rightarrow A \rightarrow B$;
- F_2. Доповнення: $A \rightarrow B \Rightarrow AC \rightarrow BC$ $(C \subset R)$;
- F_3.Транзитивність: $A \rightarrow B$ і $B \rightarrow C \Rightarrow A \rightarrow C$;

додаткові правила

- F_4. Об’єднання: $A \rightarrow B$ і $A \rightarrow C \Rightarrow A \rightarrow BC$;
- F_5. Проєктивність: $A \rightarrow BC \Rightarrow A \rightarrow B$ і $A \rightarrow C$.

AC означає конкатенацію атрибутів A і C

Приклад. Розглянемо відношення зі схемою R = (A, B, C, G, H, I) з заданою множиною функціональних залежностей F

F = (
A → B
A → C
CG → H
CG → I
B → H
).

Знайдемо деякі члени F+:

- A → H (За транзитивністю з A → B та B → H виконується A → H);
- AG → I (Конкатенація залежності A → C з G, отримуємо AG → CG, а потім за транзитивністю з CG → I отримуємо AG → I);
- CG → HI (За правилом об'єднання CG →H і CG →I, маємо CG →HI).

Ліва та права частини функціональної залежності є підмножинами R.

Мінімальне покриття

Коректної вважається така схема бази даних, в якій відсутні надлишкові функціональні залежності. Набір не надлишкових ФЗ, отриманий шляхом видалення всіх надлишкових ФЗ з вихідного набору, називається мінімальним покриттям.

Мінімальне покриття може бути не єдиним.

Мінімальне покриття для набору функціональних залежностей **F** задається набором функціональних залежностей **E** таким чином, що:

- Кожна залежність у **E** має вигляд **X → Y**, де **Y** є єдиним атрибутом.
- Жодна залежність **X → Y** не може бути замінена на залежність **Z → Y**, де **Z** є підмножиною **X**
- Жодна залежність на **E** не може бути видалена.

Алгоритм пошуку мінімального покриття:

1. Перетворіть кожну функціональну залежність у **F** на одну або кілька еквівалентних ФЗ з правою частиною, що має лише один атрибут.
2. Мінімізувати ліву частину кожної ФЗ, отриманої на кроці 1.
3. Видалити зайві ФЗ з переліку ФЗ, отриманих на кроці 2.

Для виконання алгоритму потрібно визначити так звані зайві атрибути. Атрибут функціональної залежності вважається зайвим, якщо його можна вилучити, не змінюючи закриття набору функціональних залежностей.

Приклад. Зайві атрибути в наслідок транзитивної залежності:

A→C зайве в множині ФЗ: A→B, B→C, A→ C

Приклад. Зайві атрибуту в залежній частині:

Множину ФЗ A→B, B→C, A→C,D

можна спростити до

A→B, B→C, A→D, оскільки A→C випливає з A → B, B→C.

Приклад. Зайві атрибути в детермінанті:

A→B, B→C, A,C→D

можна спростити до

A→B, B→C, A→D, оскільки A→C.

Приклад. Визначення мінімального покриття ФЗ:

Розглянемо наступну множину функціональних залежностей F на схемі R = (A, B, C)

F = (
F1 A → BC
F2 B → C
F3 A → B
F4 AB → C).

1. Оскільки F1 можна представити в вигляді двох ФЗ з одним аргументом в правої частині, то F3 зайва, і множина ФЗ приймає вигляд (A →BC, B →C, AB→C).
2. Детермінант F4 AB→C не є мінімальним. Атрибут A є зайвим, оскільки є залежність A→C і якщо видалити A, залежність B → C зберігається і інші залежності не змінюються. Результуюча множина ФЗ матиме вигляд (A →BC, B →C, B→C).
3. B→C вже присутня у множині, тому видаляється. Множина ФЗ має вигляд (A →BC, B →C).
4. Атрибут C є зайвим у множині A→BC, оскільки його вилучення надає залежність A → B і інші залежності не змінюються. Вилучивши C з A→BC, отримаємо (A →B, B →C).

Мінімальне покриття має вигляд: A → B, B → C.

4.5. Послідовність нормальних форм

Маючи схему відношення, необхідно вирішити, чи є вона вдалим дизайном, чи потрібно зробити декомпозицію на менші відношення. Таке рішення має ґрунтуватися на розумінні того, які проблеми, якщо такі є, виникають з поточною схемою. Для цього виконується процес, що має назву нормалізацій. Метою нормалізації є створення стабільного набору відношень, який є достовірною моделлю предметній області. Для забезпечення такого рішення було запропоновано декілька нормальних форм. Вважається, що реляційна таблиця має певну нормальну форму, якщо вона задовольняє певний набір обмежень.

Нормалізація відбувається у декілька етапів [4]. Перші три етапи описуються як перша нормальна форма (1NF), друга нормальна форма (2NF) і третя нормальна форма (3NF). Зі структурної точки зору, 2NF краща за 1NF, а 3NF краща за 2NF. При переході до наступної нормальній формі властивості попередніх нормальних форм зберігаються.

Послідовність процесу нормалізації наведено на рис. 4.4.

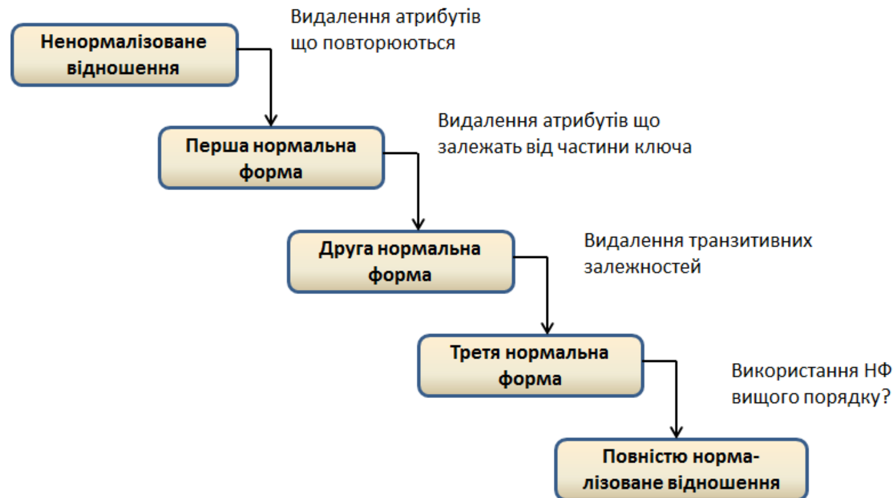


Рис. 4.4. Послідовність процессу нормалізації

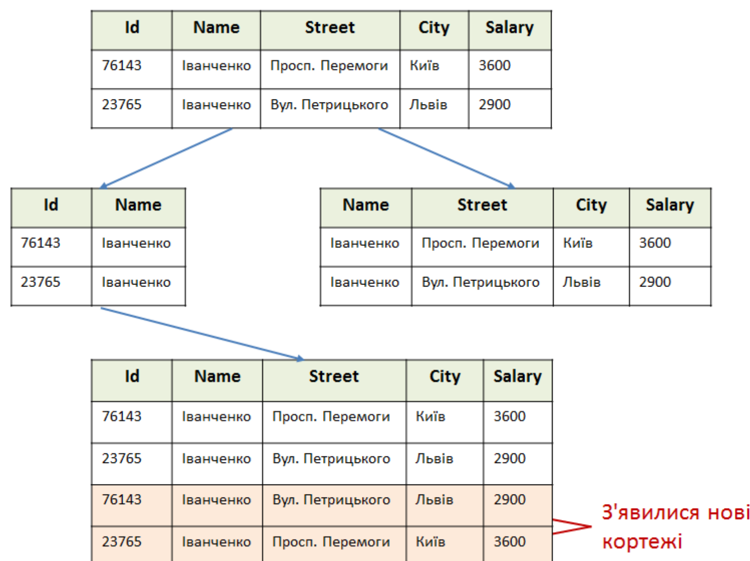
4.5.1. Декомпозиція відношень

При нормалізації схем відношень, послідовний перехід від однієї нормальної форми до інший реалізується через декомпозицію. Основний операцією, за допомогою якої здійснюється декомпозиція, є проекція.

Декомпозицією схеми відношення **R = (A_1, A_2, ...A_n)** називається заміна її сукупністю відношень **R_i**. При цьому допускається, щоб підмножини перетинались.

Декомпозиція без втрат – це декомпозиція, що забезпечує еквівалентність схем БД при заміні однієї схеми на іншу [5]. Схеми БД називаються еквівалентними, якщо зміст початкової БД може бути отримано шляхом природного з'єднання відношень, що входять в результуючу схему, і при цьому не з'являється нові кортежі. Нові відношення повинні зберігати функціональні залежності.

Приклад невдалої декомпозиції наведено на рис. 4.5. Природне з'єднання нових відношень призводить до появи кортежів, яких не було спочатку.



Id	Name	Street	City	Salary
76143	Іванченко	Просп. Перемоги	Київ	3600
23765	Іванченко	Вул. Петрицького	Львів	2900

Id	Name
76143	Іванченко
23765	Іванченко

Name	Street	City	Salary
Іванченко	Просп. Перемоги	Київ	3600
Іванченко	Вул. Петрицького	Львів	2900

Id	Name	Street	City	Salary
76143	Іванченко	Просп. Перемоги	Київ	3600
23765	Іванченко	Вул. Петрицького	Львів	2900
76143	Іванченко	Вул. Петрицького	Львів	2900
23765	Іванченко	Просп. Перемоги	Київ	3600

Рис. 4.5. Приклад некоректної декомпозиції відношень

Алгоритм декомпозиції засновано на наступній теоремі Хіта.

Нехай **R (A, B, C)** – відношення, **A, B, C** – атрибути. Якщо **R** задовольняє залежності **A → B**, то **R** дорівнює природному з'єднанню його проекцій (**A, B**) і (**A, C**):

$$\mathbf{R\ (A, B, C) \Rightarrow R\ (A, B)\ JOIN\ R\ (A, C)}$$

Приклад. Відношення R (A, B, C, D), A, B, C, D – атрибути. Функціональні залежності: A → B, C → D

1. Виносимо функціональну залежність в нове відношення

   $R\ (A, B, C, D) \Rightarrow R_1\ (A, B, C),\quad R_2\ (C, D)$

2. Теж саме для відношення R_1 (A, B, C)

   $R_1\ (A, B, C) \Rightarrow R_3\ (A, C),\quad R_4\ (A, B)$

3. Результат

   $R\ (A, B, C, D) \Rightarrow R_4\ (A, B),\ R_2\ (C, D),\ R_3\ (A, C).$

Алгоритм декомпозиції відношень:

1. Розробка універсального відношення для бази даних.
2. Визначення всіх ФЗ між атрибутами відношення.
3. Визначення, чи знаходиться в BCNF[3]. Якщо так, то завершити проєктування; в іншому випадку відношення має бути розбите на два інших відношення.
4. Повторення пунктів 2 та 3 для кожного нового відношення, отриманого в результаті декомпозиції.

Якщо ФЗ мають транзитивний характер і утворюють послідовність ФЗ, корисно застосувати емпіричне «правило ланцюжка». Правило ланцюжка полягає в наступному – як ФЗ для здійснення проекції використовується крайня права залежність або «кінець ланцюжка».

Таким чином, з точки зору теорії реляційних баз даних, хороша схема відношень реляційної бази даних повинна по можливості відповідати наступним вимогам:

- Виключати надмірне дублювання;·
- Виключати потенційну суперечливість даних;·
- Мати властивість з'єднання без втрат;·
- Мати властивість збереження ФЗ.

У процесі нормалізації вихідної схеми відношень, що задаються інформаційною моделлю даних, мають бути отримані схеми відношень, що відповідають вищезазначеним вимогам.

4.5.2. Перша нормальна форма (1NF)

Відношення знаходиться в 1NF якщо:

- значення атрибутів є атомарними,
- визначено первинний ключ необхідний для забезпечення відсутності дублікатів,

[3] Нормальна форма Бойса-Кодда (BCNF) розглядається далі

- не існує повторюваних груп.

Повторювані групи – групи атрибутів, що містять однакові за змістом значення. Приклади груп що повторюються:

1. **Робітники** (Таб№, ПІБ, Адреса, **Дитина 1, Дитина 2**,..). Заздалегідь невідомо скільки дітей у робітника, тому схему зробили «с запасом».
2. **Студенти** (Залік№, ПІБ, **Семестр1, Семестр2,…,Семестр12**). В відношення фіксується факт оплати за навчання. Передбачено 12 семестрів оплати, на практиці може бути менше.

Причина, чому необхідно уникнути проблеми повторюваних груп, полягає в тому, що повторювані групи призводять до наступних аномалій:

- Деякі записи дають значення Null. Наприклад, деякі батьки мають тільки одну дитину, що призведе до Null значення в полі ДИТИНА2,… .
- Структура відношення може бути нестабільною. Наприклад, деякі батьки мають чотири дитини або навіть більше, тому потрібно часто оновлювати структуру таблиці, щоб адаптуватися до нових ситуацій.
- Неоднозначність виникає при використанні даних. Кого з дітей слід поставити на перше місце? Кого поставити на друге місце? Які правила? Всі ці питання можуть призвести до семантичної плутанини та неоднозначності у використанні даних для обробки.

Рішення полягає в тому, щоб перетворити повторювані групи у окрему таблицю:

Робітники (Таб№, ПІБ, Адреса), **Діти**(Таб№, Дитина);

Студенти (Залік№, ПІБ), **Сплата** (Залік№, Семестр№).

Але, відношенню, що знаходиться в 1NF притаманні аномалії. Розглянемо на прикладі відношення Замовлення (рис.4.6).

Замовлення

Параметри обліку (атрибути)	Тип даних
Номер замовлення	числовий
Код товару	числовий
Найменування товару	текстовий
Ціна товару в замовленні	числовий
Кількість товару в замовленні	текстовий
Одиниця виміру товару	текстовий
Код типу товару	числовий
Найменування типу товару	текстовий

Рис. 4.6. Відношення Замовлення

Аномалія включення. Оскільки код товару є частиною первинного ключа, поки не буде замовлення на товар інформація про товар буде відсутня в базі даних (найменування товару, код типу товару, найменування типу товару);

Аномалія оновлення. При зміні найменування товару необхідний повний перегляд відношення з метою знайти всі замовлення, щоб зміна найменування товару було відображено у всіх замовленнях.

Аномалія видалення. Якщо який-небудь товар видалити з бази даних (знятий з продажу), то при цьому буде втрачено не тільки інформація про тих замовленнях, в яких присутній цей товар, але і товар в цілому (код товару, його найменування і тип).

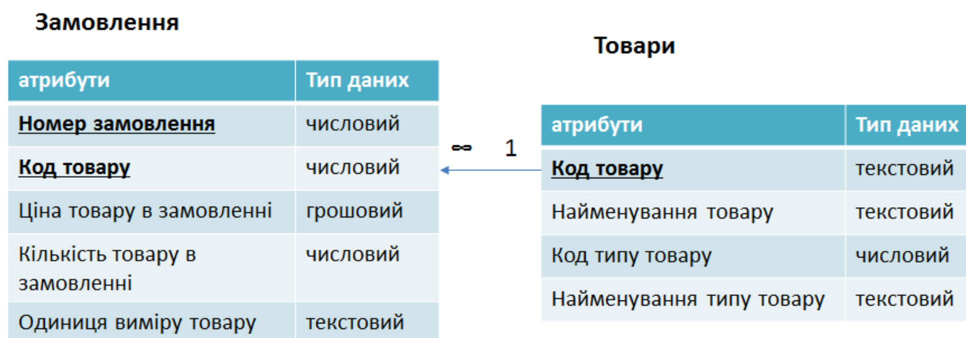
Аномалія дублювання. Деякі значення атрибутів необхідно багаторазово повторювати (найменування товару і найменування типу товару).

4.5.3. Друга нормальна форма (2NF)

Відношення знаходиться в другій нормальній формі тоді і тільки тоді, коли воно знаходиться в першій нормальній формі і не містить неповних функціональних залежностей неключових атрибутів від атрибутів первинного ключа.

Існує дві необхідні умови для виконання 2NF: по-перше, має виконуватися 1NF; по-друге, кожен неключовий атрибут повинен функціонально повно залежати від будь-якого з ключів-кандидатів. Це можна просто зрозуміти – всі атрибути не первинного ключа залежать від усього первинного ключа, а не від його частини. Те, що показано на рис. 4.6 не задовольняє 2NF, оскільки найменування товару залежить тільки від коду товару і не має нічого спільного з номером замовлення. Тому в таблиці буде багато надлишкових даних, оскільки код товару повторюється.

Декомпозиція (рис. 4.7) дозволяє отримати відношення, що відповідають умовам 2NF.


Замовлення

атрибути	Тип даних
Номер замовлення	числовий
Код товару	числовий
Ціна товару в замовленні	грошовий
Кількість товару в замовленні	числовий
Одиниця виміру товару	текстовий

Товари

атрибути	Тип даних
Код товару	текстовий
Найменування товару	текстовий
Код типу товару	числовий
Найменування типу товару	текстовий

Рис. 4.7. Відношення у 2NF

Алгоритм приведення до 2NF

- вихідне відношення $R_1 = (\underline{K_1, K_2}, A_1,\ldots A_n, B_1,\ldots B_m)$;
- ключ (K_1, K_2) – складається з кількох атрибутів.

Функціональні залежності:

- залежність всіх атрибутів від ключа відношення

$$(K_1, K_2) \rightarrow (A_1,\ldots A_n, B_1,\ldots B_m);$$

- залежність деяких атрибутів від частини складного ключа

$$(K_1) \rightarrow (A_1,\ldots A_n).$$

Декомпозиція відношення:

- Атрибути з неповною функціональною залежністю, винесені з початкового відношення разом з частиною складеного ключа (K_1), від якої вони залежать.

$$R2 = (\underline{K_1}, A_1, ... A_n)$$

- Залишається відношення з рештою атрибутів відношення з складеним ключем (K_1, K_2).

$$R_3 = (\underline{K_1, K_2}, B_1, ... B_m).$$

Приклад. Відношення Студенти Stud (ПІБ, Номер зал.кн., Група, Дисципліна, Оцінка).

Аномалія включення. Не можна доповнити відношення Stud даними про студента, який в даний час ще не здавав жодного іспиту (дисципліна є частиною первинного ключа і не може містити невизначених значень). Тим часом студент вже зарахований до інституту.

Аномалія оновлення. Щоб змінити номер групи студента, ми будемо змушені модифікувати всі кортежі з відповідним значенням атрибута Номер.зал.кн. Виконаємо декомпозицію:

- початкова схема:
  Stud = (ПІБ, Номер зал.кн., Група, Дисципліна, Оцінка)
- схема після декомпозиції:
  R_1 = (ПІБ, Номер.зал.кн., Група)
  R_2 = (Номер зал.кн., Дисципліна, Оцінка).

4.5.4. Третя нормальна форма (3NF)

Якщо повернутись до відношення Замовлення, що приведено до 2NF (рис. 4.7), бачимо що йому притаманні аномалії.

Аномалія включення. Неможливо зберегти дані про новий тип товару, поки не з'явиться товар з новим типом. (Первинний ключ не може містити невизначені значення.);

Аномалія оновлення. При зміні типу товару в деякої групі товарів, ми будемо змушені змінити значення атрибута «Тип товару» в кортежах всіх товарів, яких відносяться до цієї групи.

Аномалія видалення. При зняття товару з продажу ми втратимо інформацію про наявність такого типу.

Аномалії виникли внаслідок того, що в відношення існує транзитивна залежність.

Відношення знаходиться в третій нормальній формі (3NF) тоді і тільки тоді, коли воно знаходиться в другій нормальній формі і не містить транзитивних залежностей неключових атрибутів від інших неключових атрибутів. Тобто, все не ключові атрибути ключа залежать від усього первинного ключа, а не від інших не ключових атрибутів. Всі неключові атрибути повинні бути взаємно функціонально незалежні.

Найменування типу товару, показане на рис. 4.7, залежить від не ключового атрибуту «Код Типу Товару», тому умова 3NF не виконується.

Для приведення відношень до 3NF потрібна подальша декомпозиція (рис. 4.8).

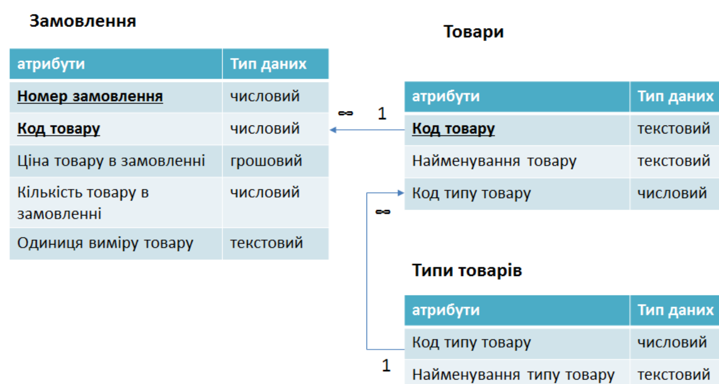


Рис. 4.8. Відношення в 3NF

Алгоритм приведення до 3NF

Початкове відношення: R = ($\underline{K}$, A_1,... A_n, B_1,... B_m), ключ: (K).

Функціональні залежності:

- залежність всіх атрибутів від ключа відношення

$$(K) \rightarrow (A_1,\dots A_n, B_1,\dots B_m),$$

- залежність деяких не ключових атрибутів від інших не ключовим атрибутів

$$(A_1,\dots A_n) \rightarrow (B_1,\dots B_m).$$

Декомпозиція відношення:

- неключових атрибути, винесені з початкового відношення разом з детермінантом функціональної залежності неключових атрибутів:

$$R_2 = (\underline{A_1,\dots A_n}, B_1,\dots B_m), \text{ ключ: } (A_1, \dots A_n);$$

- залишилися відношення з рештою атрибутів початкового відношення:

$$R_3 = (\underline{K}, A_1,\dots A_n), \text{ ключ: } (K).$$

У практичних застосуваннях достатньо, щоб реляційна модель задовольняла 3NF.

Приклад. Предметна область Бібліотека. Початкове відношення

Бібліотека (ISBN, title, author(name, date_of_birth), publisher(name, address(streetnr, streetname, zipcode, city)), pages, price)

Початкові дані:

- кожна книга має унікальний номер ISBN;
- кожен автор має унікальне ім’я;
- кожен видавець має унікальну назву;
- книга має лише одного видавця;
- видавець має лише одну адресу.
- книга може мати декількох авторів;
- автор може написати більше однієї книги;
- видавець може видати більше однієї книги.

Функціональні залежності:

ISBN → title, pages, price, publisher

AuthorName → date_of_birth

PublisherName → streetnr, streetname, zipcode, city.

Якщо винести ФЗ в окремі відношення, залишиться відношення з такими атрибутами R(ISBN, author). Це відношення моделює ситуацію, коли у автора багато книжок, а у книги може бути кілька авторів (рис. 4.9).

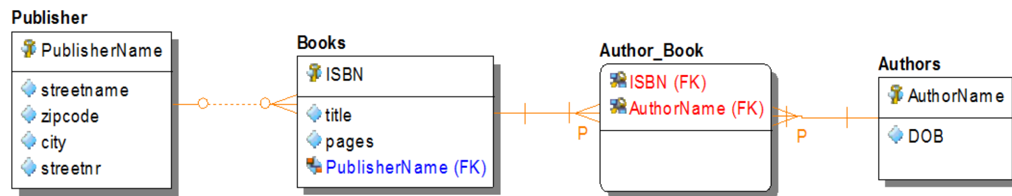


Рис. 4.9. Відношення Бібліотека в 3NF

Порівняння нормалізованих і ненормалізованих моделей наведено на рис. 4.10.

Критерій	Відношення слабо нормалізовані (1НФ, 2НФ)	Відношення сильно нормалізовані (3НФ)
Адекватність бази даних предметної області	ГІРШЕ (-)	КРАЩЕ (+)
Легкість розробки і супроводу бази даних	СКЛАДНІШЕ (-)	ЛЕГШЕ (+)
Швидкість виконання вставки, оновлення, видалення	ПОВІЛЬНІШЕ (-)	ШВИДШЕ (+)
Швидкість виконання вибірки даних	ШВИДШЕ (+)	ПОВІЛЬНІШЕ (-)

Рис. 4.10. Порівняння нормалізованих і ненормалізованих моделей

4.5.5. Нормальна форма Бойса–Кодда (Boyce–Codd normal form, BCNF)

Є один окремий випадок, який майже відповідає вимогам 3NF, але, тим не менш, породжує аномалії оновлення. Це той випадок, коли:

- є декілька потенційних ключів у відношенні;
- ці потенційні ключі складові;
- деякі з цих можливих ключів «перекриваються», т. е. містять загальні атрибути.

Відношення знаходиться в нормальній формі Бойса–Кодда (BCNF), якщо воно знаходиться в третій нормальній формі і кожен детермінант функціональної залежності відношення є можливим ключем відношення.

Приклад. Співробітник_Завдання = (ТабНомер, ІПН, Проєкт, Завдання)

Аномалія поновлення. У разі якщо ключем є ІПН, і змінився табельного номера службовця, потрібно оновити атрибут ТабНомер у всіх кортежі відношення Співробітник_Завдання, які відповідають цьому службовцю. Інакше буде порушена функціональна залежність ТабНомер → ІПН, І база даних виявиться в неузгоджену стані.

Функціональні залежності: ТабНомер → ІПН і ІПН → ТабНомер. Детеринанти ФЗ: ТабНомер, ІПН. Необхідно виконати декомпозицію таким чином, щоб детермінанти ФЗ були первинними ключами (первинні ключі підкреслено):

- початкове відношення:
  Співробітник_Завдання (ТабНомер, ІПН, Проєкт, Завдання);
- результат декомпозиції:

  Співробітник_Завдання (ТабНомер, Проєкт, Завдання);

  Співробітник (ІПН, ТабНомер).

Приклад. Відношення моделює стан сесії
R = (Номер зал.кн., ІПН, Дисципліна, Дата, Оцінка).

Можливі ключі:

(Номер_зал.кн, Дисципліна, Дата);

(ІПН, Дисципліна, Дата).

Функціональні залежності:

1. Номер_зал.кн, Дисципліна, Дата $\rightarrow$ Оцінка;
2. ІПН, Дисципліна, Дата $\rightarrow$ Оцінка;
3. Номер зал.кн. $\rightarrow$ ІПН;
4. ІПН $\rightarrow$ Номер зал.кн.

Аномалії поновлення. У разі якщо в ключ відношення входить ІПН і необхідно змінити номер залікової книжки, потрібно оновити атрибут Номер зал.кн. у всіх кортежі відношення R, які відповідають цьому студенту. Інакше буде порушена функціональна залежність 1 і база даних виявиться в неузгоджену стані. Необхідно виконати декомпозицію:

- початкове відношення R = (Номер зач.кн., ІПН, Дисципліна, Дата, Оцінка);
- результат декомпозиції

  R1 = (ІПН, Дисципліна, Дата, Оцінка);

  R2 = (Номер зал.кн., ІПН);

або так:

  R1 = (Номер зал.кн., Дисципліна, Дата, Оцінка);

  R2 = (ІПН, Номер зал.кн.).

4.5.6. Четверта нормальна форма (4NF)

Багатозначна залежність

Багатозначна залежність має місце в тому відношенні, в якому міститься два незалежних зв’язку типу 1:N [5]. Це може відбуватися в таблиці з трьома або більше атрибутами, коли **всі атрибути в таблиці є частиною складеного ключа**.

В відношенні **R (A, B, C)** існує багатозначна залежність $\mathbf{A} \twoheadrightarrow \mathbf{B}$ в тому і тільки в тому випадку, якщо множина значень **B**, відповідне парі значень **A** і **C**, та залежить тільки від **A** і не залежить від **C**. Іншими словами, за наявності двох кортежів $\mathbf{t_1}$ і $\mathbf{t_2}$ таких, що $\mathbf{t_1[X] = t_2[X]}$, зобов’язані також існувати кортежі $\mathbf{t_3}$ і $\mathbf{t_4}$ і виконуватися умови:

а) $\mathbf{t_3}[A] = \mathbf{t_4}[A] = \mathbf{t_1}[A] = \mathbf{t_2}[A]$;

б) $\mathbf{t_3}[B] = \mathbf{t_1}[B]$ і $\mathbf{t_4}[B] = \mathbf{t_2}[B]$;

в) $\mathbf{t_3}[C] = \mathbf{t_2}[C]$ і $\mathbf{t_4}[C] = \mathbf{t_1}[C]$

Приклад. Відношення моделює мережу СТО.

СТО	Спеціалізація	Район міста
Автосервіс	Ford	Святошин
Автосервіс	Ford	Дарніца
Автосервіс	Ford	Троєщина
Автосервіс	Volvo	Святошин
Автосервіс	Volvo	Дарніца
Автосервіс	Volvo	Троєщина
A1 Service Centre	Peugeot	М. Київ
A1 Service Centre	Volvo	М. Київ
Super Service Station	Volvo	Святошин
Super Service Station	Volvo	Дарніца
Super Service Station	Peugeot	Дарніца

Рис. 4.11. Приклад багатозначної залежності

Аномалія поновлення виникає тому, що по відношенню до СТО є:

- залежність множини значень атрибута Район від атрибута СТО
  - СТО —» Район ;
- залежність множини значень атрибута Спеціалізація від атрибута СТО
  - СТО —» Спеціалізація.

Відношення СТО(Назва, Спеціалізація, Район) знаходиться в 3NF, але:

- Спеціалізація не залежить від Район;
- Спеціалізація залежить від СТО;
- Район залежить від СТО;
- Район не залежить від Спеціалізація.

Четверта нормальна форма стосується багатозначної залежності. У цьому випадку може знадобитися розкласти таблицю на 2 або більше таблиць.

Вважається, що відношення знаходиться у 4NF, якщо кожна таблиця містить не більше однієї багатозначної залежності на ключовий атрибут.

Декомпозиція початкового відношення:

- СТО1 (Назва, Район),
- СТО2 (Назва, Спеціалізація).

Розглядаючи схему відношення, можна піддати її серії перевірок на предмет того, чи перебуває вона в тій чи іншій нормальній формі. Якщо необхідної глибини нормалізації не досягнуто, можна провести декомпозицію схеми.

Важливо розуміти, що сама по собі відповідність схеми відношення деякій нормальній формі не є ознакою добре виконаного проєктування – це лише одна з умов. До уваги слід брати як універсальні вимоги до будь-якої бази даних, так і інші чинники, що залежать від предметної області, особливостей обраної технологічної платформи тощо.

На практиці нормалізацію часто припиняють на рівні 3NF, тому що нормальні форми вищих порядків часто не приносять додаткових переваг схемі бази даних, а лише

ускладнюють її. Іноді нормалізація може бути припинена і на нижчих рівнях (наприклад, на рівні 2NF).

4.5.7. Денормалізація схеми відношень

Денормалізація – процес приведення схем відношень до нижчої нормальної форми шляхом їх з'єднання. Наприклад, якщо в процесі проєктування відмовляються від 3-й нормальної форми і зупиняються на 2NF. Денормалізація бази проводять зазвичай для підвищення продуктивності, рідше – для полегшення життя програмістам, які розробляють програми, що працюють з цією базою даних.

Причини денормалізації

– Велика кількість з'єднань таблиць. У запитах до повністю нормалізованої базі нерідко доводиться з'єднувати до десятка, а то й більше, таблиць. А кожне з'єднання – операція вельми ресурсномістка. Як наслідок, такі запити виконуються повільно. У такій ситуації допоможе денормалізація шляхом скорочення кількості таблиць. Краще об'єднувати в одну кілька таблиць, які мають невеликий розмір, що містять умовно-постійну, або нормативно-довідкову інформацію.

- Розрахункові значення. Найчастіше повільно виконуються і споживають багато ресурсів запити, в яких виробляються якісь складні обчислення, особливо при використанні угруповань і агрегатних функцій (Sum, Max і т.п.). Іноді має сенс додати в таблицю 1-2 додаткових стовпчика, що містять часто використовувані (і складно обчислювані) розрахункові дані.
- Довгі поля. Якщо у нас в базі даних є великі таблиці, що містять довгі поля (Blob, Long і т.п.), то серйозно прискорити виконання запитів до такої таблиці можна, якщо винести довгі поля в окрему таблицю.

Як правильно проводити денормалізацію

Кінцевою метою денормалізації є *збільшення рівня надмірності даних* за рахунок приведення результуючого відношення R до більш низького рівня нормалізації, ніж вихідні відношення R_1, ..., R_n. При цьому переслідується мета скоротити кількість з'єднань, які потрібно виконувати під час роботи застосунків, за рахунок попереднього створення цих з'єднань на одному з етапів проєктування бази даних.

Денормалізація зазвичай виконується наступними способами:

- створення дублікатів атрибутів;
- виконання попереднього з'єднання;
- додавання похідних полів;
- створення зведених або тимчасових таблиць;
- горизонтальне або вертикальне розділення таблиць.

Якщо проведено денормалізацію, в базі даних неминуче створюються дані, що дублюються. Тому перед розробниками відразу виникає завдання забезпечити несуперечливість (а частіше – ідентичність) даних що дублюються. Це необхідно робити через механізм тригерів. Наприклад, при додаванні обчислюваного поля, на кожен з стовпців, від яких обчислюється поле залежить, «вішається» тригер, що викликає єдину

(це важливо!) збережену процедуру, яка і записує потрібні дані в обчислюване поле. Треба тільки не пропустити жоден з стовпців, від яких залежить обчислюване поле.

Для OLAP-систем негативний вплив денормалізації відносно великий, але для систем, які вимагають обробки великої кількості транзакцій (OLTP), забезпечується висока продуктивність паралельної обробки. При проєктуванні реляційної бази даних майже завжди домагаються подання усіх відношень в другій нормальній формі. У часто оновлюваних базах даних зазвичай намагаються забезпечити третю нормальну форму. Нормальну форму Бойса-Кодда використовують набагато рідше, оскільки ситуації, в яких у відношення є кілька складових ключів, що перекриваються зустрічаються не часто.

Контрольні запитання

1. У чому полягає призначення методів нормалізації даних?
2. Назвіть типи аномалій оновлення, які можуть виникати у відношенні, в якому є надлишкові дані.
3. Дайте визначення поняття функціональної залежності.
4. Назвіть основні характеристики функціональних залежностей, які використовують під час нормалізації відношення.
5. Опишіть спосіб перетворення відношення в не нормалізованій формі у відношення в першій нормальній формі.
6. Назвіть нормальну форму, якій, як мінімум, має задовольняти кожне відношення. Дайте визначення цієї нормальної форми.
7. Чим небезпечне надмірне дублювання інформації?
8. Назвіть основні властивості нормальних форм.
9. Які обмеження таблиць відносять до 1NF, 2NF і 3NF?
10. Наведіть приклади таблиць, що відповідають і не відповідають вимогам нормальних форм.
11. Опишіть три типи аномалій, які можуть виникнути в таблиці, та негативні наслідки кожного з них.
12. Що таке багатозначна залежність?
13. Чи накладає нормалізація якісь обмеження на продуктивність їх обробки? Поясніть
14. Опишіть, як можна усунути транзитивні залежності у відношенні, якщо вони призводять до аномалій.
15. Перелічіть умови, які можна застосувати, щоб визначити, чи є відношення, яке знаходиться в першій нормальній формі, також і в другій нормальній формі.

Тестові завдання

1. Обов’язкова умова для першої нормальної форми?
   A. Кожен атрибут таблиці атомарний і всі рядки різні
   B. Усі атрибути таблиці є простими і кожен неключовий атрибут функціонально повно залежить від ключа
   C. Кожен неключовий атрибут нетранзитивно залежить від первинного ключа
   D. Кожен атрибут таблиці атомарний і всі рядки однакові

2. Яке з наступних тверджень є правильним?
   A. Усі функціональні залежності є відношеннями типу «багато до багатьох»
   B. Усі функціональні залежності є зв'язками типу «багато до одного»
   C. Усі функціональні залежності є відношеннями типу «один до одного»
   D. Жодне з перерахованих вище
3. Якщо атрибути A та B визначають атрибут C, то вірно, що
   A. $A \rightarrow C$
   B. $B \rightarrow C$
   C. (A,B) – складений детермінант
   D. C – детермінант
4. Яке з наступних тверджень є правильним?
   A. Функціональні залежності не пов'язані з таблицями; вони базуються на семантиці інформації, з якою ми маємо справу
   B. Якщо таблиця не містить надлишкової інформації, її атрибути не повинні мати функціональних залежностей
   C. Функціональні залежності можуть бути визначені, якщо ми маємо декілька екземплярів таблиці
   D. Нічого з перерахованого вище
5. Кожного разу, коли з'являється атрибут A, йому ставиться у відповідність одне і те ж значення атрибута B, але не одне і те ж значення атрибута C. Тому вірно, що:
   A. $A \rightarrow B$
   B. $A \rightarrow C$
   C. $A \rightarrow (B,C)$
   D. $(B,C) \rightarrow A$
6. Що з наведеного нижче найкраще описує функціональну залежність?
   A. Описує зв'язок між атрибутами сутності
   B. Описує зв'язок кортежів в сутності
   C. Описує відношення сутностей з іншими сутностями
   D. Описує ставлення ненормалізованих даних до 1NF
7. Що з наведеного нижче не є характеристикою 1NF?
   A. Дані зберігаються в одній таблиці
   B. Стовпець може містити лише одне значення
   C. Кожен стовпець містить атомарне значення
   D. Сутність пов'язана з іншою сутністю за допомогою поля зовнішнього ключа
8. В якій нормальній формі видаляються транзитивні залежності?
   A. 1NF
   B. 2NF
   C. 3NF
   D. 4NF
9. В якій нормальній формі видаляються повторювані значення в атрибутах?
   A. 1NF
   B. 2NF
   C. 3NF

D. 4NF

10. Що з наведеного нижче є тривіальною функціональною залежністю?

A. $A \rightarrow B$, якщо B є підмножиною множини A

B. $A \rightarrow A$

C. $B \rightarrow B$

D. Все перераховане вище

11. Нехай таблиця R має три ключі-кандидати A, B та (C, D). Яке з наступних тверджень не є правильним?

A. $A \rightarrow B$

B. $B \rightarrow A$

C. $A \rightarrow C$

D. $C \rightarrow AB$

12. Вважається, що відношення знаходиться у 3NF, якщо:

(а) воно знаходиться у 2NF;

(б) неключові атрибути не залежать один від одного;

(в) ключовий атрибут не залежить від частини складеного ключа;

(г) не має багатозначної залежності.

Які з цих тверджень є правильними?

A. **а** та **в**

B. **а** та **г**

C. **а** та **б**

D. **б** та **г**

13. Якщо ___, то $A \rightarrow B$ має тривіальну функціональну залежність.

A. B є підмножиною множини A

B. A є підмножиною B

C. A є підмножиною A'

D. B є підмножиною B'

14. Якщо $A \rightarrow B$ і $A \rightarrow$ ___ тоді $A \rightarrow BC$.

A. A

B. B

C. C

D. Жоден із вказаних варіантів

15. Що з наведеного нижче не є характеристикою відношення?

A. Кожен стовпчик має окрему назву (технічно називається ім'ям атрибута)

B. Кожен перетин рядка і стовпця в таблиці може містити більше одного значення

C. Всі значення в стовпчику є значеннями одного атрибуту

D. Порядок рядків і стовпців не має значення

5. МОВА SQL

Однією з основних вимог до систем управління базами даних є наявність високорівневих засобів виконання запитів. У системах, що реалізують реляційну модель даних, як такий засіб використовується мова SQL. Фактично ця мова містить повний набір операцій, необхідних для виконання будь-яких дій з базою даних. Найбільш повний опис мови SQL наведено в [16].

5.1. Загальна інформація

Історія розвитку SQL

SQL (Structured Query Language) – це скорочена назва структурованої мови запитів, яка надає засоби створення і обробки даних в реляційних БД

SQL був розроблений під назвою SEQUEL як мова обробки даних для прототипу реляційної СКБД IBM System R в середині 1970-х. У 1980 році він був перейменований на SQL (але все ще вимовляється як «sequel»). Зараз більшість людей говорять "S-Q-L" ("ess-cue-ell")

SQL є стандартною мовою для маніпулювання реляційними базами даних. Хоча версії SQL різних виробників СКБД не є однаковими, відмінності відносно незначні. Опанувавши одну версію SQL, ви можете застосувати свої навички для вивчення іншої версії SQL. У наступних прикладах використовується версія SQL - Microsoft SQL Server, якщо не вказано інше.

- 1974 – перша стаття з описом мови SEQUEL (Structured English Query Language) проєкту System / R компанії IBM.
- 1979 – з'являється перша комерційна реляційна СКБД компанії Oracle з підтримкою SQL.
- 1981 – компанія Relation Technology випускає СКБД Ingres.
- 1983 – компанія IBM оголошує про створення СКБД DB2.
- 1986 – ANSI приймає стандарт SQL-1.
- 1991 – компанія Microsoft публікує специфікацію протоколу ODBC (Open DataBase Connectivity).
- 1992 – ANSI приймає стандарт SQL-2. На даний момент більшість виробників СКБД внесли зміни у свої продукти так, щоб вони більшою мірою задовольняли стандарту SQL2.
- 1999 – ANSI / ISO опублікували стандарт SQL-3. У SQL-3 введено нові типи даних, при цьому передбачається можливість завдання складних структурованих типів даних, які більшою мірою відповідають об'єктній орієнтації. Додано розділ, який вводить стандарти на події та тригери, які раніше не розглядалися в стандарті. У рамках управління транзакціями відбулося повернення до моделі транзакцій, що допускає точки збереження (savepoints).

- 2003 – стандарт SQL2003, порівняно з SQL-3 має розширення, що стосуються об'єктно-орієнтованих типів і базових функцій OLAP.
- 2016 – Додано захист на рівні рядків, поліморфні табличні функції, підтримка JSON.
- 2019 – Описано, як SQL взаємодіє з багатовимірними масивами (MDA).

Комерційні системи пропонують більшість, якщо не всі, функції SQL-2, а також різні набори функцій від пізніших стандартів та специфічних для платформи функцій.

Форми мови SQL

- інтерактивний SQL – використовується для функціонування безпосередньо в базі даних, без додатків–посередників. За такої форми SQL, при введенні команди вона зараз же виконається і ви зможете побачити результат негайно.
- вбудований SQL – складається з команд SQL поміщених у середині програм, які зазвичай написані деякою іншою мовою (типу С #). SQL код повинен оброблятися препроцесором перед компіляцією основної програми.
- динамічний SQL – дозволяє виконувати запити, що конструюються під час виконання. SQL код повинен оброблятися препроцесором після компіляції основної програми.

Функціональні категорії команд SQL

Якщо стандарт SQL-2 виділяв три категорії команд, то в стандарті SQL2003 команди поділено на шість категорій:

- DDL (Data Definition Language, Мова Визначення Даних) складається з команд, які створюють об'єкти (таблиці, індекси, подання і так далі) в базі даних.
- DML (Data Manipulation Language, Мова Маніпулювання Даними) – це набір команд, що визначають які додавати, змінювати і видаляти дані з таблиць БД.
- DQL (Data Query Language, Мова Запитів) це набір команд, що визначають які значення представлені в таблицях в будь-який момент часу (команда SELECT);
- DCL (Data Control Language, Мова Управління Даними) складається зі засобів, які визначають чи дозволити користувачу виконувати певні дії чи ні.
- TCL (Transaction Processing Language, Мова Управління Транзакціями) складається зі засобів, які керують виконанням транзакцій.
- CCL (Cursor Control Language, Мова Управління Курсором) складається зі засобів, які керують виконанням курсорів. Курсори призначені для виконання операцій з окремими рядками однієї або декількох таблиць.

Нині застосовуються обидві термінології: SQL2 і SQL2003.

5.2. Управління базами даних за допомогою SQL

5.2.1. Синтаксис мови SQL

Щоб почати використовувати SQL, необхідно розуміти, як пишуться команди. Синтаксис SQL поділяється на чотири основні категорії:

1. Ідентифікатори. Імена, надані користувачем або системою для об'єктів бази даних, таких як бази даних, таблиці, обмеження на таблиці, стовпці в таблицях, подання та ін. В більшості популярних СКБД довжина ідентифікатора обмежена 128 символами. Назва ідентифікатора повинна починатися з літери і містити будь-які алфавітно-цифрові символи та деякі спеціальні (наприклад, _, (@), (#), $).

2. Константи. Рядки або значення, надані користувачем або системою, які не є ідентифікаторами або ключовими словами. Константами можуть бути рядки, числа, дати або логічні значення:

- числові -314; 612.716; + 551.702; 2.9E-4; -134.235E7;
- строкові 'Петренко О.І.' ; 'Київ ' ;
- дата і час 'Mar 15 2011'; '3/15/1993'; '15 - MAR - 11' ;
- логічний тип True, False і Unknown.

3. Оператори. Символи, що визначають дію, яку потрібно виконати над одним або кількома виразами, найчастіше в операторах DELETE, INSERT, SELECT або UPDATE. Оператори також використовуються при створенні об'єктів бази даних. Оператори зазвичай поділяються на такі категорії:

- арифметичні оператори;
- оператори присвоювання;
- бітові оператори;
- оператори порівняння;
- логічні оператори;
- оператори множин;
- унарні оператори.

4. Ключові і зарезервовані слова. Зарезервовані слова – це слова, значення яких дуже тісно пов'язані з роботою СКБД і їх не можна використовувати з будь-якою іншою метою. До них відносяться слова, що використовуються в операторах SQL; наприклад, «SELECT» є зарезервованим словом і не повинно використовуватися як назва таблиці. Наполегливо рекомендується ніколи не використовувати зарезервовані слова як імена, визначені користувачем.

Ключові слова можуть бути як зарезервованими або незарезервованими. Незарезервовані ключові слова, такі як назви вбудованих таблиць і функцій, мають особливе значення лише у певних контекстах; однак, щоб уникнути плутанини, краще уникати їх використання як ідентифікатори.

Списки зарезервованих слів та ключових слів широко публікуються, щоб розробники не використовували їх в якості ідентифікаторів, які можуть спричинити проблему, як в існуючих, так і в наступних версіях.

5.2.2. Оператори

Арифметичні оператори

- (+) додавання,

- (–) віднімання,
- (*) множення,
- (/) ділення,
- (%) ділення за модулем.

Оператори присвоєння

За винятком Oracle, який використовує оператор :=, оператор присвоювання (=) присвоює значення змінній або псевдоніма заголовку стовпчика. У всіх СКБД ключове слово AS може слугувати оператором для присвоювання псевдонімів заголовкам таблиць або стовпцям.

Побітові оператори

Усі бази даних підтримують побітові оператори як швидкий спосіб виконання бітових маніпуляцій між двома цифровими виразами. Типи, доступні для бітових операторів, включають BINARY, BIT, INT, SMALLINT, TINYINT і VARBINARY.

- (&) Побітове І (два операнди),
- (|) Побітове АБО (два операнди),
- (^) Побітове виключне АБО (два операнди).

Оператори порівняння

Оператори порівняння перевіряють, чи є два вирази рівними або нерівними. Результатом операції порівняння є булеве значення: True, False або Unknown. Також треба зауважити, що стандартна відповідь SQL для операції порівняння, де один або декілька виразів є Null, є повернення Null. Наприклад, вираз `23 + NULL` повертає Null, так само як і вираз `23 лютого 2022 року + NULL`.

- (=) Дорівнює,
- (>) Більше ніж,
- (<) Менше ніж,
- (>=) Більше або дорівнює,
- (<=) Менше або дорівнює,
- (<>) Не дорівнює,
- (!=) Не дорівнює (не є стандартом SQL),
- (!<) Не менше (не є стандартом SQL),
- (!>) Не більше (не є стандартом SQL).

Логічні оператори

Логічні оператори зазвичай використовуються для перевірки істинності деякої умови. Вони повертають булеве значення True, False або Unknown. Не всі системи баз даних підтримують всі оператори. Ось перелік логічних операторів і значень, які вони повертають.

- `ALL` – True, якщо всі з набору порівнянь є істина,
- `AND` – True, якщо обидва булеві вирази є істина,
- `ANY (SOME)` – True, якщо будь-яке з набору порівнянь є істина,

- `BETWEEN` – True, якщо операнд знаходиться в діапазоні, що включає нижню та верхню границю,
- `EXISTS` – True, якщо підзапит містить будь-які рядки,
- `IN` – True, якщо операнд дорівнює одному зі списку виразів або одному чи декільком рядкам, повернутим підзапитом,
- `LIKE` – True, якщо операнд відповідає шаблону,
- `NOT` – Обертає значення будь-якого іншого булевого оператора,
- `OR` – True, якщо будь-який з булевих виразів має значення істина.

Оператори обробки множин

Оператори обробки множин маніпулюють результатами двох або більше операторів SELECT і повертають єдиний результат, заснований на функції самого оператора множини. Деякі оператори множин діють так само, як внутрішні та зовнішні з'єднання. Щоб використовувати оператор множини, розмістить його після першого запиту і перед другим запитом. Кожен оператор SELECT повинен мати однакову кількість стовпців у списку елементів SELECT, а стовпці в кожному запиті повинні бути однакового або сумісного типу даних.

- `UNION` – Повертає всі рядки, вибрані у запитах, повертаючи лише один запис, якщо є дублікати;
- `UNION ALL` – Повертає всі рядки, вибрані у запитах, включаючи дублікати;
- `INTERSECT` – Повертає всі рядки, які є спільними для результатів першого та другого запитів;
- `MINUS` – Повертає всі рядки, отримані першим запитом, вилучаючи з набору записів записи, отримані другим запитом;
- `EXCEPT` – Синонім `MINUS`.

Унарні оператори

Унарні оператори виконують операцію лише над одним виразом з числовим типом даних. Загалом унарні оператори можна використовувати з будь-якими числовими типами даних, хоча побітовий оператор (~) можна використовувати лише з цілими типами даних.

- (+) Числове значення додатне,
- (–) Числове значення від'ємне,
- (~) Бітове NOT; повертає доповнення числа (не підтримується в Oracle).

Пріоритет операторів

Порядок виконання складних виразів може суттєво вплинути на результуюче значення. Оператори SQL мають різні рівні пріоритету. Оператор з вищим рівнем обчислюється перед оператором нижчого рівня. Наступний лістинг показує рівні пріоритету операторів, від найвищого до найнижчого:

- вирази в дужках ( )
- унарні оператори +, –, ~
- математичні оператори * (множення), / (ділення), % (залишок ділення)
- математичні оператори + (додавання, зчеплення), – (віднімання)

- ^побітове виключне (АБО), & (побітове І) , | (побітове АБО)
- оператори порівняння =, >, =>, <=, <>, !=, !>, !<
- NOT (логічне НІ)
- AND (логічне І)
- ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
- присвоювання змінної = .

Оператори обчислюються зліва направо, якщо вони мають однаковий пріоритет. Однак, дужки можна використовувати для перевизначення пріоритету за замовчуванням. Вирази в межах набору дужок обчислюються першими, тоді як операції поза дужками обчислюються наступними.

5.2.3. Типи даних в SQL

Для будь-якого стовпця таблиці повинен бути визначений тип даних, який забезпечує загальну класифікацію даних, що зберігаються у стовпчику. Використання певних типів даних дозволяє створювати кращі та зрозуміліші запити і допомагає контролювати цілісність даних.

Особливість реалізації типів даних SQL полягає в тому, що вони не завжди відображаються безпосередньо до ідентичних типів даних на різних платформах баз даних. Наприклад, реалізація типу даних BIT у MySQL фактично ідентична до значення типу даних CHAR(1) у стандарті SQL.

Це створює проблему під час переносу бази даних на іншу платформу. Тим не менш, специфічні для платформи типи даних, як правило, досить близькі до стандарту.

Рядкові дані (послідовності символів)

CHARACTER (n), CHAR (n)   Рядки символів постійної довжини n (N – число символів 0 <N < *8000*)

VARCHAR (n), NVARCHAR (n)   Рядки символів змінної довжини максимум n (+ 2 байта)

Числа точні

INTEGER, INT   Цілі числа (±2,147,483,648, 4 Bytes)

SMALLINT   Малі цілі числа (± 32,768, 2 Bytes)

BIGINT   Великі цілі числа ($\pm 2^{63}-1$, 8 Bytes)

NUMERIC (X, Y), DECIMAL (X, Y)   число, в якому усього X розрядів (точність), з яких Y розрядів (масштаб) відводиться для дробової частини (0 <X < *18*, Y <= X) ($\pm 10^{38}$)

Числа приблизні

FLOAT (n) (n <=53) Дійсне число з плаваючою комою. n – к-ть біт зберігання мантиси при експоненційному поданні ($\pm 1.79*E^{+308}$)

REAL (4 байта)   Дійсне число з плаваючою комою низької точності ($\pm 3.40*E^{+38}$)

DOUBLE PRECISION Дійсне число з плаваючою комою високої точності

Інші типи даних

LONG, TEXT, LONG VARCHAR Неструктуровані дані великого обсягу

BOOLEAN (bit) Логічний тип. Можливі значення True, False і Unknown. Значення Unknown (невідоме) було введено для позначення результату, що виходить при порівнянні зі значенням Null. У SQL–виразах логічні значення записують в лапках, наприклад, `'TRUE'`.

DATE Дата в форматі, визначеному спеціальної командою (за замовчуванням mm/dd/yy). Зберігання – 3 байта (від 01.01.0001 до 31.12.9999, точність 1 день)

TIME Час в форматі, визначеному спеціальної командою (за замовчуванням hh.mm.ss). зберігання – 5 байтів, точність 100 нс.

DATETIME Комбінація дати і часу. зберігання – 8 байтів, точність 0,003 сек. (Від 01.01.1753 00:00:00 до 31.12.9999 23: 59: 59,997)

MONEY Гроші. зберігання – 8 байтів, точність до однієї десятитисячної грошової одиниці

BIT (n) Рядок бітів довжини *n*

XML Дані типу XML.

Рекомендації щодо вибору типів даних

- Тип VARCHAR при фізичному зберіганні займає на два байти більше, ніж тип CHAR при одній і тій же оголошеній довжині.
- Для числових значень фіксованої довжини краще використовувати тип DECIMAL. Він обробляється процесором швидше, ніж тип FLOAT.
- Використовуйте INT і SMALLINT для лічильників.
- Намагайтеся уникати використання LONG VARCHAR без зайвої потреби.
- Уникайте використовувати тип CHAR для представлення числових даних. По-перше, може знадобитися додаткова перевірка, а по-друге, можуть виникнути проблеми при сортуванні таких колонок.
- Використовуйте типи DATE і TIME тільки для зберігання хронологічних даних.
- Використовуйте тип DATETIME виключно для цілей управління даними.

5.3. Команди DDL

5.3.1. Обмеження команд

Обмеження дозволяють автоматично застосовувати правила цілісності даних і фільтрувати дані, які розміщуються в базі даних. У певному сенсі, обмеження – це правила, які визначають, які значення даних є допустимими під час операцій `INSERT`, `UPDATE` та `DELETE`. Коли транзакція модифікації даних порушує правила обмеження, оператор відхиляється або ефект оператора скасовується.

Обмеження можуть бути застосовані на рівні стовпців або на рівні таблиць:

- Обмеження на рівні стовпця – оголошуються як частина визначення стовпця і застосовуються лише до цього стовпця і застосовуються тільки до цього стовпця.
- Обмеження на рівні таблиці – оголошуються незалежно від визначення будь-яких стовпців (традиційно, в кінці інструкції CREATE TABLE) і можуть застосовуватися до одного або кількох стовпців у таблиці або навіть одного чи кількох стовпців в інших таблицях. Обмеження таблиці потрібне, коли ви хочете визначити обмеження, яке застосовується до кількох стовпців.

Типи обмежень:

- Обмеження первинного ключа. Обмеження PRIMARY KEY оголошує один або декілька стовпців, значення яких однозначно ідентифікують кожен запис у таблиці.
- Обмеження на зовнішні ключі. Обмеження FOREIGN KEY визначає один або кілька стовпців у таблиці як такі, що посилаються на первинний у ключі або унікальне значення в іншій таблиці. (Зовнішній ключ може посилатися на унікальний або первинний ключ у тій самій таблиці, але такі зовнішні ключі зустрічаються рідко).
- Обмеження UNIQUE. Обмеження UNIQUE, яке іноді називають ключем-кандидатом або альтернативним ключем, оголошує, що значення в одному стовпчику або комбінація значень у більш ніж одному стовпчику повинні бути унікальними.
- Обмеження CHECK. Обмеження CHECK дозволяють виконувати операції порівняння, щоб переконатися, що значення, які встановлено, відповідають певним умовам. Кілька умов перевірки можна застосувати до стовпця за допомогою операторів AND та OR. Обмеження CHECK вважається виконаним, коли умови перевірки набувають значення True або Null.
- Обмеження NOT NULL – Обов'язковий для заповнення – не дозволяє залишити поле без заповнення.

5.3.2. Команди створення модифікації і видалення таблиць

Після того, як було створено нормалізовану модель бази даних, розробники починають фізичну реалізацію моделі. Перший крок це створення таблиць БД і встановлення зв'язку між ними за допомогою зовнішніх ключів.

Синтаксис команди створення таблиці [9]:

```
CREATE TABLE < table name > (
< column name1 >  < data type >  [ < size > ] [< colconstr > ...],
< column name2 >  < data type >  [ < size > ] [< colconstr > ...],
  [< tabconstr >] ) ;
```

< *table name* > – ім'я таблиці створюваної цією командою

< *column name* > – ім'я стовпця таблиці

< *data type* > – тип даних, який може міститися в стовпці

< *colconstr* > – обмеження для стовпця

< *tabconstr* > – обмеження для таблиці.

Колонки, що є частиною складеного первинного ключа, завжди повинні мати обмеження NOT NULL. Зовнішні ключі повинні також визначатися як NOT NULL, якщо не встановлено правило контролю цілісності SET NULL.

Команду створення таблиць розглянемо на прикладі БД Виробнича бригада (рис. 5.1). Сутність Працівники відображає перелік працівників бригади. Особливість сутності в тому, що в ній зберігається інформація про керівника кожного працівника. Реалізовано це за допомогою рекурсивного зв’язку. Сутність Вироби містить перелік виробів що виготовляються, а сутність Виробництво відображає факти виготовлення виробів.

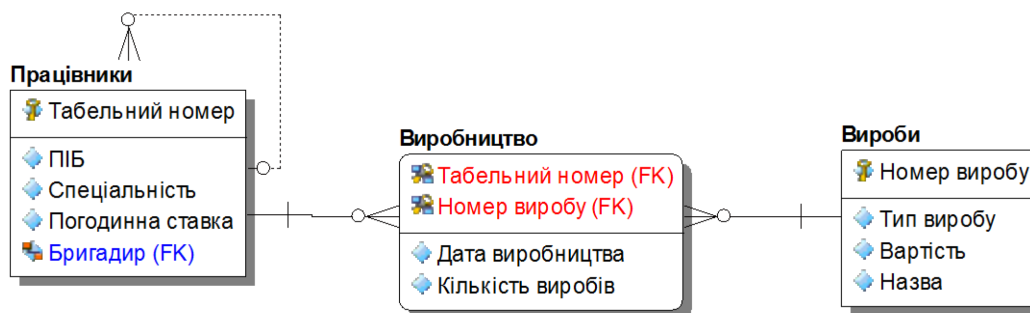


Рис.5.1. ER-діаграма бази даних «Виробнича бригада»

Приклад. Створення таблиць для БД «Виробнича бригада»

Назви відношень і атрибутів замінимо на англійські назви.

```
CREATE TABLE Employees (                   --Працівники
  EmpId INTEGER PRIMARY KEY,               --створення первинного ключа
  EmpName CHARACTER(15) NOT NULL,          --обмеження NOT NULL
  EmpSpec    CHARACTER (15),               --тип даних
  EmpRate DECIMAL (6,1),                   --ставка працівника
  Manager  INTEGER,                        --посилання на керівника
  FOREIGN KEY Manager                      --створення рекурсивного зв'язку
    REFERENCES Employees  (EmpId)          --зовнішній ключ з тієї ж таблиці
    ON UPDATE CASCADE,                     --правила контролю цілісності
    ON DELETE CASCADE
);

CREATE TABLE Items (                              --Вироби
  ItemId INTEGER  PRIMARY KEY,                    --створення первинного ключа
  ItemName CHARACTER (15) NOT NULL,               --обмеження NOT NULL
  ItemType CHARACTER (9) DEFAULT  'чайник' --значення за замовчанням
  /*обмеження значень*/
  CHECK (ItemType IN ('чайник','праска','пилосос','нагрівач')),
  ItemPrice MONEY                                 --вартість
);

CREATE TABLE Operations (                            --Виробництво
  /*обмеження NOT NULL необхідні для створення ключа*/
  EmpId INTEGER NOT NULL,
  ItemId INTEGER NOT NULL,
  StartDate DATE,
  ItemQty  INTEGER,
  PRIMARY KEY (EmpId, ItemId),                        --складовий первинний ключ
  FOREIGN KEY EmpId REFERENCES  Employees    --створення зовнішнього ключа
    ON DELETE CASCADE,
  FOREIGN KEY ItemId REFERENCES  Items       --створення зовнішнього ключа
    ON DELETE CASCADE
```

```
);
```

Реляційна схема створеної БД наведена на рис. 5.2.

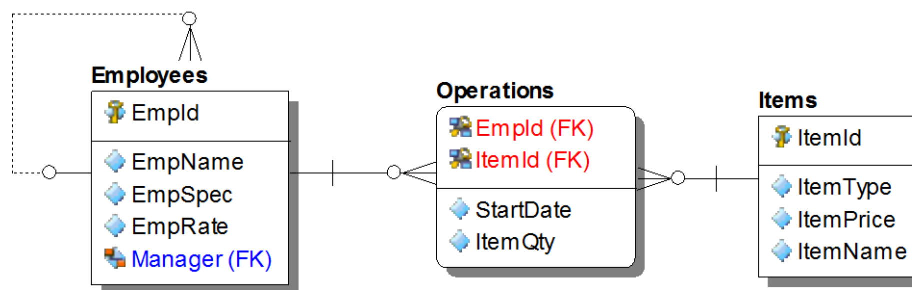


Рис. 5.2. Схема бази даних «Виробнича бригада»

Опції операторів контролю цілісності `ON DELETE`, `ON UPDATE`:

- `CASCADE` – поширити на підлеглі таблиці. Якщо первинний ключ батьківської таблиці змінено або видалено, зовнішні ключі підлеглої таблиці також змінюються або видаляються разом з рядком.
- `SET NULL` – встановити Null значення. Якщо первинний ключ батьківської таблиці видалено, зовнішні ключі приймають значення Null.

5.3.3. Зміна таблиці після того, як вона була створена

Внесення змін в структуру таблиці

Зміни у структурі таблиці здійснюються командою `ALTER TABLE`. За допомогою команди можна додавати нові стовпці, видаляти або змінювати існуючі стовпці, додавати, змінювати і видаляти обмеження. Наприклад,

- додати стовпчик:

```
ALTER TABLE < table name >
ADD < column name > < data type > < size > ;
```

- видалити стовпчик:

```
ALTER < column name > < SET DEFAULT value | DROP DEFAULT  >;
DROP < column name > < RESTRICT | CASCADE} > ;
```

Якщо додається новий стовпець:

- Стовпець буде додано зі значенням Null для всіх рядків таблиці.
- Новий стовпець стане останнім по порядку стовпцем таблиці.
- Можна додати відразу кілька нових стовпців, відокремивши їх комами, в одній команді.

Видалення таблиць

Усі об’єкти бази даних, створені за допомогою інструкцій CREATE, можуть бути знищені за допомогою інструкцій DROP. На деяких платформах оператор ROLLBACK, що виконується після оператора DROP, відновлює знищений об’єкт. Однак на інших платформах баз даних оператор DROP є незворотнім і постійним, тому бажано використовувати команду з обережністю. Синтаксис команди:

```
DROP TABLE < table name >;
```

Щоб мати можливість видалити таблицю, користувач повинен мати права власника таблиці. Перш, ніж видалити таблицю з бази даних SQL може потребувати очистити її від даних.

5.4. Читання даних

Мова запитів SQL складається з однієї команди `SELECT`.

Формат команди SELECT

```
SELECT * | [специфікатор] <список_полів>
FROM  <список_таблиць> [<псевдоніми>]
[ WHERE <предикат - умова вибірки або з'єднання> ]
[ GROUP  BY  <список_полів_групування> |   <номера_колонок>]
[ HAVING <предикат - умова для груп>]
[ ORDER BY <список_полів_впорядкування_виводу> | <номера_колонок>]
```

Пояснення по синтаксису запиту:

`SELECT` – Вибрати дані із зазначених стовпців і (якщо необхідно) виконати перед виведенням їх перетворення відповідно з зазначеними виразами і (або) функціями. *Специфікатор* визначає обмеження на поля виводу і може складатися з наступних елементів:

- `ALL` (за замовчуванням) – виведення рядків, навіть якщо вони дублюються;
- `DISTINCT` – виведення тільки рядків, що не повторюються в обраних полях;
- `TOP` *n* – повертає n записів, що знаходяться на початку результуючого списку рядків;
- `TOP` *n* `PERCENT` – повертає n відсотків записів, що знаходяться на початку результуючого списку рядків (SQL Server);

< *список_полів* > не може бути порожнім. Елементи списку перераховуються через кому і можуть бути:

- – усі поля таблиці;
- <ім'я стовпця> [AS <псевдонім>];
- скалярний вираз – обчислюване значення;
- константа;
- посилання на функцію агрегації (статистичну функцію);
- підзапит.

`FROM` – З перелічених таблиць, в яких розташовані ці стовпці *Список_таблиць* може бути:

- ім'я таблиці;
- список імен таблиць;
- з'єднання двох і більше таблиць;
- підзапит.

`WHERE` – рядки із зазначених таблиць повинні задовольняти зазначеному переліку умов відбору рядків.

```
WHERE <предикат - умова вибірки або з'єднання>
```

У специфікації умови відбору можуть використовуватися будь-які комбінації констант, літералів, функцій, імен полів, а також операторів, що породжують істинне значення:

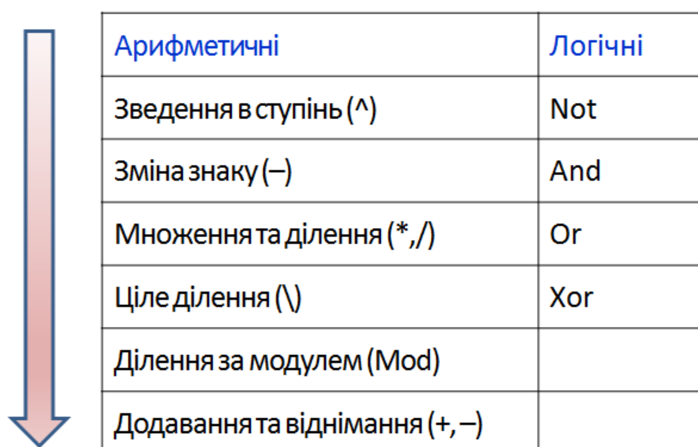
- операції порівняння;
- оператор `Between` <*значення1*> `And` <*значення2*>;
- оператор `Like` <*шаблон*>;
- оператор `In` (<*список значень*>);
- оператор інверсії `NOT` <*вираз*>;
- оператор кон'юнкції <*вираз1*> `AND` <*вираз2*>;
- оператор диз'юнкції <*вираз1*> `OR` <*вираз2*>;
- підзапит.

Речення `WHERE` не є обов'язковим. Однак якщо воно присутнє, то має слідувати після речення `FROM`. Імена полів, які містять пробіли або розділові знаки, необхідно укладати в квадратні дужки [ ].

Якщо вираз містить кілька операторів, то значення компонентів виразу розраховуються в певному порядку. Такий порядок називають порядком *старшинства* або *пріоритетом* операторів. Якщо вираз містить оператори різних типів, то першими виконуються арифметичні операції, слідом за ними – операції порівняння, а останніми – логічні операції.

Усі оператори порівняння мають рівний пріоритет, тобто виконуються в порядку їх розташування у виразі зліва направо. Арифметичні та логічні оператори виконуються в порядку відповідно пріоритету (табл. 5.1).

Таблиця 5.1. Пріоритет виконання операцій в запиті в порядку зменшення



Арифметичні	Логічні
Зведення в ступінь (^)	Not
Зміна знаку (–)	And
Множення та ділення (*,/)	Or
Ціле ділення (\)	Xor
Ділення за модулем (Mod)	
Додавання та віднімання (+, –)	

5.4.1. Функції і оператори

В умовах фільтрації вибірки `WHERE` або відбору після групування даних `HAVING` для формування логічних виразів можна використовувати функції і оператори [12].

- Арифметичні оператори: + – * / %
- Оператори порівняння: > < = >= <= <> !=

- Логічні оператори: AND, OR, ANY, ALL, NOT, BETWEEN, EXISTS, IN, LIKE, IS NULL
- Строкові оператори: + ||
- Функції: Sin(), Trim() та інші.

Оператор BETWEEN

Спеціальний оператор BETWEEN спрощує вирази, що зазвичай вимагають використання операторів OR або AND. Оператор BETWEEN дозволяє скоротити вираз AND з операторами більше або дорівнює (>=) і менше або дорівнює (<=) у вираз з одним оператором, для порівняння деякої величини з діапазоном значень.

Оператор BETWEEN завжди вимагає відповідного AND, розміщеного між двома значеннями. Межі діапазону можна задавати даними типу число, рядок або дата. Якщо значення збігається з межею діапазону, воно потрапляє у вибірку.

Оператор NOT можна використовувати разом з оператором BETWEEN.

Оператор IN

Подібно до того, як BETWEEN є окремим випадком оператора AND, оператор IN дозволяє використовувати окремий випадок оператора OR. Якщо значення що перевіряється, належить множині, зазначеної в дужках оператора IN, то умова відбору WHERE вважається істиною.

Треба сказати, що множину значень можна задати перерахуванням, а можна вкладеним у дужки запитом. Це загальне правило для всіх операторів і функцій, аргументом яких є множина значень. Як і у випадку з оператором BETWEEN, з оператором IN можна використовувати оператор NOT.

Оператор LIKE

Відбір даних на основі неточних збігів у рядкових змінних часто називають зіставленням за зразком. У мові SQL оператор LIKE використовується в реченні WHERE, щоб знайти збіги з частиною значення стовпця. Оператор LIKE вимагає використання спеціальних символів підстановки, щоб точно вказати, як саме має працювати збіг. Наразі стандарт SQL підтримує два метасимволи шаблону:

- _ (Підкреслення) – заміщає рівно один невизначений символ.
- % (Відсоток) – заміщає довільне число символів, починаючи з нуля.

Розробники СКБД пропонують розширені набори метасимволів. Наприклад, в MS SQL Server можна використовувати метасимволи [] і [^]:

- [] – заміщає один символ з діапазону: LIKE '[Л-С] омов ';
- [^] – заміщає один символ НЕ з діапазону: LIKE 'Ів[^ а]% '.

Оператор LKE застосовується тільки до даних текстового типу.

Оператор IS NULL

IS NULL – оператор порівняння з невизначеним значенням (невідомим на даний момент часу). При порівнянні невизначених значень не діють стандартні правила

порівняння: одне невизначене значення ніколи не вважається рівним іншому невизначеному значенню

$$Null_1 \neq Null_2 .$$

Результат арифметичного виразу (що включає, наприклад, +, –, ∗ або ⁄) є Null, якщо будь-яке з вхідних значень є Null. Наприклад, якщо запит містить вираз r.A + 5, а r.A дорівнює Null для певного кортежу, то результат виразу також має бути Null для цього кортежу.

Порівняння з Null-значеннями є більш проблематичними. Наприклад, розглянемо порівняння «`1 < NULL`». Було б неправильно стверджувати, що воно є істинним, оскільки ми не знаємо, що являє собою Null-значення. Але також було б неправильно стверджувати, що цей вираз є хибним (False); якщо ми це зробимо, то «`NOT (1 < NULL)`» буде оцінено як True, що не має сенсу.

Тому SQL розглядає як невідомий (Unknown) результат будь-якого порівняння з Null-значенням (за винятком предикатів `IS NULL` і `IS NOT NULL`).

Це створює третє логічне значення на додаток до істинного і хибного. Оскільки предикат у реченні де може включати булеві операції, такі як І, АБО, і НІ результати порівнянь, визначення булевих операцій розширено в SQL-3, щоб впоратися з невідомим значенням.

В умовах існування невизначених значень Null результати логічних перетворень відповідають правилам тризначній логіки (табл. 5.2).

Таблиця 5.2. **Правила тризначної логіки**

A	B	A and B	A or B	Not A
True	True	True	True	False
True	False	False	True	False
True	Unknown	Unknown	True	False
False	True	False	True	True
False	False	False	False	True
False	Unknown	False	Unknown	True
Unknown	True	Unknown	True	Unknown
Unknown	False	False	Unknown	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

Якщо предикат `WHERE` має значення False або Unknown для кортежу, цей кортеж не буде додано до результату. Тому в запиті необхідно контролювати наявність Null. Це можна зробити за допомогою операторів `IS NULL` або `IS NOT NULL`. Також в запитах можна використовувати функцію `ISNULL(`*`вираз, значення`*`)` яка перевіряє *значення* і якщо це Null, повертає *значення*, інакше повертає *вираз*.

Приклад. Працівники, для яких відділ не визначено:

```
SELECT EmpName, City
FROM Employees
WHERE DeptName IS NULL
```

Звичайно, умови фільтрації можна комбінувати за допомогою дужок і логічних операторів AND, OR.

Приклади використання операторів і функцій розглянемо використовуючи таблицю Працівники (рис. 5.3).

Empid	EpName	Gender	Position	DOB	City	Salary	Deptname
101	Артем	M	бухгалтер	1976-01-11	Київ	8000	Бухгалтерія
102	Віктор	M	електромонтер	1972-04-13	Київ	8000	ВГМ
103	Дмитро	M	токар	1981-01-10	Дніпро	9000	Цех#1
104	Олена	F	менеджер	1986-04-11	Харків	7000	NULL
105	Віктор	M	слюсар	1989-05-05	Дніпро	7000	ВГМ
106	Олена	F	електромеханік	1996-05-11	Львів	5000	ВГМ
107	Сергій	M	токар	1966-01-11	Запоріжжя	7000	Цех#1
108	Ганна	F	програміст	1999-01-11	Харків	4000	IT
109	Сергій	M	адміністратор	1989-05-12	Запоріжжя	8000	IT
110	Дмитро	M	Токар	1981-01-10	Одеса	8500	Цех#1
111	Ганна	F	менеджер	1989-05-26	Одеса	8700	Бухгалтерія

Рис. 5.3. Таблиця Працівники

- Список підрозділів (рис 5.4 a):

```
SELECT DISTINCT Dept FROM Employees;
```

- Якщо треба вибрати всі атрибути таблиць, можна вказати * (зірочка). Привести всі дані про працівників цеху №1 (рис 5.4 b):

```
SELECT * FROM  Employees
WHERE Deptname  = 'Цех#1';
```

- Працівники, дата народження яких більше визначеної дати (рис 5.4 c):

```
SELECT EmpName
FROM Employees
WHERE DOB > '1993-01-10';
```

- Працівники, зарплата яких знаходиться в визначених межах (рис 5.4 d):

```
SELECT EmpName, Salary
FROM Employees
WHERE Salary BETWEEN 8000 AND 9000;
```

a)

Бухгалтерія
Цех#1
ВГМ
Цех#1
IT

b)

103	Дмитро	F	токар	1981-01-10	Дніпро	9000	Цех#1
107	Сергій	M	токар	1966-01-11	Запоріжжя	7000	Цех#1

c)

Олена
Сергій
Ганна

d)

Артем	8000
Віктор	8000
Дмитро	9000
Сергій	8000
Дмитро	8500
Ганна	8700

Рис. 5.4. Результати запитів

- Працівники, з вказаного переліку місць та їхня зарплата (рис 5.5 a):

```
SELECT EmpName, Salary
FROM Employees
WHERE City IN ('Харків','Одеса');
```

- Працівники, посада яких містить рядок «елек» (рис 5.5 b):

```
SELECT EmpName, Position
FROM Employees
WHERE City LIKE '%елек%';
```

- Працівники, для яких відділ не визначено і зарплата менше вказаної (рис 5.5 c):

```
SELECT EmpName, City, Salary
FROM Employees
WHERE DeptName IS NOT NULL AND Salary < 8000;
```

- Оператори і функції в полях виведення. Розрахувати, скільки заробляє за день фахівець відділу Бухгалтерія (25 робочих днів) і розмір податкового податку (15%) (рис 5.5 d):

```
SELECT Empname AS Name, Salary/25 AS Daily, Salary*0.15 AS Tax
FROM Employees
WHERE Deptname = 'Бухгалтерія';
```

Зверніть увагу, як задаються псевдоніми стовпців. Ключове слово AS необов'язкове. Якщо псевдонім складається з одного слова, його можна не брати в лапки. Якщо псевдонім містить пробіли, його треба брати або лапки або в квадратні дужки.

a)

Олена	7000
Сергій	7000
Дмитро	8500
Ганна	8700

b)

Віктор	електромонтер
Олена	електромеханік

c)

Віктор	Дніпро	7000
Олена	Львів	5000
Сергій	Запоріжжя	7000
Ганна	Харків	4000

d)

Name	Daily	Tax
Артем	320	1200
Ганна	348	1305

Рис. 5.5. Результати запитів

Оператор ALL

Оператор ALL виконує перевірку відповідності всієї множини значень умові перевірки. Множина значень як правило формується за допомогою підзапиту.

У разі використання оператора ALL предикат є істинним, якщо кожне зі значень, вибраних у підзапиту, задовольняє умові порівняння.

Оператор > ALL задовольняє умові, заданій у підзапиту, якщо значення в стовпці, для якого вводиться підзапит, повинно бути більше будь-якого значення зі списку, що повертається у підзапиту. ALL(1,2,3) означає «більше 3».

Оператор <>ALL означає те ж, що і NOT IN.

Приклад. Знайти співробітника, зарплата якого більше зарплати будь-якого киянина (рис 5.6 a):

```
SELECT EmpName, City, Salary
FROM Employees
WHERE Salary > ALL (
    SELECT Salary FROM Employees
    WHERE City = 'Київ');
```

Оператор > = ALL зручно використовувати для пошуку максимального значення.

Приклад. Знайти наймолодшого співробітника (рис 5.6 b):

```
SELECT EmpName, DOB
FROM Employees
WHERE DOB > = ALL (SELECT DOB FROM Employees);
```

Оператор ANY

Оператор `ANY` (в деяких СКБД використовується оператор `SOME`) виконує перевірку відповідності умові перевірки будь якого значення з множини значень.

Множина значень як правило формується за допомогою підзапиту. Оператор `ANY` повертає значення True, якщо підзапит повертає хоча б один рядок, що задовольняє оператору порівняння.

Оператор `> ANY` задовольняє умові, заданій у підзапиту, якщо значення в стовпці, для якого вводиться підзапит, повинно бути більше хоча б одного значення зі списку, що повертається у підзапиту. Так `>ANY (1,2,3)` означає «більше 1».

Оператор `=ANY` еквівалентний оператору `IN`. Однак оператор `<> ANY` відрізняється від `NOT IN`:

- `< > ANY` означає «не дорівнює a, або не дорівнює b, або не дорівнює c ».
- `NOT IN` означає «не дорівнює a, і не дорівнює b, і не дорівнює c ».

Приклад. Знайти співробітників-жінок, місце проживання яких збігається з будь яким містом проживання співробітників відділу IT (рис 5.6 c):

```
SELECT EmpName, City
FROM Employees
WHERE City = ANY (
  SELECT City FROM Employees
  WHERE Deptname = 'BK');
```

EpName	City	Salary
Дмитро	Дніпро	9000
Дмитро	Одеса	8500
Ганна	Одеса	8700

a)

Ганна	Харків

b)

Олена	Харків
Сергій	Запоріжжя
Ганна	Харків
Сергій	Запоріжжя

c)

Рис. 5.6. Результати запитів

Оператор `EXIST`

Оператор `EXISTS (`*`підзапит`*`)` повертає True, якщо підзапит повертає хоча б один запис. Детально розглянемо цей оператор далі.

5.4.2. Агрегатні функції

Досі всі обчислення і функції, які ми використовували, використовували окремі рядкам у таблицях бази даних. Тепер перейдемо до різних методів підсумовування даних шляхом об’єднання значень у кількох рядках. Здатність агрегувати та підсумовувати дані є ключовим фактором для того, щоб вийти за межі простого відображення даних і наблизитись до чогось, що є змістовною інформацією.

Агрегатні функції оперують набором значень і повертають єдине підсумкове значення. Агрегатні функції також часто комбінують з операторами `GROUP BY` і `HAVING`, які визначають кілька груп даних [10].

Стандарт SQL визначає багато агрегатних функцій, розглянемо найбільш популярні: `COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()`.

Приклади використання агрегатних функцій розглянемо з використанням таблиці на рис.5.3.

Функція COUNT

Функція `COUNT(ім'я_стовпця)` підраховує число значень в заданому стовпці, відмінних від Null або загальне число рядків в таблиці. Можна використовувати ключове слово `DISTINCT` для підрахунку різних числових значень в даному стовпці. Тип даних в стовпці може бути будь-яким, функція `COUNT` завжди повертає <u>цілий</u> тип даних незалежно від типу даних стовпців.

Щоб підрахувати загальну кількість рядків в таблиці, треба використовувати функцію `COUNT` із зірочкою замість імені поля `COUNT (*)`.

Приклад. Загальна кількість рядків таблиці (рис 5.7 a):

```
SELECT COUNT(*) Всього        --11
FROM Employees;
```

Приклад. Кількість рядків таблиці, не враховуючи значення Null (рис 5.7 b)::

```
SELECT COUNT(EmpName ) [В гуртожитку]
FROM Employees                    --8
WHERE City <> 'Київ';
```

Приклад. Скільки співробітників зі іменами що збігаються:

```
SELECT COUNT(Name) - COUNT(DISTINCT Name) --5
FROM Employees;
```

Функції MAX і MIN

Функції `MAX()` і `MIN()` оперують одним стовпцем таблиці. Вони вибирають максимальне або мінімальне значення відповідно з цього стовпчика.

Стовпець може містити числові або строкові значення, або значення дати / часу. Результат, що повертається цими функціями, має точно такий же тип даних, що і сам стовпець.

Приклад. Знайти максимальну і мінімальну зарплату працівників (рис 5.7 c):

```
SELECT MAX(Salary) [Макс. з/п], MIN(Salary) [Мін. з/п]
FROM Employees;
```

Функції SUM і AVG

Функція `SUM()` обчислює суму всіх значень стовпця, функція `AVG()` обчислює середнє значення. Дані, що містяться в стовпці, повинні мати числовий тип (містити цілі числа, десяткові числа або числа з плаваючою комою або грошові величини). Результат, що повертається функціями `SUM()` і `AVG()`, має той же тип даних, що і стовпець.

Особливо зручно використовувати псевдоніми для обчислювальних стовпців, для яких спочатку ім’я не передбачено.

Приклад.. Підрахувати фонд заробітної плати відділу головного механіка і середню зарплату працівників (рис 5.7 d):

```
SELECT SUM(salary) [Фонд З/П ВГМ], AVG(salary) [Середня З/П ВГМ],
 FROM Employees
WHERE DeptName = 'ВГМ';
```

Всього
11

a)

В гуртожитку
8

b)

Макс. з/п	Мін. з/п
9000	4000

c)

Фонд З/П ВГМ	Середня З/П ВГМ
20000	6667

d)

Рис. 5.7. Результати запитів

Оператор GROUP BY

Для цілей аналізу даних, однорідні рядки реляційної таблиці можуть бути об'єднані в групи. Наприклад, дані на рис. 5.3 можна аналізувати в розрізі підрозділів або міст. Речення GROUP BY об'єднує в одну групу записи з однаковими значеннями в зазначених стовпцях.

Речення GROUP BY використовується на практиці, коли потрібна статистична інформація не по окремому об'єкту, а про кожну групу. Групуючи по зазначеному переліку стовпців, можна отримати для кожної групи єдине агреговане значення, використовуючи у реченні SELECT SQL-функції SUM(), COUNT(), MIN(), MAX() або AVG(). Замість імен стовпців можна використовувати їх номери – GROUP BY 1,2.

Імена обраних в команді SELECT стовпців повинні бути присутніми в реченні GROUP BY за винятком тих, до яких застосовані агрегатні функції. Порядок групування визначається порядком стовпців у списку GROUP BY (спочатку за першим стовпцем, потім за другим і т. д.);

Приклад. Підрахувати середню, максимальну і загальну суму зарплати по відділах (рис 5.8 а):

```
SELECT Deptname Підрозділ, AVG(ISNULL(salary, 0)) Середня,
SUM(Salary) Сумарна, MAX(Salary) Максимальна
FROM Employees
GROUP BY Deptname
```

Зверніть увагу, як обчислюється середнє значення. Використання функції ISNULL(salary,0) дозволяє обчислювати середнє значення з урахуванням всіх рядків таблиці. Якщо стовпчик групування має значення Null, усі такі значення вважаються ідентичними й об'єднуються в одну групу (як і під час виключення повторюваних рядків DISTINCT).

Оператор HAVING

Якщо для відбору рядків, наприклад тих, що підлягають групуванню, використовується речення WHERE, то для відбору груп застосовується речення HAVING.

Речення HAVING визначає, які згруповані записи відображаються під час використання інструкції SELECT із реченням GROUP BY. В результаті залишаться лише ті групи, які задовольняють зазначеної умові відбору груп.

Речення HAVING у загальному випадку має такий вигляд:

```
HAVING <умова відбору груп>
```

Як умову відбору груп можуть використовуватися:

- операції порівняння;
- оператори BETWEEN, LIKE, IN;
- оператор інверсії NOT <*вираз*>;
- підзапит.

Приклад. Знайти співробітників, що мають однакову дату народження. Підзапит повертає перелік дат народження, що повторюються (рис 5.8 b):

```
SELECT Name, DOB
FROM Employees
WHERE DOB IN (
     SELECT DOB FROM Employees
     GROUP BY DOB,
     HAVING COUNT(DOB)> 1);
```

Підрозділ	Середня	Сумарна	Максимальна
Бухгалтерія	8350	16700	8700
ВГМ	6667	20000	8000
IT	6000	12000	8000
Цех#1	8000	16000	9000
NULL	7000	7000	7000

a)

Дмитро	1981-01-10
Дмитро	1981-01-10

b)

Рис. 5.8. Результати запитів

5.4.3. Багатотабличні запити

Для нормалізованих таблиць бази даних обов’язковою є наявність пов’язаних таблиць. І, як правило, запити збирають інформацію з кількох пов’язаних таблиць [9, 15, 16].

Зв’язок між таблицями визначається первинним ключем в батьківській таблиці і зовнішнім ключем в пов’язаній таблиці. Цей факт використовується для формування запитів до кількох таблиць.

Найпростіший спосіб зазначення вибірки з декількох таблиць – перерахувати список таблиць після ключового слова FROM (FROM t1, t2). Такий спосіб має назву неявне з’єднання. У цьому випадку виконується декартів добуток таблиць (на практиці це не так) з подальшою фільтрацією за умовою з’єднання в реченні WHERE.

Для прикладу, розглянемо таблиці, створеної раніше бази даних Виробнича бригада (рис. 5.2).

Приклад. Робітники яких спеціальностей виробляли виріб типу кавоварка:

```
SELECT EmpSpec
FROM Employees, Operations, Items
WHERE Employees.EmpId = Operations.EmpId
AND Operations.ItemId = Items.ItemId
AND ItemType  = 'кавоварка';
```

Умова з'єднання – рівність значення первинного ключа в таблиці Employees і значення зовнішнього ключа в таблиці Operations.

Цей спосіб цілком робочий і раніше використовувався, але зараз вважається застарілим і не рекомендованим для використання. Стандарт SQL рекомендує інші варіанти вибірки з кількох таблиць – шляхом явного зазначення з'єднання.

Види з'єднань таблиць

- INNER JOIN – внутрішнє з'єднання;
- LEFT [OUTER] JOIN – ліве зовнішнє з'єднання;
- RIGHT [OUTER] JOIN – праве зовнішнє з'єднання;
- FULL [OUTER] JOIN – повне зовнішнє з'єднання;
- CROSS JOIN – перехресне з'єднання (декартов добуток).

Значення вказаних видів з'єднання розглядалось в розділі 2.

INNER JOIN

INNER JOIN – це тип з'єднання за замовчуванням (в запиті можна вказати тільки JOIN), який використовується для відбору даних зі збіжними значеннями в обох таблицях. Синтаксис використання внутрішнього з'єднання:

```
SELECT [Column List]
FROM [Table1] INNER JOIN [Table2]
ON [Table1.ColumnName] = [Table2.ColumnName]
WHERE [Condition];
```

Приклад. Робітники яких спеціальностей виробляли виріб типу кавоварка:

```
SELECT EmpSpec
FROM Employees
INNER JOIN Operations ON Employees.EmpId = Operations.EmpId
INNER JOIN Items ON Operations.ItemId= Items.ItemId
WHERE ItemType  = 'кавоварка';
```

Порядок таблиць що з'єднуються немає значення.

LEFT OUTER JOIN

На відміну від внутрішніх з'єднань, існує три типи зовнішніх з'єднань: ліве зовнішнє з'єднання (LEFT OUTER JOIN), праве зовнішнє з'єднання (RIGHT OUTER JOIN) і повне зовнішнє з'єднання (FULL OUTER JOIN).

Їх можна зазначити просто LEFT JOIN, RIGHT JOIN і FULL JOIN. У цьому випадку слово OUTER не є обов'язковим

У загальному випадку, при використанні внутрішньому з'єднанні, деякі кортежі в одному або обох відношеннях, що з'єднуються, можуть бути «втрачені». Операція зовнішнього з'єднання працює подібно до операції JOIN, але вона зберігає ті кортежі, які було б втрачено при внутрішньому з'єднанні.

Відмінності між різними видами з'єднання можна показати на рис. 5.9.

Синтаксис використання лівого зовнішнього з'єднання:

```
SELECT [Column List]
FROM [Table1] LEFT OUTER JOIN [Table2]
ON [Table1.ColumnName] = [Table2.ColumnName]
WHERE [Condition];
```

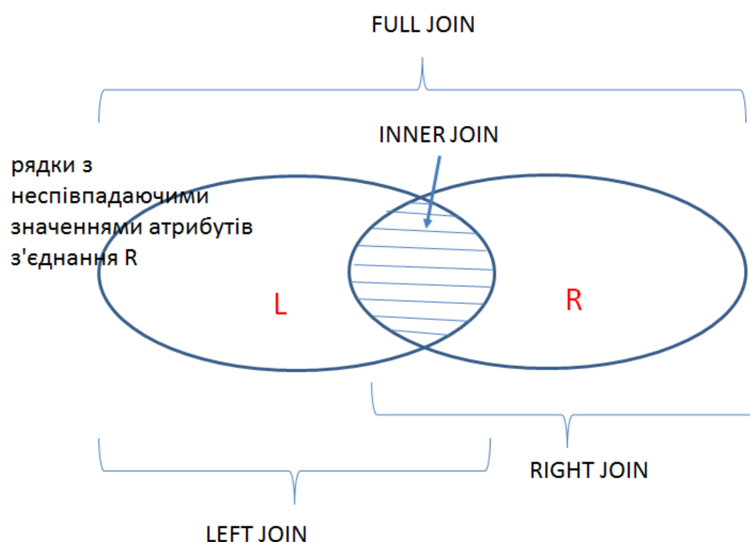


Рис. 5.9. Відмінності між різними видами з’єднання

Можемо обчислити ліве зовнішнє з’єднання наступним чином: спочатку виконується внутрішнє з’єднання, потім для кожного кортежу у лівому відношенні, який не збігається з жодним кортежем у відношенні зліва, додається кортеж до результату з’єднання, побудованого наступним чином:

- Атрибути, що стосується лівому кортежу додаються в результат,
- Решта атрибутів результату заповнюються Null значеннями.

Таким чином, в випадку лівого зовнішнього з’єднання в результуючу вибірку потраплять всі рядки лівої таблиці.

Зараз продемонструємо різницю між `INNER JOIN` та `LEFT JOIN`.

Приклад. Фрагмент бази даних наведено на рис. 5.10. Як підрахувати суму продажів по кожному клієнту. Нас цікавлять і ті клієнти, по яким не було продажів.

Customer

IdCust	Name
1	Згода
2	Злагода
3	Зоря

Sale

Id	Date	Price
1	02.03.19	7000
2	02.03.19	9000
2	04.03.19	15000
2	05.03.19	3000
1	05.03.19	1000
1	07.03.19	2000

Рис. 5.10. База даних Продажі клієнтам

Сформуємо два запити з різними видами з’єднання (рис. 5.11):

```
SELECT Назва, SUM(Продажі)
FROM Клієнт
INNER JOIN Продажі ON
Клієнт.Номер = Продажі.Номер
GROUP BY Назва;
```

a

```
SELECT Назва, SUM(Продажі)
FROM Клієнт
LEFT JOIN Продажі ON
Клієнт.Номер = Продажі.Номер
GROUP BY Назва;
```

b

Рис. 5.11. Текст запитів

І ось які результати отримуємо (рис. 5.12):

Назва	Сума
Згода	10000
Злагода	27000

a

Назва	Сума
Згода	10000
Злагода	27000
Зоря	NULL

b

Рис. 5.12. Результати запитів

Як бачимо, в випадку з'єднання `INNER JOIN` сталася втрата інформації стосовно клієнтів. Тому, при створенні запитів необхідно добре уявляти призначення того чи іншого виду з'єднання.

Приклад. Список працівників, що виготовляли праски:

```
SELECT DISTINCT EmpName
FROM Employees  E
INNER JOIN Operations  O ON  E.EmpId = O.EmpId
INNER JOIN Items  I ON  O.ItemId = I.ItemId
WHERE ItemType = 'праска';
```

Приклад. Підрахувати вартість виготовлених виробів по кожному працівнику:

```
SELECT EmpName, SUM(ItemQty * Price)
FROM Employees  E
LEFT JOIN Operations  O ON  E.EmpId = O.EmpId
LEFT JOIN Items  I ON  O.ItemId = I.ItemId
GROUP BY EmpID, EmpName;
```

RIGHT OUTER JOIN

Синтаксис використання правого зовнішнього з'єднання:

```
SELECT [Column List]
FROM [Table1] RIGHT OUTER JOIN [Table2]
ON [Table1.ColumnName] = [Table2.ColumnName]
WHERE [Condition];
```

В випадку правого зовнішнього з'єднання в результуючу вибірку потрапляють всі рядки правої таблиці.

FULL OUTER JOIN

Синтаксис використання повного зовнішнього з'єднання:

```
SELECT [ColumnList]
FROM [Table1] FULL OUTER JOIN [Table2]
ON [Table1.ColumnName] = [Table2.ColumnName]
WHERE [Condition];
```

Повне зовнішнє з’єднання можна розглядати як комбінацію лівого і правого зовнішніх з’єднань. В випадку повного зовнішнього з’єднання в результуючу вибірку потрапляють всі рядки обох таблиць.

Повне з’єднання рідко використовується на практиці з тієї простої причини, що цей тип зв’язку між таблицями є відносно рідкісним. По суті, повне з’єднання показує дані, в яких є невідповідності в обох напрямках між двома таблицями. Зазвичай цікавлять лише дані, в яких є повний збіг між двома таблицями (внутрішнє з’єднання) або, можливо, односторонній збіг (ліве або праве з’єднання).

Оператор `CROSS JOIN` (перехресне з’єднання) визначає декартів добуток двох відношень. По суті, перехресне з’єднання – це спосіб з’єднання двох таблиць без вказівки зв’язку між ними. Оскільки зв’язок не вказується, перехресне з’єднання створює будь-яку комбінацію рядків між таблицями. Якщо одна таблиця має три рядки, а друга – чотири, і ці таблиці з’єднані перехресним з’єднанням, то в результаті вийде 12 рядків.

Самоз’єднання

Самоз’єднання в SQL – це спеціалізована техніка, яка використовується для з’єднання таблиці з самою собою, що дозволяє отримувати ієрархічні або рекурсивні зв’язки в межах однієї і тієї ж сутності. Поширені сценарії включають використання самоз’єднання:

- Ієрархічні дані: Відображають зв’язки «батько-дитина» в організаційних структурах, наприклад, зв’язки «працівник-керівник».
- Мережеві дані: Моделювання відносин у соціальних мережах, де користувачі пов’язані з іншими користувачами.

Найпоширенішим застосуванням самоз’єднання є робота з таблицями, що посилаються на себе. Такі таблиці містять стовпець, який посилається на інший стовпець в тій же таблиці. Поширеним прикладом такого типу зв’язку є таблиця, що містить інформацію про працівників. В створеній базі даних (рис. 5.2) кожен рядок таблиці «Employees» має стовпець «Manager», який вказує на інший рядок тієї ж таблиці, що представляє керівника працівника (рис. 5.13).

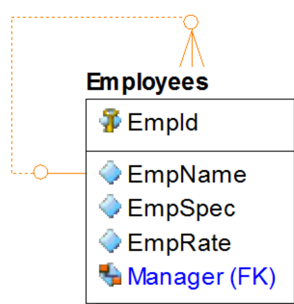


Рис. 5.13. Схема таблиці Працівники

Атрибут «Manager» є зовнішнім ключем.

Приклад. Список працівників із зазначенням імен їх керівників:

```
SELECT Employees.EmpName 'Працівник', Managers.EmpName 'Керівник'
FROM Employees AS  Emp1
```

```sql
INNER JOIN Employees  AS Managers ON  Managers.EmpId = Emp1. Manager
ORDER BY Emp1.EmpName;
```

Всі вказані види з’єднань можна використовувати для з’єднання таблиці із самою собою. В розглянутому прикладі може статися втрата інформації, оскільки для головного керівника установи атрибут Manager має значення Null. Для виправлення помилки слід використовувати зовнішнє з’єднання:

```sql
SELECT Employees.EmpName 'Працівник', Managers.EmpName 'Керівник'
FROM Employees AS  Emp1
LEFT JOIN Employees  AS Managers ON  Managers.EmpId = Emp1. Manager
ORDER BY Emp1.EmpName;
```

Рекомендації щодо з’єднань

- Завжди використовуйте псевдоніми для таблиць. У деяких незначних випадках можна виконати з’єднання без псевдонімів ваших таблиць, але щойно ви почнете вибирати окремі стовпці, все почне заплутуватися. Завжди корисно надавати таблицям псевдоніми.
- Яка таблиця буде першою? Очевидно, що є вибір, яку таблицю писати першою. На практиці, однак, розробник ймовірно, більше зацікавлений в одній таблиці, ніж в іншій. Це означає, що коли мова йде про зовнішні з’єднання, швидше за все, будете використовуватися ліве з’єднання. Це те, що використовується здебільшого.

Наприклад, СКБД SQLite навіть не підтримує `RIGHT JOIN`, і необхідно міняти місцями таблиці, щоб використовувати `LEFT JOIN`.

5.4.4. Вкладені запити

Мова SQL дозволяє вкладати один запит в інший. Зазвичай, є головний (зовнішній) запит і один або декілька підзапитів, вкладених у зовнішній запит. Вкладений запит (підзапит) генерує набір значень, який перевіряється операторами `WHERE` і `HAVING` зовнішнього запиту [12].

Підзапити застосовуються в різних практичних сценаріях:

- Фільтрація: Фільтрація даних на основі результатів іншого запиту, наприклад, пошук товарів, продажі яких перевищують певний поріг.
- Агрегація: Виконання агрегатних функцій (наприклад, `SUM`, `AVG`) над результатами підзапитів для обчислення зведеної статистики.
- Кореляція: Співвіднесення даних між зовнішнім і внутрішнім запитами за допомогою корельованих підзапитів для оновлення або видалення записів на основі умов.

Підзапити самі можуть містити інші підзапити. Рівень вкладання у стандарті SQL не обмежується, але у деяких СКБД такі обмеження є. Наприклад, поточна версія Oracle підтримує до 255 рівнів вкладеності всередині `WHERE`.

Підзапити можна використовувати в операторах `SELECT`, `INSERT`, `UPDATE` і `DELETE` для отримання або маніпулювання даними на основі умов, що обчислюються в підзапиті.

Ось загальний вигляд команди `SELECT`, який ми розглядали раніше:

```
SELECT columnlist
FROM tablelist
WHERE condition
GROUP BY columnlist
HAVING condition
ORDER BY columnlist;
```

Підзапити можна вставляти практично в будь-яке речення в інструкції SELECT. Однак спосіб створення та використання підзапиту дещо відрізняється залежно від того, де він використовується – у списку таблиць, умові вибору чи у списку стовпців. Підзапити можна вказати трьома різними способами:

- Коли підзапит є частиною списку таблиць (*tablelist*), він вказує джерело даних. Це стосується ситуацій, коли підзапит є частиною речення FROM.
- Коли підзапит є частиною умови (*condition*), він стає частиною критеріїв відбору. Це стосується ситуацій, коли підзапит є частиною речення WHERE або HAVING.
- Коли підзапит є частиною списку стовпців (columnlist), він створює один обчислювальний стовпець. Це стосується ситуацій, коли підзапит є частиною речення SELECT, GROUP BY або ORDER BY.

За типом результату що повертається підзапити бувають:

- скалярні – повертають завжди тільки одне значення. Для перевірки результатів в зовнішньому запиті використовуються оператори порівняння: (>, =,> =, <,! =, <=). Такі запити можна використовувати в реченнях SELECT, WHERE або HAVING;
- рядкові підзапити – повертають один рядок даних. Для перевірки результатів в зовнішньому запиті використовуються оператори: ((NOT) IN, (NOT) EXISTS;
- табличні – повертають набір рядків і стовпців і можуть бути використаний в операторах FROM або JOIN.

За способом отримання результату розрізняють підзапити:

- прості – не залежать від зовнішнього запиту і можуть оброблятися окремо;
- корельовані – залежать від зовнішнього запиту в контексті виконання, часто посилаючись на стовпці зовнішнього запиту в реченнях WHERE або HAVING.

До підзапитів застосовуються такі правила оформлення:

- підзапити повинні міститися в круглих дужках;
- у підзапиту не можна використовувати речення ORDER BY;
- якщо підзапит є операндом операції порівняння, то підзапит має вказуватися в правій частині цієї операції.

5.4.5. Прості вкладені запити

Прості підзапити формально не пов’язані із зовнішнім запитом. Виконання підзапиту і основного запиту можна розділити на два незалежні етапи, тобто спочатку виконується підзапит, а потім виконується основний запит.. Підзапит надає результат, який використовується зовнішнім запитом як константа або параметр.

Прості підзапити можуть використовуватися після будь-якого з операторів SELECT, FROM, WHERE і HAVING.

Підзапити в списку виводу SELECT

Підзапит в списку виводу SELECT повинен повертати одне значення (скалярний підзапит).

Приклад. Для БД Виробнича бригада визначити погодинну ставку кожного працівника разом її з середньою ставкою по підприємству:

```
SELECT EmpName, EmpRate,
 (SELECT AVG(EmpRate) FROM Employees) AS AvgRate
FROM Employees;
```

Тут слід звернути увагу на кілька речей:

- Під час написання підзапиту в списку виводу SELECT наполегливо рекомендується вказувати псевдонім стовпця. Таким чином, стовпець матиме просту назву в результатах.
- У стовпці AvgRate є лише одне значення, яке повторюється в усіх рядках.

Підзапити в реченні FROM

Коли підзапит вказується як частина речення FROM, він створює нове джерело даних. Підзапит у реченні FROM можна розглядати як різновид віртуальної таблиці.

Далі використовується база даних «Виробнича бригада» (рис. 5.1).

Приклад. Підрахунок кількості унікальних комбінацій з декількох стовпчиків. Скільки різних виробів зробив кожен працівник.

```
SELECT EmpId , COUNT(*) AS Num_unique
FROM (SELECT DISTINCT O.EmpId, O.ItemId
      FROM Operations O) My_subquery
GROUP BY EmpId;
```

Приклад. Робочі яких спеціальностей виготовляли чайники?

```
SELECT EmpSpec
FROM Employees AS Emp INNER JOIN
    (SELECT EmpId
     FROM Operations O
     INNER JOIN Items I ON O.ItemId = I.ItemId
     WHERE ItemType ='чайник' ) AS Obj
ON Emp.EmpId = Obj.EmpId;
```

Підзапити в реченні WHERE

Підзапит в реченні WHERE дозволяє виконувати фільтрацію даних.

Приклад. Вивести список робітників, які мають таку ж спеціальність, що і робочий з табельним номером 2145:

```
SELECT EmpName
FROM Employees
WHERE EmpSpec =
    ( SELECT EmpSpec
    FROM Employees
    WHERE EmpId = 2145 );
```

Оператори порівняння =,>, <,> =, <=,! = можна використовувати, якщо підзапит повертає тільки одне значення при будь-яких наборах даних. Для повернення одного значення можна використовувати агрегатні функції.

Якщо основний запит створює багато рядків, для порівняння з результатами підзапиту треба використовувати оператори ALL, ANY, (NOT) IN, (NOT) EXISTS.

Приклад. Використання оператора IN. Вивести список робітників, які не виготовляли пилососи:

```
SELECT EmpName
FROM Employees
WHERE EmpId NOT IN
    (SELECT DISTINCT EmpId
     FROM Operations  O
     INNER JOIN Items I ON O.ItemId = I.ItemId
     WHERE I.ItemName = 'пилосос');
```

Приклад. Використання оператора ANY. Вивести працівників, у яких погодинна ставка більше, ніж ставка хоча б одного працівника, чий бригадир має табельний номер 2356

```
SELECT EmpName
FROM Employees
Where EmpRate > ANY (
  SELECT EmpRate
  From Employees
  Where Manager = 2356);
```

Приклад. Використання оператора ALL. Список працівників, у яких ставка менше ніж у електриків

```
SELECT EmpName
FROM Employees
WHERE EmpRate < ALL (
     SELECT EmpRate
     FROM Employees|
     WHERE EmpSpec = 'Електрик');
```

Приклад. Використання агрегатних функцій і підзапиту. У кого з працівників погодинна ставка вища за середню?

```
SELECT EmpName
FROM Employees
WHERE EmpRrate > (
     SELECT AVG(EmpRate)
     FROM Employees);
```

Підзапити в реченні HAVING

Підзапити в реченні HAVING дозволяють фільтрувати груповані дані за значенням агрегатних функцій.

Приклад. Вивести назви виробів, для яких кількість зроблених виробів більше кількості зроблених прасок:

```
SELECT ItemName, SUM(ItemQty)
FROM Operations Obj
INNER JOIN Items It ON Obj.ItemId = It.ItemId
GROUP BY ItemName
HAVING SUM(ItemQty) >
  (SELECT SUM(ItemQty)
   FROM Operations  O
```

```
INNER JOIN Items I ON O.ItemId = I.ItemId
WHERE I.ItemName = 'праска');
```

5.4.6. Корельовані вкладені запити

Корельований підзапит означає, що при виконанні запиту спочатку отримується значення властивості зовнішнього запиту, потім виконується підзапит, пов'язаний з цією властивістю, а після завершення виконання отримується наступне значення зовнішнього запиту, і підзапит виконується знов. Корельовані підзапити не можуть виконуватися самостійно, тому що якийсь елемент у запиті робить його залежним від зовнішнього запиту. Для того, щоб підкреслити посилання на зовнішній запит, обов'язково використовуються псевдоніми.

Корельовані підзапити також можуть використовуватися після будь-якого з операторів SELECT, FROM, WHERE і HAVING.

Корельовані підзапити в списку виводу SELECT

Приклад. Вивести список працівників і кількість виготовлених кожним з них виробів:

```
SELECT EmpName,
  (SELECT COUNT(*)
   FROM Operations AS O
   WHERE O.EmpId = Emp.EmpId) AS num_items
FROM Employees AS Emp;
```

Псевдонім Emp використовується для посилання на поточний рядок зовнішнього запиту в підзапиту.

Корельовані підзапити в операторі WHERE

Приклад. Вивести працівників, чиї погодинні ставки більше, ніж погодинні ставки їх бригадирів:

```
SELECT EmpName
FROM Employees AS Emp
WHERE Emp.EmpRate >
  (SELECT Head.EmpRate
   FROM Employees AS Head
   WHERE Head. EmpId = Emp.Manager);
```

Приклад. Задачу вивести список робітників, які не виготовляли пилососи також можна вирішити за допомогою корельованого запиту використовуючи оператор EXISTS:

```
SELECT EmpName
FROM Employees Emp
WHERE NOT EXISTS
      (SELECT *
   FROM Operations  O
   INNER JOIN Items I ON O.ItemId = I.ItemId
   WHERE O.EmpId = Emp. EmpId AND I.ItemType= 'пилосос');
```

Приклад. Вивести працівників, які виготовляли вироби всіх видів:

```
SELECT EmpName
FROM Employees Emp
WHERE NOT EXISTS (
  (SELECT ItemId                -- Множина X - всі вироби
   FROM  Items I) AS X
EXCEPT
  (SELECT DISTINCT ItemId                  -- Множина Y - вироби
```

```
   FROM Operations  O           -- які працівники виробляли
   WHERE O.EmpId =  Emp. EmpId ) AS Y
);
```

В цьому запиті використано властивість відношень – якщо множини збігаються, їх різниця дасть пусту множину. Зверніть увагу, що це завдання не можна записати за допомогою операторів ALL, ANY, IN. Оператор EXCEPT визначає різницю множин.

Оптимізація вкладених запитів

Підзапити є важливими компонентами розробки SQL-запитів, що дозволяють складний аналіз, фільтрацію та маніпулювання даними. Але вкладені запити можуть створювати проблеми з швидкістю виконання запитів. Для оптимізація підзапитів з метою підвищення продуктивності можна використовувати наступні рекомендації:

- Перетворення в з'єднання. Переписування корельованих підзапитів у вигляді операцій з'єднання JOIN для кращої оптимізації та виконання запитів.
- Витіснення предикатів. Переміщення предикатів із зовнішніх запитів у підзапити для мінімізації пошуку даних і підвищення ефективності.
- Обмеження результатів підзапитів. Використання операторів TOP або LIMIT для оптимізації продуктивності запитів і зменшення споживання ресурсів.
- Підзапити краще використовувати, коли потрібно швидко обчислити агрегатну функцію і використовувати її значення для порівняння;
- З'єднання мають перевагу над підзапитами, якщо список вибору оператора SELECT містить стовпці більш ніж з однієї таблиці.

Як правило, запити з підзапитах можуть бути записані у вигляді з'єднання таблиць і навпаки. Є багато ситуацій, в яких JOIN є кращим рішенням. Речення JOIN не містить додаткових запитів. Використання операторів з'єднання робить запит більш зрозумілим і полегшує ядру сервера побудову плану запиту. , Але є й інші ситуації, де краще використовувати підзапит. Іноді є вибір між ними, але бувають випадки в яких підзапит є єдиним реальним варіантом.. Підзапити і JOIN можуть використовуватися в складному запиті для відбору даних з декількох таблиць, але вони роблять це по-різному.

5.4.7. Загальний табличний вираз (Common Table Expression *CTE*)

Common Table Expression (CTE) – це тимчасові результуючі набори (тобто результати виконання SQL запиту), які не зберігаються в базі даних у вигляді об'єктів, але до них можна звертатися. CTE підвищують читаність коду шляхом поділу запиту на логічні блоки, і тим самим спрощують роботу зі складними запитами [10].

Вони призначені:

- для написання рекурсивних запитів;
- для заміни подання (view), наприклад, коли немає необхідності зберігати в базі SQL визначення запиту;
- для визначення кілька тимчасових таблиць;
- для багаторазових посилань на результуючий набір з однієї і тієї ж SQL інструкції.

Існує два типи CTE:

1. Нерекурсивний CTE. Запит, на який посилаються інші запити. Нерекурсивні CTE – це прості CTE, які не посилаються самі на себе і корисні для спрощення SQL–запитів, що передбачають складний пошук даних або маніпуляції з ними. Нерекурсивні CTE підвищують гнучкість і «ремонтопридатність» SQL-запитів, роблячи складну логіку простішою в управлінні і розумінні.
2. Рекурсивний CTE. Запит, який посилається на сам себе. Рекурсивні CTE дозволяють SQL-запитам обробляти ієрархічні структури даних, такі як організаційні схеми, специфікації або мережеві маршрути, де зв'язки між даними мають рекурсивну природу.

CTE визначається за допомогою оператора WITH, і визначити його можна як в звичайних запитах, так і в функціях, збережених процедурах, тригерах і поданнях.

Нерекурсивні CTE

Нерекурсивні CTE зустрічаються набагато частіше, ніж рекурсивні.

Синтаксис нерекурсивного CTE:

```
WITH Cte_name (column1, column2, ...)
AS
(
-- визначення запиту CTE
SELECT column1, column2, ...
FROM table_name
WHERE condition
)
-- головний запит
SELECT column1, column2, ...
FROM Cte_name
WHERE condition;
```

`Cte_name` – це псевдонім або ідентифікатор CTE. Звертатися до CTE можна використовуючи цей псевдонім. `Column1, column2`, ...– перелік стовпців, який буде визначений в CTE. Їх кількість повинна збігатися з кількістю стовпців, що повертаються запитом;

Приклад. Знайти всіх працівників з максимальною ставкою

```
WITH max_budget (value) AS
    (SELECT MAX(EmpRate)
     FROM Employees)
SELECT EmpName
FROM Employees, max_budget
WHERE Employees.EmpRate = max_budget.value;
```

В цьому прикладі max_budget – назва CTE, value – значення що повертає CTE.

Рекурсивні CTE

Синтаксис рекурсивного CTE:

```
WITH RecursiveCTE AS
(
-- визначення кореня ієрархії
SELECT column1, column2, ...
FROM table_name
WHERE condition
UNION ALL
-- рекурсивна частина
SELECT column1, column2, ...
```

```
FROM table_name
INNER JOIN RecursiveCTE ON table_name.foreign_key = RecursiveCTE.primary_key
)
SELECT column1, column2, ...
FROM RecursiveCTE;
```

Приклад. Рекурсивне CTE . Таблиця Employees (рис. 5.14) зберігає інформацію щодо підлеглості працівників (стовпчик Manager). Треба визначити на якому рівні в ієрархії підлеглості знаходиться кожен працівник (рис. 5.15).

EmpId	Speciality	Manager
1	Директор	NULL
2	Головний бухгалтер	1
3	Бухгалтер	2
4	Керівник відділу продажу	1
5	Старший менеджер з продажу	4
6	Менеджер з продажу	5
7	Начальник відділу інформаційних технологій	1
8	Старший програміст	7
9	Програміст	8
10	Системний адміністратор	7

Рис. 5.14. Таблиця Працівники

```
WITH TestCTE(EmpId, Speciality, Manager, Level)
AS (
--корінь ієрархії
SELECT EmpId, Speciality, Manager, 0 AS Level
FROM Employees
WHERE Manager IS NULL
UNION ALL
--рекурсивна частина
(SELECT t1.EmpId, t1.Speciality, t1.Manager, t2.Level+1
FROM Employees t1
INNER JOIN TestCTE t2 ON t1.Manager = t2.EmpId);
--головний запит
SELECT * FROM TestCTE ORDER BY Level;
```

EmpId	Speciality	Manager	Level
1	Директор	NULL	0
2	Головний бухгалтер	1	1
3	Керівник відділу продажу	1	1
4	Начальник відділу інформаційних технологій	1	1
5	Старший менеджер з продажу	4	2
6	Старший програміст	7	2
7	Системний адміністратор	7	2
8	Бухгалтер	2	2
9	Менеджер з продажу	5	3
10	Програміст	8	3

Рис. 5.15. Ієрархія працівників

Використання декількох CTE в одному запиті

SQL дозволяє визначати декілька CTE в одному запиті, на які можна посилатися послідовно, щоб крок за кроком будувати складну логіку.

Приклад. Пошук найбільш вагомих типів продукції

```
WITH FilteredOperations AS
(SELECT * FROM Operations
WHERE StartDate >= '2023-01-01'
),
CategoryItems AS
(
SELECT ItemType, SUM(ItemQty* ItemPrice) AS total_sum
FROM FilteredOperations FO JOIN Items ON FO.ItemId = Items.ItemId
GROUP BY ItemType
),
TopCategories AS
(
SELECT ItemType, total_sum
FROM CategoryItems
WHERE total_sum > 5000
)
SELECT * FROM TopCategories;
```

5.4.8. З'єднання, підзапити, CTE

Мова SQL дозволяє вирішити завдання в різний спосіб. В більшості випадків, запити с підзапитами можна записати з використанням з'єднання, і навпаки. Розглянемо приклад реєстрації в базі даних (рис. 5.16) результатів чемпіонату з футболу (ці дані не є реальними).

Teams

Team_Id	Назва
1	Динамо
2	Шахтар
3	Ворскла

Goals

Home	Visit	H_Goals	V_Goals
1	2	2	2
1	3	1	0
2	1	2	1
2	3	3	0
3	1	2	3
3	2	2	4

Рис. 5.16. Результати чемпіонату з футболу

Приклад. Задачу підрахувати кількість голів, що забила кожна команда можна вирішити двома способами – з'єднанням, з використанням підзапиту або CTE:

Підзапити:

```
SELECT T.name,
(Select SUM(G.H_Goals)
 From Goals AS G
 Where G.Home =  T.team_id)
+
(Select SUM(G.V_Goals)
 From Goals AS G
 Where G.Visit =  T.team_id)
FROM Team AS T;
```

З'єднання:

```
SELECT T.name, (SUM(G1.H_Goals)+ SUM(G2.V_Goals)) AS ALLgoal
```

```
FROM Team AS T
JOIN  Game AS G1 ON G1.Home = T.team_id
JOIN  Game AS G2 ON G2.Visit = T.team_id AND  G2.Home = G1.Visit
GROUP BY T.Name;
```

CTE:

```
WITH ALLResults AS
(SELECT T.Name Name, G1.H_Goals Goals
FROM Team AS T
JOIN  Game AS G1 ON G1.Home = T.team_id)
UNION ALL
(SELECT T.Name Name, G2.V_Goals Goals
FROM Team AS T
JOIN  Game AS G2 ON G1.Visit = T.team_id)

SELECT Name, SUM(Goals) AS ALLgoal
FROM AllResults
GROUP BY Name
```

Який спосіб вибрати, вирішує розробник. Але слід дотримуватися рекомендацій:

- використання операторів з'єднання робить запит більш зрозумілим і полегшує ядру сервера побудову плану запиту;
- підзапити краще використовувати, коли потрібно швидко обчислити агрегатну функцію і використовувати її значення для порівняння;
- з'єднання мають перевагу, якщо список вибору оператора SELECT містить стовпці більш ніж з однієї таблиці.

І CTE, і підзапити дозволяють вам написати запит, а потім написати інший запит, який посилається на перший запит.

Переваги CTE над підзапитом:

- Кілька посилань. Визначивши CTE, ви можете посилатися на нього за назвою кілька разів у наступних SELECT-запитах. У випадку з підзапитом вам потрібно було б щоразу писати повний підзапит.
- Кілька таблиць. Синтаксис CTE є більш читабельним при роботі з декількома таблицями, оскільки ви можете перерахувати всі CTE заздалегідь. У випадку з підзапитами, підзапити будуть розкидані по всьому запиту.

Але CTE не підтримуються у старих версіях SQL, тому підзапити все ще широко використовуються.

5.5. Подання

Подання (view) – це запити SELECT, які зберігаються в базі даних. Після збереження до подання можна звертатися так, ніби воно є таблицею в базі даних. У той час як таблиці бази даних містять фізичні дані, подання не містять даних, але дозволяють діяти так, ніби подання є реальною таблицею з даними. Одного разу створене подання продовжує використовуватись до тих пір, поки його не буде видалено [12].

Подання надають додаткову гнучкість у способах обмеження доступу до даних. З часом вимоги до доступу до цих даних змінюються, але реорганізація таблиць відповідно до нових вимог не є тривіальною справою. Великою перевагою подання є те, що вони дозволяють створювати еквівалент таблиць без необхідності прямого доступу до даних. За

допомогою подання можна надавати різний доступ до одних і тих самих даних для різних груп користувачів, без необхідності змінювати спосіб зберігання даних.

Основні функції подання наступні:

1. Спрощення операцій. При виконанні запитів часто доводиться використовувати агрегатні функції і відображати інформацію про інші поля, а також може знадобитися зв'язати інші таблиці, що призводить до написання додаткових операторів. Якщо така дія відбувається часто, можна створювати подання, надалі просто виконавши інструкцію `SELECT * FROM` з подання.
2. Покращує безпеку. Користувачі можуть запитувати і змінювати тільки ті дані, які вони бачать в поданні.
3. Досягається логічна незалежність від структури реальних таблиць. Подання дозволяють прикладним програмам і таблицям бази даних бути дещо незалежними один від одного. Додатки відокремлюється від таблиці бази даних за допомогою подання.

Подання можуть бути поділені на дві категорії:

- Просте подання. Запит `SELECT` звертається тільки до однієї базової таблиці.
- Складене подання. Запит `SELECT` звертається до кількох базових таблиць.

Синтаксис створення подання:

```
CREATE VIEW Ім'я_подання [(Псевдоніми стовпчиків)]  AS
    SELECT    <підзапит – критерій відбору>
WITH LOCAL CHECK OPTION;
```

де

- Псевдоніми стовпчиків це назви всіх стовпців у поданні. Кількість стовпців, оголошених тут має відповідати кількості стовпців, зазначених оператором SELECT. Якщо не вказано, стовпці у поданні отримують свої імена від стовпців у таблиці. Цей пункт є обов'язковим, коли один або декілька стовпців є похідними і не мають в базової таблиці стовпець для посилання.
- `WITH LOCAL CHECK OPTION` – Використовується лише у поданнях, які дозволяють оновлення своїх базових таблиць. Гарантує, що лише ті дані, які можуть бути прочитані, можуть бути вставлені, оновлені або видалені поданням.

Приклад. Створити подання, у якому реалізовано запит: для кожної спеціальності вивести максимальну погодинну ставку.

```
CREATE VIEW V_max_rate (EmpSpec, Max_rate ) AS
    SELECT EmpSpec, MAX (EmpRate) AS Max_rate
    FROM Employees
    GROUP BY EmpSpec;
```

Звернення до подання:

```
SELECT  EmpSpec
FROM V_max_rate            -- вибірка з представлення
WHERE Max_rate > 700;
```

Обмеження використання подання:

- У всіх полів подання мають бути унікальні імена.

- Якщо будь-яка колонка подання є результатом арифметичних перетворень чи вбудованої функції, її ім'я необхідно задати явно.
- Якщо подання вибирає дані зі зв'язаних таблиць, з іменами колонок, що збігаються, колонкам подання необхідно задати унікальне ім'я.
- Операції `UNION` та `UNION ALL` не дозволяються.
- `ORDER BY` ніколи не використовується у визначенні подання.

Особливий інтерес викликають подання, за допомогою яких дані можна не тільки переглядати а і модифікувати – так звані подання, що оновлюються. Такі подання може змінюватися командами модифікації DML, але модифікація не впливатиме на саме подання. Команди будуть насправді спрямовані до базової таблиці. Такі подання повинні відповідати наступним вимогам:

- У команді `FROM` стоїть посилання лише на одну таблицю.
- Таблиця, на яку посилається речення `FROM`, повинна бути або базовою таблицею, або представленням, що оновлюється.
- Немає операторів `GROUP BY`, `HAVING`, `UNION`, `EXCEPT`, `INTERSECT`.
- У команді `SELECT` перераховані лише імена стовпчиків, тобто. немає обчислень та вбудованих функцій, відсутнє ключове слово `DISTINCT`.
- В базовій таблиці, що оновлюється, відсутні стовпці `NOT NULL`, що не входять в подання (для команди `INSERT`).
- Воно не повинно використовувати підзапити (це обмеження ANSI, яке не передбачено для деяких реалізацій SQL).

Приклад. Подання, що дає можливість редагувати інформацію тільки стосовно електриків:

```
CREATE VIEW V_electrics
AS
SELECT EmpName, EmpRate, Manager
FROM Employees
WHERE EmpSpec = 'електрик'
WITH LOCAL CHECK OPTION;
```

Правило, яке слід пам'ятати при оновленні базової таблиці через подання є те, що всі стовпці базової таблиці, які визначені як `NOT NULL`, повинні отримувати значення для команд додавання або оновлення через подання. Можна зробити це явно, безпосередньо вказав такі значення, або покладаючись на значення за замовчуванням. Крім того, подання не знімають і не накладають обмежень на базову таблицю. Таким чином, значення, що вставляються або оновлюються у базовій таблиці повинні відповідати всім обмеженням, які були спочатку накладені на таблицю через первинні ключі, обмеження `CHECK` тощо.

Матеріалізовані подання

Матеріалізоване подання зберігає дані, повернуті запитом визначення подання, і автоматично оновлюється після змін даних у базових таблицях. Це підвищує продуктивність складних запитів (зазвичай запити з об'єднаннями та агрегатами), а також спрощує обслуговування.

Матеріалізовані подання є надлишковими даними, оскільки їхній вміст можна вивести з визначення представлення та решти вмісту бази даних. Однак у багатьох випадках набагато дешевше прочитати вміст матеріалізованого подання, ніж обчислювати його вміст шляхом виконання запиту, що визначає це подання.

Матеріалізовані подання є важливими для підвищення продуктивності деяких додатків. Розглянемо матеріалізоване подання, яке обчислює загальну зарплату в кожному відділі:

```
CREATE MATERIALIZED VIEW department_total_salary
(dept_name, total_salary) AS
SELECT dept name, SUM (salary) AS total_salary
FROM instructor
GROUP BY dept name;
```

Проблема з матеріалізованими поданнями полягає в тому, що їх потрібно підтримувати в актуальному стані. Коли змінюються базові дані, що використовуються у визначенні подання, то матеріалізовані подання не змінюються автоматично. Наприклад, якщо змінюється розмір заробітної плати співробітника, матеріалізоване подання не відповідатиме базовим даним, і його потрібно буде оновити. В сучасних системах баз даних як тільки подання оголошено матеріалізованим, його вміст оновлюється, коли змінюються базові дані.

5.6. Процедури і тригери

5.6.1. Збережені процедури

Під час програмування в SQL Server введений код спочатку компілюється, потім запускається. Процес компіляції може займати певний час. Однак є можливість написаний блок коду зберегти і заздалегідь скомпілювати. Особливо, якщо код багаторазово застосовується. Для цієї мети використовуються збережені процедури (Stored Procedure – SP), які являють собою набір інструкцій, що виконуються як єдине ціле [9,12].

Збережені процедури пропонують кілька переваг, які роблять їх привабливими для розробників і адміністраторів баз даних:

1. Покращення продуктивності: збережені процедури попередньо скомпільовані та зберігаються в базі даних, що може призвести до швидшого часу виконання порівняно з динамічними запитами SQL. Це особливо корисно для складних запитів і частих операцій.
2. Безпека: SP усувають необхідність надавати дозволи на рівні таблиць. За допомогою SP можна управляти доступом до таблиць БД, тобто реалізувати всі операції над даними без безпосереднього доступу до даних. Також за допомогою збережених процедур можна запобігти атакам типу «ін’єкція SQL». Це допомагає запровадити надійну модель безпеки.
3. Ремонтопридатність: інкапсуляція бізнес-логіки в збережених процедурах полегшує підтримку та оновлення коду. Зміни можна вносити в одному місці, не змінюючи код прикладних програми, що зменшує ризик помилок.

4. Повторне використання: збережені процедури можна повторно використовувати в різних програмах і модулях, сприяючи повторному використанню коду та зменшуючи надмірність.
5. Зниження мережевого трафіку: обробляючи складні операції на сервері, збережені процедури мінімізують обсяг даних, що передаються між клієнтом і сервером, що призводить до зменшення мережевого трафіку. Клієнтська програма може багаторазово виконувати SP, без пересилання її серверу і без повторної компіляції.

Збережені процедури це об'єкт бази даних, що представляє собою набір SQL-інструкцій, який складається з заголовка і тіла. Заголовок визначає ім'я процедури і її параметри. Тіло процедури це SQL-інструкції. Для виконання процедури необхідно виконувати команду Execute або Exec.

Синтаксис створення процедури (на прикладі MS SQL Server):

```
CREATE PROC SP_Name                --назва процедури
(
 @ Param1  DataType,               --визначення параметру з типом даних
 @ Param2  DataType [= Default],   -- параметр зі значенням за замовчуванням
 @ParamN DataType [OUTPUT]         -- параметр що повертається
)
AS
 BEGIN
    --команди процедури….
 END;
```

Змінні, що використовуються в процедурі необхідно декларувати. Наприклад:

```
DECLARE @LastName varchar (50);
```

Командами SELECT або SET змінним присвоюється значення:

```
SET @LastName = 'Smith'
SET @FirstName = 'Bill'
або
SELECT @LastName = 'Smith', @FirstName = 'Bill';
```

Приклад. Використання SP для вставки запису

```
CREATE Proc SP_InsertEmployeeInfo
(@EmpID int,
@EmpName varchar (50),
@EmpSpec varchar (50),
@EmpRate money,
@Manager int
)
AS
BEGIN
-- перевіримо унікальність ключа
IF EXISTS (SELECT EmpId FROM Employees WHERE EmpId = @EmpId)
    BEGIN
    RETURN 0
    END
ELSE
    BEGIN
    INSERT INTO Employees (EmpId, EmpName, EmpSpec, EmpRate, Manager)
    VALUES (@EmpId, @EmmName, @EmpSpecl, @EmpRate, @Manager)
    END
BEGIN
    RETURN 1
END
END;
```

Виконання SP

```
EXEC SP_InsertEmployeeInfo 4375, 'Тарасенко', 'монтажник',200.00, 234
```

Приклад. Використання параметрів зі значеннями за замовчуванням

```
CREATE PROC Update_Price
(@t VARCHAR (20)  = 'Цукерки',
 @p FLOAT = 0.1
)
 AS
 BEGIN
    UPDATE Product
    SET Price = Price * (1 + @p)
    WHERE Type= @t
 END;
```

Звернення до процедури:

1. Позиційна передача параметрів

```
Exec Update_Price 'Фанта', 0.32
Exec Update_Price DEFAULT, 0.32
```

2. Передача параметрів по імені

```
Declare MyProduct VARCHAR (20)
Declare MyPrice float
Select MyProduct= 'Pepsi', MyPrice = 0.1
Exec Update_Price @t = MyProduct, @p = MyPrice
Exec Update_Price @p = MyPrice;
```

Приклад. Повернення кодів стану з SP. За замовчанням SP повертає 0.

```
CREATE PROC InsertEmployeeInfo (@a int)
AS
BEGIN
IF EXISTS (SELECT EmpId FROM Employees WHERE EmpId = @a)
    BEGIN
    RETURN 0
    END
 ELSE
BEGIN
    RETURN 1
 END
END;
```

Перевірка значення що повертається:

```
DECLARE @return_status INT;
EXEC @return_status = InsertEmployeeInfo 3215;
SELECT 'Return Status' = @return_status;
```

5.6.2. Тригери

Тригер – це спеціальний тип збереженої процедури, яка автоматично виконується при появі деякої події (спробі виконати операції видалення, додавання, редагування) [12].

Існує як мінімум три причини, чому використовуються тригери.

По-перше, тригер може бути запущений незалежно від застосунку, що дозволяє використовувати його для відміни підозрілих дій.

По-друге, тригер може контролювати обмеження цілісності. В минулому тригери були єдиним засобом забезпечення цілісності за посиланнями. Сучасні СКБД мають можливість використовувати засоби цілісності за посиланнями, яке більш надійне і якому

надають перевагу. Однак, якщо обмеження цілісності задані в вигляді складних умов і перевірок, використання тригерів є найкращим рішенням.

По-третє, тригери дозволяють підтримувати деяку надлишковість в базі даних в випадках, коли схема БД була денормалізована. Наприклад, якщо в структуру таблиці додано стовпчик з похідними даними, тригер відповідає за створення значень в стовпчику.

Тригери дозволяють:

- каскадно оновлювати таблиці, забезпечуючи комплексну перевірку даних на відміну від CHECK;
- посилатися на поля в інших таблицях і базах даних;
- порівнювати стан даних до і після зміни;
- створювати індивідуальні повідомлення про помилки (на відміну від системних);
- накопичувати аудиторську інформацію;
- підтримувати реплікації.

Недоліки тригерів:

- складність. При переміщенні деяких функцій в базу даних ускладняються завдання її проєктування, і адміністрування;
- прихована функціональність. Перенесення частини функцій в базу даних і збереження їх у вигляді одного або декількох тригерів іноді призводить до приховування від користувача деяких функціональних можливостей. В деяких випадках це може призвести до надання надлишкових можливостей користувачам.
- вплив на продуктивність. Перед виконанням кожної команди по зміні стану бази даних СКБД повинна перевірити тригерні умови з метою з'ясування необхідності запуску *тригера* для цієї команди. Виконання подібних обчислень позначається на загальній продуктивності СКБД, а в моменти пікового навантаження її зниження може стати особливо помітним.

Тригери не мають параметрів і не виконуються явно. Це означає, що тригер запускається тільки при спробі зміни даних. Подія, що змушує тригер виконувати свої дії, називається дозвільною подією і зазвичай говорять, що вона запускає тригер.

Як і будь-який інший об'єкт бази даних тригер створюється по команді CREATE. Синтаксис команди створення тригера:

```
CREATE TRIGGER <ім'я_тригера>
ON <ім'я подання або таблиці>
[WITH ENCRYPTION]
{{{FOR|AFTER}<[DELETE][,][INSERT] [,][UPDATE]>}| INSTEAD OF}
 [NOT FOR REPLICATION]
AS
<sql -оператори >
```

де

- ON <*ім'я* > – ім'я об'єкта, для якого тригер використовується;
- DELETE, INSERT, UPDATE – дозвільні події;
- AFTER, INSTEAD OF – момент спрацювання тригера;
- WITH ENCRYPTION – кодує текст тригера;

- NOT FOR REPLICATION – змінює правила запуску тригера. Такий тригер не буде стартувати про виконанні над таблицею операцій пов'язаних з реплікацією даних.

При спрацюванні тригера створюються таблиці INSERTED і DELETED (в інших СКБД такі таблиці мають назву NEW і OLD). INSERTED – для зберігання добавлених записів, DELETED – для зберігання видалених записів. Таблиці доступні тільки для тригера і існують тільки при виконанні тригера.

Таблиця може мати довільну кількість тригерів будь-яких типів (INSERT, UPDATE, DELETE). За умовчанням тригер виконується, коли зміна даних завершена. Якщо вказати опцію BEFORE, спочатку виконується тригер, а потім операція зміни. В MS SQL Server опція BEFORE не використовується, можна вказати опцію INSTEAD OF і створюється тригер, який виконується замість зміни даних.

У Microsoft SQL Server використовуються два види тригерів AFTER (за замовчуванням) та INSTEAD OF , а також три їх типу: INSERT, UPDATE, DELETE. Тригери вставки INSERT – стартують кожен раз при додаванні в таблицю нового запису при цьому створюється таблиця INSERTED. Тригери видалення DELETE – стартують кожен раз при видаленні з таблиці записів і, як наслідок, створюється таблиця DELETED. Тригер редагування UPDATE – стартує при внесенні змін в існуючі записи таблиці, оскільки при редагуванні виконуються дві операції видалення і вставки, то і службових таблиць створюється дві – INSERTED та DELETED.

Приклад. Тригер, який забороняє корекцію даних в таблиці PROGRESS між сесіями.

```
CREATE TRIGGER ProgressTerm
ON PROGRESS
FOR INSERT, UPDATE, DELETE
AS
IF  EXISTS
(SELECT 'TRUE'
 FROM Progress
 -- NTerm  - номер семестру
 WHERE (DATEPART(MM, GETDATE())<>'01' AND NTerm  %2=1)
 OR (DATEPART(MM, GETDATE())<>'06' AND  NTerm  %2=0))
BEGIN
   RAISERROR('Не можна виправляти оцінку!!!',20,1)
-- Відкат транзакції при помилці
      ROLLBACK TRAN
END;
```

Тепер будь-яка спроба додавати чи змінити дані в не сесійний період згенерує помилку. Виконаємо наступне

```
UPDATE Progress SET mark=2 WHERE NRecordBook='050001';
INSERT INTO Progress VALUES ('050001',1,2,1,4,5);
```

На що отримаємо:

```
Server: Msg 50000
«Не можна виправляти оцінку!!!»
```

В MS SQL SERVER тригер завжди спрацює після виконання операції (INSERT, UPDATE, DELETE). Відповідно, вся перевірки наявності або відсутності рядка

виконується після того, як рядок вже додано, або видалено. Тому, рекомендується використовувати тригер з опцією INSTEAD OF.

Наприклад, таким чином:

```
CREATE TRIGGER EmpInsertion
ON Employees
INSTEAD OF INSERT
AS
BEGIN
 IF EXISTS
 (SELECT 'TRUE'
  FROM INSERTED INNER JOIN Employees ON INSERTED.EmpId = Employees.EmpId
BEGIN
    RAISERROR('Не можна додати !', 16, 1)
    ROLLBACK TRAN
 END
ELSE
     INSERT  Employees  SELECT * FROM INSERTED
END;
```

5.7. Операції зміни даних

Мова маніпулювання даними (DML) – це набір спеціальних команд, які дозволяють маніпулювати даними, що зберігаються в існуючій таблиці. Ці команди використовуються для виконання певних операцій, таких як вставка, видалення, оновлення даних в таблицях бази даних [9].

Дані в таблиці можуть бути додані, скориговані та видалені трьома командами мови SQL з категорії DML:

- INSERT (додати),
- UPDATE (модифікувати),
- DELETE (видалити).

5.7.1. Команда INSERT

Синтаксис команди:

```
INSERT  INTO < table name > [(coulmn_names)]
           VALUES ( < value >, < value > . . . ) ;
```

Оператор INSERT додає рядки даних до таблиці або подання одним з наступних методів:

1. Фактичні значення, що вставляються до кожного стовпчика запису, можуть бути оголошені (це найпоширеніший спосіб).

Приклад. Додати один рядок в таблицю Operations з вказанням стовпців, в які будуть внесені значення:

```
INSERT INTO Operations (EmpId, ItemId,  StartDate)
VALUES (1284, 485, '13.11.2001' );
```

2. Один або декілька рядків можна вставити, використовуючи значення за замовчуванням, вказані для стовпця за допомогою інструкцій CREATE TABLE або ALTER TABLE.

Приклад. Додати кілька рядків в таблицю Operations. Порядок значень визначається порядком стовпців в команді CREATE TABLE

```
INSERT INTO Operations
VALUES (1284, 485, '13.11.2001'),
VALUES (1286, 485, '14.11.2001'),
VALUES (1289, 487, '15.11.2001');
```

3. Набір результатів оператора SELECT можна вставити у таблицю, заповнюючи його багатьма записами одночасно. Таким чином можна створити копію таблиці. Синтаксис команди:

```
INSERT  INTO < table name > [(coulmn_names)]
             query;
```

Приклад. Створити копію таблиці Items

```
INSERT INTO Items_2
 SELECT ItemId, ItemType, ItemPrice, ItemName
 FROM  Items
WHERE ItemType = 'чайник';
```

Команда INSERT завжди завершається невдало за таких обставин:

- Коли виникає невідповідність типу даних між стовпцем та його значенням. Але деякі платформи баз даних автоматично і неявно перетворюють певні типи даних. Наприклад, SQL Server автоматично перетворить значення дати в символьний рядок для вставки в стовпець VARCHAR.
- Коли стовпець визначено як NOT NULL, а значення, що вставляється, є Null.
- Якщо значення, що вставляється, повторюється при наявності UNIQUE або PRIMARY KEY обмежень.
- Коли значення, що вставляються, не відповідають вимогам обмеження CHECK.
- Коли вставлене значення обмежене умовою FOREIGN KEY, оскільки значення не є похідним від оголошеного первинного ключа батьківської таблиці.

Найбільш поширеною помилкою є невідповідність між кількістю стовпців та кількістю значень що додаються. Іншою поширеною проблемою при виконанні інструкції INSERT, є невідповідність розміру між значенням і цільовим стовпцем. Наприклад, вставка довгого рядка у стовпчик CHAR(5) або вставка дуже великого цілого числа у стовпчик TINYINT. Залежно від платформи, невідповідність розмірів може спричинити пряму помилку і відкат транзакції, або ж сервер бази даних може просто обрізати зайві дані. Будь-який зазначений результат небажаний.

5.7.2. Команда UPDATE

Оператор UPDATE змінює існуючі дані в таблиці. Треба бути дуже обережним при виконанні оператора UPDATE без оператора WHERE, оскільки оператор впливає на кожен рядок у всій таблиці. Ви можете оновити скільки завгодно стовпців в однієї команді.

Синтаксис команди:

```
UPDATE table_name
SET columnname=new_value[,columnname=new_value….]
WHERE (condition);
```

Приклад. Підвищити на 5 відсотків погодинну ставку кожного електрика. Змінити назву спеціальності.

```
UPDATE Employees
SET  EmpRate = 1,05 * EmpRate, EmpSpec = 'електромонтажник'
WHERE  EmpSpec = 'електрик';
```

Або в такій формі:

```
UPDATE Employees
SET  (EmpRate, EmpSpec) = (1,05 * EmpRate, 'електромонтажник')
WHERE  EmpSpec = 'електрик';
```

Нові значення також можна отримати за допомогою підзапиту. Такий підзапит повинен повертати одне значення, наприклад, значення агрегатної функції.

5.7.3. Команда DELETE

Команда DELETE видаляє один або декілька записів, які задовольняють заданої умові.

Синтаксис команди:

```
DELETE  FROM table_name
WHERE (condition);
```

Оператор DELETE видаляє рядки з таблиці або подання. Звільнене місце буде повернуто до бази даних, де знаходиться таблиця, хоча це може статися не одразу. Однак майте на увазі, що стандарт SQL нічого не говорить про звільнення місця – це деталь реалізації.

Простий оператор DELETE, який видаляє всі записи в заданій таблиці, не містить оператора WHERE, як у наведеному нижче прикладі:

```
DELETE FROM Operations;
```

Можна використовувати будь-який допустимий оператор WHERE для фільтрації записів, які ви не хочете видаляти.

Приклад. Припустимо, що всіх робітників, чий бригадир має табельний номер 1520, було звільнено, і необхідно видалити відповідні рядки з бази даних.

```
DELETE  FROM  Employees
WHERE Manager = 1520;
```

Оператор DELETE рідко використовується без речення WHERE, оскільки це призведе до видалення всіх рядків з таблиці. Бажано спочатку виконати команду SELECT з тім самим реченням WHERE. Таким чином, ви можете бути впевнені, які саме записи буде видалено.

Якщо необхідно видалити всі рядки в таблиці, слід розглянути можливість використання нестандартного, але дуже поширеного оператора TRUNCATE TABLE у тих базах даних, які підтримують таку команду. TRUNCATE TABLE зазвичай є швидшим способом фізичного видалення всіх рядків, оскільки видалення окремих записів не записується до журналу, а пам’ять звільняється одразу. Зменшення накладних витрат на ведення журналу значно економить час при видаленні великої кількості записів, але на деяких платформах це робить неможливим відкат оператора TRUNCATE. Крім того, на

деяких платформах баз даних платформ всі зовнішні ключі в таблиці повинні бути вилучені перед виконанням інструкції TRUNCATE.

Контрольні запитання

1. Дані два відношення R1={<u>AB</u>} и R2={<u>A</u>C}. Первинні ключі підкреслено. В якій послідовності потрібно створювати таблиці, щоб задати обмеження цілісності, використовуючи тільки команду CREATE?
2. З якою ціллю використовується пропозиція ON DELETE CASCADE?
3. У чому відмінність обмежень PRIMARY KEY та UNIQUE?
4. Які оператори в команді SELECT є обов'язковими?
5. В чому різниця між функціями WHERE та HAVING?
6. За допомогою яких операторів в команді SELECT здійснюється перевірка входження результату обчислення вираження в задану множину?
7. Як потрібно створити запит до таблиці R1={AB} и R2={CD}, щоб отримати їх декартів добуток?
8. В якої частині команди SQL можуть бути використані підзапити?
9. Який порядок виконання корельованих запитів?
10. Які вирази допустимі для основного запита не можуть використовуватись в підзапиті?
11. Які оператори порівняння працюють з декількома рядками?
12. Які оператори порівняння працюють тільки з одним рядком?
13. Які вирази не можуть містити запит, що визначає подання?
14. В чому перевага збереженої процедури?
15. Які операції не можуть виконуватися в тригерах?

Тестові завдання

1. Яка з наведених команд DDL не відноситься до SQL:
   A. CREATE TABLE
   B. MODIFY TABLE
   C. CREATE DOMAIN
   D. DROP TABLE
2. Які з наступних тверджень про мову SQL є правильними?
   A. У таблиці SQL не можна вставляти повторювані рядки
   B. Таблиці SQL не мають ніякого впорядкування стовпців
   C. Таблиці SQL не обов'язково повинні мати атомарні значення стовпців
   D. Таблиці SQL можуть мати значення за замовчуванням для кожного стовпця
3. Що з наведеного нижче не можна включати в команду CREATE TABLE в SQL?

A. Імена та типи даних стовпців таблиці
B. Первинний та зовнішні ключі таблиці
C. Значення за замовчуванням для стовпців таблиці
D. Хто має право доступу до таблиці

4. Яке з наступних тверджень не є правильним?
   A. SQL-запит не повинен мати більше одного з'єднання в одному запиті
   B. Всі імена стовпців в SQL-запиті повинні бути визначені іменем таблиці
   C. Всі імена таблиць, вказані в реченні SQL `FROM`, повинні бути різними
   D. SQL-запит не повинен мати функції агрегації в реченні `HAVING`
   E. Все перераховане вище
5. Яке з наведених нижче тверджень не є правильним при використанні функцій в операторах SQL?
   A. `DISTINCT` потрібно використовувати, якщо перед застосуванням функції потрібно видалити дублікати
   B. Всі випадки використання `COUNT` повинні включати `DISTINCT`
   C. Будь-які Null у стовпчику аргументів ігноруються, коли використовуються такі функції, як `AVG`, `MAX` і `MIN`
   D. Якщо аргумент є порожньою множиною, `COUNT` повертає нуль, тоді як інші функції повертають Null
6. Яке з наступних тверджень про мову SQL є правильним?
   A. SQL автоматично видаляє повторювані рядки з результатів
   B. SQL автоматично сортує рядки з результатів
   C. SQL не надає команд для безпосереднього обчислення всіх операторів реляційної алгебри
   D. SQL не допускає ніяких інших з'єднань, окрім з'єднань за умовою рівності
7. Таблиця Results містить інформацію щодо результатів сесії. Розглянемо наступний запит:

```
SELECT student_id
FROM Results;
```

Є 100 студентів і 500 записів, хоча 5 студентів ще не склали жодного іспиту. Кількість рядків, які повертає наведений вище запит, є такою:

   A. 100
   B. 500
   C. 95
   D. Неможливо визначити
8. Таблиця Subject містить перелік дисциплін, Results містить інформацію щодо результатів сесії. Розглянемо наступний запит:

```
SELECT subject_name
FROM Subject, Results
GROUP BY subject_id
HAVING MAX(AVG(mark)) > 60;
```

Яке з наступних тверджень є правильним для цього запиту?

   A. Запит помилковий
   B. Запит виводить імена предметів, у яких максимальна оцінка перевищує 60

C. Запит повертає назви предметів, де максимальний середній бал перевищує 60
D. Запит повертає назви предметів, де середній бал є максимальним з усіх предметів і перевищує 60

9. Таблиця Results містить інформацію щодо результатів сесії. Розглянемо наступний запит:

```
SELECT AVG(mark)
FROM Results
WHERE subject_id = 'IT302';
```

Яке з наведених нижче значень поверне запит, якщо значеннями mark в рядках для IT302 є 90, 60 і Null?

A. 75
B. 50
C. Null
D. Не визначено

10. Таблиця Results містить інформацію щодо результатів сесії. Розглянемо наступний запит:

```
SELECT student_id
FROM Results
WHERE subject_id = 'IT1500'
AND subject_id = 'IT1200';
```

Яка з наступних відповідей є правильною?

A. Запит є помилковим
B. Запит повертає ідентифікатори студентів, які вивчають IT1500 та IT1200
C. Запит повертає Null результат
D. Нічого з перерахованого вище

11. Таблиця Subject містить перелік дисциплін, Results містить інформацію щодо результатів сесії. Розглянемо наступний запит:

```
SELECT subject_name
FROM Subject, Results
GROUP BY subject_id;
```

Яке з наступних тверджень є правильним для цього запиту?

A. Запит помилковий
B. Запит повертає імена всіх дисциплін
C. Запит повертає один раз назви всіх предметів, на які записані студенти
D. Запит не може бути сформульовано без з'єднання або підзапиту
E. Жодне з перерахованих вище

12. Таблиця Student містить перелік студентів, Results містить інформацію щодо результатів сесії. Розглянемо наступний запит:

```
SELECT student_name
FROM Student, Results
WHERE Student.student_id = Results.student_id
AND mark IS NULL;
```

Яке з наступних тверджень є правильним для цього запиту?

A. Запит помилковий
B. Запит виводить ім'я кожного з тих студентів, які не мають жодної оцінки з усіх предметів

C. Запит виводить прізвища всіх студентів, які не мають оцінок з деяких предметів
D. Запит повертає ім'я студента кожного разу, коли він не має жодної оцінки з якогось предмета
E. Вищевказаний запит не повертає імена студентів, що повторюються

13. Таблиця Subject містить перелік дисциплін, Results містить інформацію щодо результатів сесії. Розглянемо наступний запит:

```
SELECT subject_id
FROM Subject, Results
WHERE Subject.subject_id = Results.subject_id
GROUP BY subject_id
HAVING AVG(mark) >70;
```

Яке з наступних тверджень є правильним для цього запиту?

A. Запит не є допустимим
B. Запит повертає імена всіх предметів
C. Запит повертає назви предметів, які мають середній бал більше 70
D. Запит повертає назви предметів, у яких деяка оцінка перевищує 70
E. Нічого з перерахованого вище

14. . Адміністратору бази даних у банку потрібно створити механізм, який автоматично надсилає сповіщення про низький баланс клієнтам, коли залишок на їхньому рахунку падає нижче певного порогу. Який об'єкт бази даних потрібно використати для виконання цього завдання?

A. Збережена процедура
B. Функція
C. Тригер перед оновленням
D. Тригер після оновлення

15. Таблиця Results містить інформацію щодо результатів сесії. Розглянемо наступний запит:

```
SELECT student_id
FROM Results r1
WHERE NOT EXISTS
(SELECT *
  FROM Results r2
  WHERE r1.student_id = r2.student_id
  AND mark < 50);
```

Яке з наступних тверджень є правильним щодо вищенаведеного запиту?

A. Запит помилковий
B. Запит повертає ідентифікатор студента, який має оцінку нижче 50 з кожного предмету, який він вивчає
C. Запит повертає ідентифікатор студентів, які мають оцінку нижче 50 з якогось предмета, який вони вивчають
D. Запит повертає ідентифікатор студентів, які мають оцінку 50 або вище з кожного предмету, який вони вивчають

6. АНАЛІТИЧНА ОБРОБКА ДАНИХ ЗАСОБАМИ SQL

Аналіз даних – це набір інструментів і методів, що використовуються для отримання інформації з даних. Типовою сферою застосування аналітичної обробки даних є бізнес-аналітика (BI – Business Intelligence). BI передбачає використання інструментів і методів аналізу даних для перетворення необроблених даних на змістовні ідеї, які використовуються для прийняття бізнес-рішень. Хоча аналіз даних традиційно відбувається поза межами СКБД, за допомогою спеціалізованих інструментів або мов, таких як Excel, R та Python, мова SQL включає в себе потужний набір функцій, корисних для аналітичної обробки. SQL відіграє важливу роль в BI, забезпечуючи ефективні запити, маніпуляції з даними та виокремлення інформації з великих наборів даних. Деякі методи аналізу можуть бути реалізовані в SQL без надмірної складності. Це стосується методів створення зведених статистичних даних, зведених таблиць і аналізу трендів за допомогою аналітичних функцій SQL [11].

6.1. Реалізація базових операцій над множинами

Спочатку розглянемо питання реалізації операцій реляційної алгебри мовою SQL.

Об'єднання таблиць

Синтаксис `UNION [ALL]`

```
SELECT column_names FROM table1
UNION
SELECT column_names FROM table2;
```

Для реляційних відношень умовою об'єднання є еквівалентність схем, а для запитів умова виглядає так:

- всі запити повертають однакову кількість атрибутів;
- типи атрибутів попарно збігаються.

Якщо умова (1) не виконується , можна штучним чином скоригувати запити для виконання умови.

Приклад. Об'єднання таблиць рис. 6.1

Countries

Id	Name
1	Ukraine
2	Poland
3	Hungary

Cities

Id	Name	Country_id
1	Kyiv	1
2	Lviv	1
3	Warsaw	2
4	Budapest	3

Countries UNION Cities

Id	Name
1	Ukraine
2	Poland
3	Hungary
1	Kyiv
2	Lviv
3	Warsaw
4	Budapest

Рис. 6.1. Об'єднання таблиць

Приклад. Об'єднання запитів з різною кількістю атрибутів. В другому запиті додано штучний атрибут (``):

```
SELECT City, Country FROM Customers
WHERE Country = 'Україна'
UNION
SELECT Town, ' ' FROM Suppliers
WHERE Country = 'Україна'
ORDER BY 1;
```

Оператор ORDER BY може використовуватись тільки після останнього запиту об'єднання. Причому використовують колонки (або їх псевдоніми) першого SELECT. За замовчанням результати запиту відсортовані по першому стовпцю по зростанню.

Перетин таблиць

Синтаксис INTERSECT

```
SELECT column_names FROM table1
INTERSECT
SELECT column_names FROM table2;
```

Умови виконання оператора INTERSECT такі ж, що і для операції об'єднання.

Магазин_№_1

Prod_ID	ProdName	Maker	Quantity
1	хліб	AB	100
2	молоко	CD	65
3	м'ясо	EF	75
4	риба	GH	60
5	цукор	IJ	45

Магазин_№_2

Prod_ID	ProdName	Maker	Quantity
1	хліб	QW	85
2	молоко	LD	70
3	сир	MV	45
4	масло	DG	62
5	риба	LN	55

Рис. 6.2. Вхідні таблиці

Приклад. Перетин таблиць наведених на рис. 6.2:

```
SELECT ProdName FROM Магазин_№_1
INTERSECT
SELECT ProdName FROM Магазин_№_2;
```

Результат операції наведено на рис. 6.3.a.

Різниця таблиць

Синтаксис EXCEPT (MINUS)

```
SELECT column_names FROM table1
EXCEPT
SELECT column_names FROM table2;
```

Різниця таблиць наведених на рис. 6.2:

```
SELECT ProdName FROM Магазин_№_1
MINUS
SELECT ProdName FROM Магазин_№_2;
```

Результат операції наведено на рис. 6.3.b.

ProdName
хліб
молоко
риба

a

ProdName
м'ясо
цукор

b

Рис. 6.3. Результати операції

Ділення таблиць

Мовою SQL ділення можна реалізувати в різний спосіб:

1. Подвійне заперечення. Цей варіант найгроміздкіший – основний запит і два вкладених запити з'єднаних умовою `NOT EXISTS`.
2. Пошук порожньої різниці множин (дивись в розділі 5 приклад «Вивести працівників, які виготовляли вироби всіх видів»). Недоліки способу – дуже повільно і некоректний результат при діленні на 0. В наведеному прикладі – якщо X пуста множина.
3. Універсальний спосіб, швидкий, коректне ділення на 0:

D1(a,b) – таблиця – ділене

D2 (в) – таблиця – дільник

a,b – набори атрибутів.

```
SELECT a
FROM D1 INNER JOIN D2 ON D1.b = D2.b
GROUP BY b
HAVING COUNT(*) = (
    SELECT COUNT(*)
    FROM D2);
```

Інші операції реляційної алгебри

Всі інші операції РА реалізуються в мові SQL безпосередньо.

- Розширений декартів добуток можна отримати в операторі `FROM`:

  `FROM t1,t2`
- Фільтрація або вибірка реалізується за допомогою оператора `WHERE`:

  `WHERE a=b`
- Проекція реалізується списком вибору команди `SELECT`:

  `SELECT a, b, c`
- Різні види з'єднань відношень мають очевидний аналог в мові SQL.

6.2. Логічні умови в запитах SQL

Часто вихідні дані, представлені у результатах запиту, можуть бути не в бажаній формі. Може знадобитися видалити значення, замінити значення або зіставити значення з іншими значеннями. Для виконання цих завдань у мові SQL передбачено широкий спектр операторів і функцій.

Функція `CASE` забезпечує функціональність IF/THEN/ELSE в операторах `SELECT`, `INSERT` або `UPDATE`. Вона обчислює список умов і повертає одне значення з декількох можливих. `CASE` має два варіанти використання: простий і з пошуком. Прості CASE-вирази порівнюють одне значення зі списком інших значень і повертають результат, пов'язаний з першим знайденим значенням. CASE-вирази з пошуком дозволяють аналізувати декілька логічних умов і повертають результат, пов'язаний з першою з них, яка є істинною [10].

Перший варіант, простий CASE-вираз:

```
CASE ColumnOrExpression
    WHEN value1 THEN result1
    WHEN value2 THEN result2
    --повторюємо необхідну кількість разів
    [ELSE DefaultResult]
END;
```

Другий варіант, простий CASE-вираз з пошуком:

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    --повторюємо необхідну кількість разів
    [ELSE DefaultResult]
END;
```

Приклад. Перетворення ідентифікатора в назву.

В початковій таблиці Foods (рис. 6.4) категорія товару позначена одним символом і його необхідно перетворити в назву.

Foods

FoodID	CategoryId	Description
1	Ф	Яблуко
2	Ф	Апельсин
3	Г	Гірчиця
4	О	Морква
5	В	Вода

Category	Description
Фрукти	Apple
Фрукти	Orange
Інше	Mustard
Овочі	Carrot
Інше	Water

Рис. 6.4. Результати перетворення ідентифікатора в назву

Простий CASE:

```
SELECT
  CASE CategoryId
  WHEN 'Ф' THEN 'Фрукти'
  WHEN 'О' THEN 'Овочі'
  ELSE 'Інше'
  END AS Category, Description
FROM Foods;
```

Пошуковий CASE:

```
SELECT
CASE
  WHEN CategoryId = 'Ф' THEN 'Фрукти'
  WHEN CategoryId = 'О' THEN 'Овочі'
  ELSE 'Інше' END AS Category, Description
FROM Foods;
```

А от як такий же результат можна отримати, якщо початкова таблиця існує в вигляді крос-таблиці (рис 6.5)

Foods

FoodD	Fruit	Vegetable	Spice	Beverage	Description
1	X				Яблуко
2	X				Апельсин
3			X		Гірчиця
4		X			Морква
5				X	Вода

Рис. 6.5. Таблиця в вигляді кросс-таблиці

```
SELECT
  CASE
  WHEN Fruit = 'X' THEN 'Фрукти'
  WHEN Vegetable = 'X' THEN 'Овочі'
  ELSE 'Інше' END AS Category, Description
FROM Foods;
```

Приклад. Використання CASE в агрегатних функціях.

В таблиці Warehouse (рис. 6.6) необхідно знайти підсумки за категоріями товарів:

Warehouse

ID	Category	Product	Qty	Qty_type
1	Фрукти	Яблуко	3	кг
2	Фрукти	Персик	2	уп
3	Овочі	Картопля	4	кг
4	Соки	Вишневий сік	5	уп
5	Соки	Березовий сік	2	п
6	Овочі	Цибуля	4	уп

Category	Кг	Уп	Інше
Фрукти	3	2	0
Овочі	4	4	0
Соки	0	5	2

Рис. 6.6. Підсумки за різними категоріями

```
SELECT Category,
  SUM(CASE WHEN qty_type = 'кг' THEN quantity ELSE 0 END) AS Кг,
  SUM(CASE WHEN qty_type = 'уп' THEN quantity ELSE 0 END) AS Уп,
  SUM(CASE WHEN qty_type NOT IN ('кг', 'уп')
       THEN quantity ELSE 0 END) AS Інше
FROM Warehouse
GROUP BY Category;
```

6.3. Ранжування даних

Іноді може знадобитися інформація про характеристики конкретного рядку даних щодо його положення у наборі даних. Типовим прикладом є ранг поточного рядка, тобто, кількість рядків, які знаходяться перед поточним рядком. [15]

Розглянемо функції ранжування на прикладі біржових даних. На рис. 6.7 приведено список акцій із зазначенням їх символу, назви, біржі, на якій вони торгуються, і співвідношення ціни та прибутку (PE Price Earnings Ratio). Наприклад, Apple (AAPL) торгується на NASDAQ і має коефіцієнт PE 36.

Stocks

StockId	StockName	Exchange	PE
AAPL	Apple	NASDAQ	36
AMZN	Amazon.com	NASDAQ	80
BAC	Bank of America	NYSE	17
GE	General Electric	NYSE	29
GOOG	Alphabet	NASDAQ	40
HSY	The Hershey	NYSE	27
KO	The Coca-Cola	NYSE	33
MCD	McDonalds	NYSE	36
MMM	3M	NYSE	22
MSFT	Microsoft	NASDAQ	39
NFLX	Netflix	NASDAQ	61
ORCL	Oracle	NASDAQ	18
SBUX	Starbucks	NASDAQ	205
TGT	Target	NYSE	22
WMT	Wal-Mart	NYSE	30

Рис. 6.7. Вхідні дані для ранжування

Існує декілька функцій ранжування: ROW_NUMBER(), RANK(), DENSE_RANK(). Загальний синтаксис визначення рангу:

```
Rank_Function()OVER (ORDER BY expression [[ASC]|DESC])
```

де *expression* – показник, по якому використовується ранжування.

1. Функція ROW_NUMBER() створює колонку з номерами рядків на основі вказаного порядку іншого стовпця або виразу, пов'язаного з функцією. Після розміщення рядків у зазначеному порядку номери згенерованих рядків починаються з 1 і послідовно збільшуються на 1 (рис. 6.8). Функція ROW_NUMBER() не потребує параметрів.

```
SELECT ROW_NUMBER()OVER (ORDER BY PE) AS Row,
StockSymbol AS Symbol,
StockName AS Name,
Exchange AS Exchange,
PE AS 'PE Ratio'
FROM Stocks
ORDER BY PE;
```

Row	Symbol	Name	Exchange	PE Ratio
1	BAC	Bank of America Corporation	NYSE	17
2	ORCL	Oracle Corporation	NASDAQ	18
3	TGT	Target Corporation	NASDAQ	22
4	MMM	3M Company	NYSE	22
5	GE	General Electric Company	NASDAQ	23
6	HSY	The Hershey Company	NYSE	27
7	WMT	Wal-Mart Inc	NYSE	30
8	KO	The Coca-Cola Company	NYSE	33
9	MCD	McDonalds Corporation	NYSE	36
10	AAPL	Apple Inc	NYSE	36
11	MSFT	Microsoft Corporation	NYSE	39
12	GOOG	Alphabet Inc	NYSE	40
13	NFLX	Netflix Inc	NASDAQ	61
14	AMZN	Amazon.com Inc	NASDAQ	80
15	SBUX	Starbucks Inc	NASDAQ	205

Рис. 6.8. Ранжування на номером рядку стовпця PE Ratio

2. Функція RANK() така ж, як і ROW_NUMBER(), за винятком того, що якщо два чи більше рядків мають однакове значення для вказаного стовпця або виразу, їм обом присвоюється однакове число. Наприклад, якщо другий і третій рядки мають однакове значення, створені ранги будуть 1, 2, 2, 4 тощо. Оскільки два рядки зі значенням 2 мають однакове значення, SQL пропускає число 3.

3. Функція DENSE_RANK() така ж, як і функція RANK(), за винятком того, що вона не пропускає жодних чисел, навіть якщо є повторювані значення. У попередньому прикладі згенерований щільний ранг буде 1, 2, 2, 3 тощо. Число 3 не пропускається.

Ось як функції працюють разом (рис. 6.9):

```
SELECT ROW_NUMBER()OVER (ORDER BY PE) AS Row,
 RANK()OVER (ORDER BY PE) AS 'Rank',
 DENSE_RANK()OVER (ORDER BY PE) AS 'Dense Rank',
StockSymbol AS Symbol,
PE AS 'PE Ratio'
FROM Stocks
ORDER BY PriceEarningsRatio;
```

Row	Rank	Dense Rank	Symbol	PE Ratio
1	1	1	BAC	17
2	2	2	ORCL	18
3	3	3	TGT	22
4	3	3	MMM	22
5	5	4	GE	23
6	6	5	HSY	27
7	7	6	WMT	30
8	8	7	KO	33
9	9	8	MCD	36
10	9	8	AAPL	36
11	11	9	MSFT	39
12	12	10	GOOG	40
13	13	11	NFLX	61
14	14	12	AMZN	80
15	15	13	SBUX	205

Рис. 6.9. Результати ранжування різними способами

6.4. Проміжні підсумки в агрегованих даних

Коли дані згруповані по декільком показникам, запит повертає агреговане значення для кожного набору показників. Але якщо треба отримати також агреговані значення по кожному показнику окремо, необхідно модифікувати запит. Для цього використовуються проміжні підсумки. Проміжні підсумки зазвичай надаються за допомогою додаткових рядків, доданих із детальними даними, які підсумовують ключові стовпці. Особливості проміжних підсумків розглянемо на прикладі наступної таблиці (рис. 6.10)

InventoryID	Category	Subcategory	Product	Quantity
1	Меблі	Стілець	Крісло для керівник	3
2	Меблі	Стілець	Поворотний робочий стіл	2
3	Меблі	Письмовий стіл	Студентський комп'ютерний стіл	4
4	Папір	Папір у пачках	Багатоцільовий папір	5
5	Папір	Папір у пачках	Білий папір для лазера	2
6	Папір	Блокнот	College Ruled Paper	4

Рис. 6.10. Вхідні дані для проміжних підсумків

Можна отримати підсумки за категоріями та підкатегоріями, якщо згрупувати дані належним чином. Якщо необхідно отримати рядок проміжних підсумків для кожної категорії та останній рядок наприкінці с загальними підсумками, необхідно використати функцію `ROLLUP()`.

Приклад. Використання функції `ROLLUP` (рис. 6.11).

```
SELECT Category, Subcategory,
SUM(Quantity) AS 'Quantity'
FROM Inventory
GROUP BY ROLLUP(Category, Subcategory);
```

Category	Subcategory	Quantity
Меблі	Стілець	5
Меблі	Письмовий стіл	4
Меблі	NULL	9
Папір	Папір у пачках	7
Папір	Блокнот	4
Папір	NULL	11
NULL	NULL	20

Рис. 6.11. Підсумки за категоріями та підкатегоріями

Зведені дані добре працюють у ситуаціях, коли дані мають ієрархічну структуру. У попередньому прикладі існувала природна ієрархія від категорії до підкатегорії. Ключове слово ROLLUP надає проміжні підсумки для кожної категорії та підсумок у кінці. Однак в інших ситуаціях, коли дані не є ієрархічними, також виникає необхідність додати рядки проміжних підсумків. Щоб проілюструвати, розглянемо такі дані в таблиці продажів за різними напрямками, інтернет, або роздріб (рис. 6.12):

SalesSummary

SalesDate	CustomerID	City	Channel	SalesAmt
12/1/2021	101	Київ	Онлайн	50
12/1/2021	102	Київ	Ритейл	30
12/1/2021	103	Львів	Онлайн	120
12/2/2021	145	Львів	Ритейл	90
12/2/2021	180	Київ	Ритейл	300
12/2/2021	181	Львів	Онлайн	130
12/2/2021	182	Київ	Онлайн	520
12/2/2021	184	Київ	Ритейл	80

Рис. 6.12. Вхідні дані з продажів

Використання ключового слова CUBE має на меті виразити багатовимірний погляд на дані. У той час як ключове слово ROLLUP дозволяє деталізувати дані в ієрархічній формі, ключове слово CUBE дозволяє використовувати кілька вимірів. У прикладі (рис. 6.13) можна переглянути проміжні підсумки за містом і каналом продажу:

```
SELECT State, Channel, SUM(SalesAmount) AS 'Sales Amount'
FROM SalesSummary
GROUP BY CUBE(City, Channel)
ORDER BY City, Channel;
```

City	Channel	Sales Amount
NULL	NULL	1320
NULL	Онлайн	820
NULL	Ритейл	500
Київ	NULL	980
Київ	Онлайн	570
Київ	Ритейл	410
Львів	NULL	340
Львів	Онлайн	250
Львів	Ритейл	90

Рис. 6.13. Результат використання функції CUBE

Зі збільшенням кількості рядків результат вже важче інтерпретувати. Використовуючи функцію PIVOT(), можна створити ці самі дані в макеті крос-таблиці, що полегшує пошук потрібних значень даних [10]. Мета – отримувати дані у такому форматі, щоб можна отримати просту відповідь типу: 2021-12-01 в місті Львів продажів не було (рис. 6.14):

```
SELECT * FROM
(SELECT SalesDate, Channel, City, SalesAmount
  FROM SalesSummary) AS mainquery
PIVOT (SUM(SalesAmount) FOR City IN ([Київ], [Львів])) pvt
ORDER BY SalesDate;
```

SalesDate	Channel	Київ	Львів
2021-12-01	Онлайн	50	120
2021-12-01	Ритейл	30	NULL
2021-12-02	Онлайн	520	130
2021-12-02	Ритейл	380	90

Рис. 6.14. Результат використання функції PIVOT

6.5. Віконні функції

Віконні функції (window functions) було додано до стандарту SQL у 2003 році, щоб забезпечити більшу гнучкість, ніж дозволяє GROUP BY. Вікно – стандартний термін SQL, для опису контексту в якому працює функція. Вікно – це визначений користувачем набір рядків з таблиці, над якими виконуються певні обчислення. У певному сенсі вони схожі на групи, оскільки таблицю можна розбити на набір вікон. Однак, на відміну від груп, вікна не згортаються до рядка; в них можна оперувати, оскільки окремі рядки, що складають вікно, можна переглядати та маніпулювати ними так, як зручно.

Віконні функції застосовуються до набору рядків, що визначаються за допомогою речення OVER. Здебільшого їх використовують для аналітичних задач, для обчислення наростаючих сум і ковзаючих середніх, а також виконувати багато інших обчислення. Віконні функції дозволяють вирішувати багато завдань, дозволяючи виконувати обчислення в рамках наборів набагато простіше, інтуїтивно зрозуміліше і ефективніше. Загалом, віконні функції надають потужний спосіб виконання обчислень над підмножинами рядків, розширюючи аналітичні можливості SQL [10, 11, 15].

Синтаксис віконних функцій

```
<function>  OVER ([PARTITION BY clause]
        [ORDER BY clause]
        [ROWS or RANGE clause])
```

де

- *function* – агрегатна функція або функція ранжування;
- OVER – визначає вікно рядків, до яких буде застосовано функцію;
- PARTITION BY – розбиває результуючу множину на розділи для віконної функції;
- ORDER BY – визначає порядок рядків у кожному розділі;

* ROWS визначає рамку вікна, вказуючи діапазон рядків, які будуть включені в обчислення.

Приклади віконних функцій.

Функції ранжування. Функції ранжування присвоюють ранг або номер рядка кожному рядку в розділі на основі заданих критеріїв. Прикладами є ROW_NUMBER(), RANK() і DENSE_RANK(), які надають унікальні ідентифікатори або рейтинги для кожного рядка.

Агрегатні функції. Віконні версії агрегатних функцій (наприклад, SUM(), AVG() і COUNT()) обчислюють сукупні значення у вікні, а не для всього набору даних. Вони дозволяють обчислювати такі значення, як проміжний підсумок або ковзне середнє у вказаному вікні.

Аналітичні функції. Аналітичні функції надають додаткову інформацію про певний рядок, обчислюючи значення на основі значень у вікні. Прикладами є LAG(), LEAD() і FIRST_VALUE(), які отримують значення з попередніх або наступних рядків.

Процентильні функції. Процентильні функції (наприклад, PERCENT_RANK(), NTILE()) допомагають визначити відносну позицію рядка в розділі на основі відсортованих значень. Їх можна використовувати для виявлення розподілу даних і визначення процентилів.

Функції зсуву. Функції зсуву (наприклад, LAG(), LEAD()) отримують значення з попередніх або наступних рядків у розділі. Вони корисні для порівняння значень або обчислення різниці між послідовними рядками.

Розглянемо як працює віконна функція на прикладі таблиці медичних записів (рис. 6.15). Таблиця (а) зберігає дані візитів пацієнтів і вартість візиту. Використовуючи віконну функцію, отримаємо наростаючу вартість лікування (таблиця b).

```
SELECT patient_id, visit_date, cost,
SUM(cost) OVER (PARTITION BY patient_id ORDER BY visit_date) as SUMM
FROM medical_records;
```

medical_records (a)

patient_id	visit_dat e	cost
1	1/1/2020	100
1	1/2/2020	200
1	1/3/2020	150
2	2/1/2020	50
2	2/2/2020	75
3	3/1/2020	250

medical_records (b)

patient_id	visit_dat e	cost	SUMM
1	1/1/2020	100	100
1	1/2/2020	200	300
1	1/3/2020	150	450
2	2/1/2020	50	50
2	2/2/2020	75	125
3	3/1/2020	250	250

Рис. 6.15. Використання функції SUM() з вікном

Приклад. Використання віконної функції COUNT() для таблиці Співробітники (рис. 6.16):

Id	Name	Gender	Total	By gender
001	Андрій	M	1	**1**
002	Зінаїда	F	2	**1**
003	Микола	M	3	**2**
004	Ксенія	F	4	**2**
005	Богдан	M	5	**3**
006	Наталія	F	6	**3**
007	Василь	M	7	**4**
008	Анна	F	8	**4**
009	Софія	F	9	**5**
010	Павло	M	10	**5**
011	Петро	M	11	**6**
012	Віра	F	12	**6**
013	Семен	M	13	**7**

Рис. 6.16. Вхідні дані Співробітники

```
SELECT Id, Name, Gender,
 COUNT(*)OVER (ORDER BY Id) AS Total,
 COUNT(*)OVER (PARTITION BY Gender BY  Id) AS 'By gender'
FROM Employees;
```

В доданому стовпці Total наростаючий підсумок за стовпцем Id. В стовпці By gender наростаючий підсумок за статтю.

Приклад. Хто отримує найбільший відсоток заробітної плати у кожному підрозділі (рис. 6.17). Вікном будуть співробітники одного департаменту (рядки з однаковим значенням у колонці Department). Показник Max salary це загальна сума заробітної плати по кожному підрозділу, а показник dep_ratio визначає відсоток зарплати в межах підрозділу, що отримує працівник.

```
SELECT id, first_name, department, salary,
MAX(salary) OVER (PARTITION BY department) AS 'Max salary',
(Salary / SUM(Salary) OVER (PARTITION BY department) * 100, 2) AS 'dep_ratio'
FROM Salary;
```

Salary

Id	First_name	Department	Salary	Max_salary	dep_ratio
3	Микола	HR	150000	**150000**	**71**
7	Василь	HR	60000	**150000**	**29**
8	Анна	IT	98000	**300000**	**15**
4	Ксенія	IT	300000	**300000**	**47**
5	Богдан	IT	10000	**300000**	**2**
6	Наталія	IT	133000	**300000**	**21**
9	Софія	IT	98000	**300000**	**15**
2	Зінаїда	Marketing	100000	**100000**	**40**
11	Олександра	Marketing	50000	**100000**	**20**
10	Павло	Marketing	98000	**100000**	**40**
1	Андрій	Sales	10000	**98000**	**8**
12	Віра	Sales	98000	**98000**	**77**
13	Володимир	Sales	20000	**98000**	**15**

Рис. 6.17. Розрахунок показників по кожному підрозділу

Приклад. Розрахунок ковзного середнього. Завдання: обчислити 7-денне ковзне середнє цін на акції (рис. 6.18). Для обмеження розміру вікна використано ключове слово ROWS.

```
SELECT Date, StockPrice, AVG(StockPrice)
OVER (ORDER BY Date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS MovingAvg
FROM StockData;
```

Stock Data (a)

Date	Stock Price
01.01.2022	10,00
01.02.2022	12,00
01.03.2022	15,00
01.04.2022	16,00
01.05.2022	14,00
01.06.2022	13,00
01.07.2022	12,50
01.08.2022	13,00

Stock Data (b)

Date	Stock Price	Moving Avg
01.01.2022	10,00	10,00
01.02.2022	12,00	11,00
01.03.2022	15,00	12,33
01.04.2022	16,00	13,25
01.05.2022	14,00	13,40
01.06.2022	13,00	13,50
01.07.2022	12,50	13,43
01.08.2022	13,00	13,25

Рис. 6.18. Результат виконання запиту

Приклад. Використання функцій зсуву `LAG()` і `LEAD()` (рис. 6.19). Функція `LAG(`*`column, offset, default`*`)` повертає дані із попередніх рядків колонки *column* зі зсувом *offset*. Функція `LEAD(`*`column, offset, default`*`)` повертає дані з наступних рядків колонки *column* зі зсувом *offset.* Атрибут *default* визначає значення, що поверне функція, якщо значення рядка Null.

```
SELECT Month, Sales,
Lag(Sales, 1, 0) OVER (ORDER BY Month) AS PrevMonthSales,
Lead(Sales, 1, 0) OVER (ORDER BY Month) AS NextMonthSales
FROM Sales;
```

Sales

Month	Sales	PrevMonthSales	NextMonthSales
Jan	100	0	150
Feb	150	100	200
Mar	200	150	175
Apr	175	200	225
May	225	175	0

Рис. 6.19. Результат виконання запиту

Контрольні запитання

1. В чому різниця між функціями ранжування `RANK()` і `DENSE_RANK()`?
2. Що повертає функція `ROW_NUMBER()`?
3. Що повертає функція `PERCENT_RANK()`?
4. Які функції використовуються для обчислення проміжних підсумків в агрегованих даних?
5. В чому призначення функції `ROLLUP()`?
6. В чому призначення функції `CUBE()`?
7. В чому призначення функції `PIVOT()`?
8. Що в SQL означає термін Вікно (Window)

9. Для чого в віконних функціях використовується речення `PARTITION BY`?
10. Для чого в віконних функціях використовується речення `ROWS`?
11. Що таке віконні функції в SQL і чим вони відрізняються від звичайних агрегатних функцій?
12. Поясніть на прикладі речення `OVER` у віконних функціях.
13. Яке призначення речення `ORDER BY` у віконних функціях?
14. Які існують функції ранжування?
15. Поясніть концепцію рекурсивних CTE в SQL.

Тестові завдання

1. Яке з наступних тверджень не є правильним?

   A. SQL-запит не повинен мати більше одного з’єднання в одному запиті
   B. Всі імена стовпців в SQL-запиті повинні бути уточнені іменем таблиці
   C. Всі імена таблиць, вказані в реченні SQL FROM, повинні бути різними
   D. SQL-запит не повинен мати функції агрегації в реченні HAVING
   E. Все перераховане вище

2. SQL надає ряд спеціальних агрегатних функцій. Яка з наведених нижче функцій не входить до складу SQL?

   `A. COUNT`
   `B. MEDIAN`
   `C. SUM`
   `D. MAX`
   `E. MIN`

3. Яке з наступних тверджень не є правильним при використанні функцій в операторах SQL?

   A. `DISTINCT` потрібно використовувати, якщо перед застосуванням функції потрібно видалити дублікати
   B. Всі випадки `COUNT` повинні включати `DISTINCT`
   C. Будь-які Null у стовпчику аргументів ігноруються, коли використовуються такі функції, як `AVG`, `MAX` і `MIN`
   D. Якщо аргумент є порожньою множиною, `COUNT` повертає нуль, тоді як інші функції повертають Null

4. Яке з наведених нижче тверджень є неправильним?

   A. `GROUP BY` логічно розбиває таблицю на частини; фізичного перегрупування не відбувається
   B. Кожен вираз у реченні `SELECT` (крім агрегатних функцій) повинен бути єдиним значенням для кожної групи, коли використовується `GROUP BY`
   C. Не можна використовувати WHERE, а потім використовувати `GROUP BY` в одному запиті

   D. HAVING для GROUP BY використовується так само, як WHERE для SELECT
   E. Речення WHERE вибирає рядки, які задовольняють задану умову, а речення HAVING вибирає групу, яка задовольняє умову, використовуючи агрегування
5. Яке з наступних тверджень невірне?
   A. Запит може мати до шести або більше різних формулювань на мові SQL
   B. Різні версії одного і того ж запиту гарантовано обробляються однаково ефективно
   C. Підзапити можуть бути вкладені на будь-яку глибину
   D. SQL не є точною реалізацією реляційної моделі

В наступних тестах використовується база даних Сесія такої структури

```
students(student_id, student_name, address)
results(student_id, subject_id, mark)
subjects(subject_id, subject_name)
```

6. Що є вірним для наступного запиту

```
SELECT subject_name, COUNT(*)
FROM subjects, results
WHERE subject.subject_id = results.subject_id AND mark > 60
GROUP BY subject_name
HAVING COUNT(*) > 5;
```

   A. Запит є неприпустимим
   B. Цей запит повертає назви предметів і учнів . які мають більше п'яти оцінок, які більше 60
   C. Запит повертає назви предметів і кількість учнів у кожному предметі, який має більше п'яти учнів
   D. Запит повертає назви предметів і кількість студентів у кожному предметі, який має більше п'яти студентів, кожен з яких має оцінку більше 60
7. Що є вірним для наступного запиту

```
SELECT student_name, AVG(mark)
FROM students, results
WHERE students.student_id = results.student_id;
```

   A. Запит не є коректним
   B. Запит повертає імена студентів та їхні середні бали
   C. Запит повертає імена студентів та загальний середній бал для кожного рядка
   D. Запит повертає імена студентів та оцінки з кожного предмету

8. Що є вірним для наступного запиту:

```
SELECT student_name, mark
FROM results, students
WHERE (results.student_id = students.student_id)
AND mark > 60));
```

   A. Запит не є коректним
   B. Запит повертає імена студентів та їхні оцінки

C. Запит повертає імена студентів що мають оцінку > 60 по всім дисциплінам
D. Запит повертає імена студентів що мають оцінку > 60 хоча б з однієї дисципліни

9. Що є вірним для наступного запиту

```
SELECT student_name, subject_name, AVG(mark)
FROM subjects, results
WHERE subjects.subject_id = results.subject_id
GROUP BY subject_id
HAVING AVG(mark) >80;
```

A. Запит не є коректним
B. Запит повертає ім'я студента, назву предмету та середній бал з цього предмету
C. Запит повертає ім'я студента, назву предмета і середній бал студента
D. Запит повертає ім'я студента, назву предмета і середній бал тільки для тих предметів, де середній бал більше 80

10. Що є вірним для наступного запиту

```
SELECT subject_id
FROM subjects, results
WHERE subjects.subject_id = results.subject_id
GROUP BY subject_id
HAVING AVG(mark) >80;
```

A. Запит не є коректним
B. Запит повертає ідентифікатори всіх предметів
C. Запит повертає ідентифікатори предметів, які мають середній бал більше 80
D. Запит не може бути написаний без з'єднання або підзапиту

11. Що є вірним для наступного запиту

```
SELECT subject_name
FROM subjects, results
GROUP BY subjects_id;
```

A. Запит не є коректним
B. Запит повертає назви всіх предметів
C. Запит повертає по одному разу назви всіх предметів, які здавали студенти
D. Запит не може бути сформульовано без з'єднання або підзапиту
E. Нічого з перерахованого вище

12. Що є вірним для наступного запиту

```
SELECT subject_name
FROM subjects, results
WHERE subjects.subject_id = results.subject_id
GROUP BY subject_name
HAVING mark > 80;
```

A. Запит не є коректним
B. Запит повертає назви всіх предметів
C. Запит повертає назви предметів, які мають середній бал більше 80
D. Запит повертає назви предметів, у яких деяка оцінка перевищує 80
E. Нічого з перерахованого вище

13. Що є вірним для наступного запиту

```
SELECT student_name
```

```sql
FROM students, results
WHERE students.student_id = results.student_id
AND mark IS NULL;
```

A. Запит не є коректним
B. Запит повертає прізвища всіх студентів, які не мають жодної оцінки з усіх предметів
C. Запит повертає прізвище кожного студента, який не має жодної оцінки з деяких предметів
D. Запит повертає ім'я студента кожного разу, коли він не має оцінки з якогось предмета
E. Наведений вище запит не повертає імена студентів, що дублюються

14. Що є вірним для наступного запиту

```sql
SELECT subject_name
FROM subjects, results
GROUP BY subject_id
HAVING MAX(AVG(mark)) < 60;
```

A. Запит не є коректним
B. Запит повертає назви предметів, максимальний бал з яких менше 60
C. За запитом знайдено назви предметів, максимальний середній бал яких менше 60
D. За запитом будуть знайдені назви предметів, середній бал яких є максимальним з усіх предметів і становить менше 60

15. Що є вірним для наступного запиту

```sql
SELECT subject_name
FROM subject
WHERE subject_id IN
    (SELECT subject_id
     FROM enrolment
     WHERE student_id IN
         (SELECT student_id
          FROM student
          WHERE student_name = 'Іван Відмінник'));
```

A. Запит не є коректним
B. Запит повертає назви всіх дисциплін, які складав Іван Відмінник
C. Запит повертає назви всіх дисциплін, які не складав Іван Відмінник
D. Нічого з перерахованого

7. ФІЗИЧНА РЕАЛІЗАЦІЯ БАЗИ ДАНИХ

7.1. Організація пам'яті

Фізичне проєктування бази даних зводиться до перекладу логічної моделі даних у внутрішню модель даних, яку також називають фізичною моделлю даних. При цьому враховуються фізичні властивості носіїв інформації, а також статистичні властивості даних і типи операцій (пошук, вставка, оновлення, видалення), які виконуються над ними. Внутрішня модель даних повинна забезпечувати адекватну підтримку найчастіших та/або найбільш критичних за часом операцій.

Пам'ять комп'ютерної системи можна розглядати як ієрархію (рис. 7.1), з високошвидкісною пам'яттю, яка є дуже дорогою і обмеженою за обсягом, на вершині, і повільнішою пам'яттю, яка є відносно дешевою і набагато більшою за розміром, внизу [3]. На вершині ієрархії знаходиться центральний процесор (ЦП) з його регістрами, в яких виконуються математичні та логічні операції процесора. Найчастіше деяка високошвидкісна кеш-пам'ять фізично інтегрована з центральним процесором та/або з материнською платою, яка містить центральний процесор. Кеш-пам'ять працює майже на тій самій швидкості, що й центральний процесор. Нижче знаходиться центральна пам'ять, яку також називають оперативною пам'яттю. Вона складається з мікросхем пам'яті, продуктивність яких виражається в наносекундах. Кожен окремий байт у центральній пам'яті має власну адресу, і до нього безпосередньо звертається операційна система. Вся пам'ять, описана вище, називається первинною пам'яттю. Цей тип пам'яті вважається енергозалежною пам'яттю, тобто її вміст очищується при вимкненні живлення. Безумовно, оперативна пам'ять відіграє важливу роль у системі баз даних, оскільки вона містить буфер бази даних, а також виконавчий код додатків і СКБД.

Вторинна пам'ять, яка складається з постійних носіїв, що зберігають свій вміст навіть без живлення. Фізичні файли бази даних знаходяться у вторинній пам'яті. Найважливішим пристроєм вторинної пам'яті все ще є жорсткий диск (HDD), хоча твердотільні накопичувачі (SSD) на основі флеш-пам'яті швидко наздоганяють його.

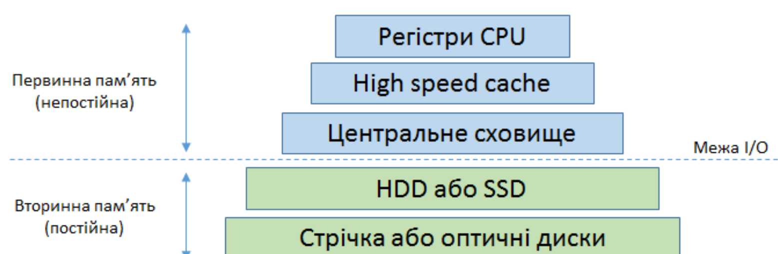


Рис. 7.1. Ієрархія пам'яті

Первинна і вторинна пам'ять розділені так званою межею вводу/виводу. Це означає, що вся пам'ять вище цієї межі, хоч і працює повільніше, ніж процесор, але все ж працює зі швидкістю, яка дозволяє процесору ефективно «чекати», поки дані будуть отримані з первинної пам'яті. Вторинна пам'ять працює набагато повільніше. Середній

час обробки запису в первинної пам’яті – 100 наносекунд; тобто ефективність фізичної організації бази даних багато в чому визначається мінімізацією взаємодії з вторинною пам’яттю.

Для прикладу розглянемо диск зі швидкістю обертання 10 000 об/хв, середнім часом пошуку 5 мс і швидкістю передачі 83 МБ/с. Ось деякі розрахунки витрат часу [14]:

- середня затримка обертання – 3 мс,
- час передачі 1 Б – 0,000012 мс,
- час передачі 1000 Б – 0,012 мс,
- орієнтовний час доступу до 1 Б – 8,000012 мс,
- розрахунковий час доступу до 1000 Б – 8,012 мс.

Розрахунковий час доступу до 1000 байтів практично такий самий, як і до 1 байта. Іншими словами, немає сенсу отримувати доступ до декількох байтів з диска. Сучасні диски за один раз обробляють цілий сектор. Типовий розмір сектора – 512 Б. Файл бази даних розміщується на сторінках розміром сектор або більше. Ефективність читання/запису даних визначається в кількості сторінок в одиницю часу. Тому основне завдання СКБД є раціональна організація файлу на сторінках вторинної пам’яті.

7.2. Організація вторинної пам’яті

Незалежно від типу файлової структури, зберігання даних у вторинної пам’яті у відомих СКБД (Oracle, IBM DB2, Microsoft SQL Server) організовано дуже схожим чином. Основними одиницями фізичного зберігання є сторінка даних (page), що є одиницею обміну із зовнішньою пам’яттю. Розмір сторінки є параметром СКБД, типові значення – 4 КБ або 8 КБ. Вартість сторінкового вводу/виводу домінує над вартістю типових операцій з базами даних, і системи баз даних ретельно оптимізуються для мінімізації цієї вартості [14].

Розмір сторінки має великий вплив на продуктивність бази даних – за великих розмірів швидкість операцій читання/запису зростає (особливо це характерно для повних переглядів таблиць і операцій інтенсивного завантаження даних), однак зростають накладні витрати на зберігання (база збільшується) і знижується ефективність пошуків за індексом. Менший розмір сторінки дає змогу економніше витрачати пам’ять, але водночас збільшується кількість звернень до диску. Довгі сторінки (16, 32 або 64 КБ) краще використовувати для великих об’єктів даних: повнотекстові фрагменти, мультимедіа-об’єкти, довгі рядки тощо. Короткі сторінки (2 або 4 КБ) краще підходять для значень числових типів, не довгих рядків, значень дати і часу. Слід також враховувати розмір блоку ОС, він має бути кратний розміру сторінки бази даних.

7.2.1. Роль диспетчера записів

СКБД взаємодіє з операційною системою через проміжне програмне забезпечення, що має назву диспетчер записів. Саме диспетчер записів визначає фізичну структуру бази даних. Для оптимальної організації сторінок диспетчер записів повинен вирішити такі питання, як [14]:

- Чи повинен кожен запис бути повністю розміщений на однієї сторінці?
- Чи всі записи на сторінці будуть з однієї таблиці?
- Чи кожне поле може бути представлене заздалегідь визначеною кількістю байтів?
- Де слід розміщувати значення кожного поля у записі?

Розділені та нерозділені записи

Коли записи додаються на сторінку, може виникнути ситуація, коли на сторінці ще є вільне місце, але для чергового запису місця не вистачає. В такому випадку диспетчер записів може або зберегти такій запис на нової сторінці, або розділити запис на дві частини і одну частину записати на першої сторінці, а залишок на другій (рис. 7.2).

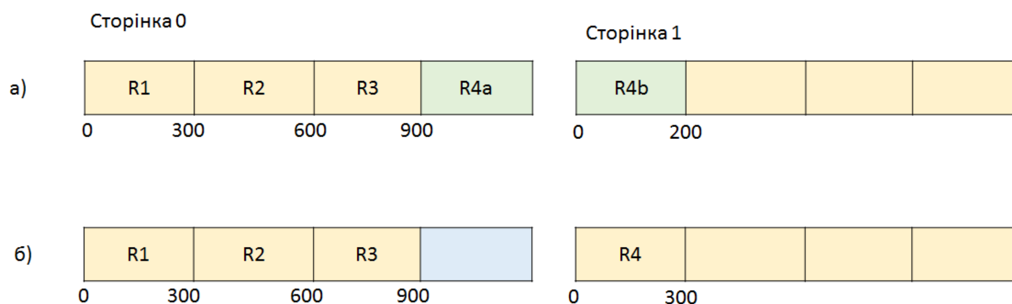


Рис. 7.2. Розділені та нерозділені записи: (a) Запис R4 збережено на сторінках 0 та 1; (b) запис R4 повністю зберігається на сторінці 1

Недоліком нерозділених записів є те, що вони марно витрачають місце на диску. Іншим недоліком є те, що розмір нерозміщеного запису обмежується розміром блоку. Якщо записи можуть бути більшими за блок, то необхідне розбиття на частини.

Основним недоліком розділених записів є те, що вони збільшують складність доступу до записів. Оскільки розділені запис розбивається на декілька сторінок, для його читання потрібно декілька звернень. Крім того, може виникнути потреба у відновленні запису з цих сторінок шляхом зчитування його в окрему область пам'яті.

Однорідні та неоднорідні файли

Файл є однорідним, якщо всі його записи походять з однієї таблиці. Диспетчер записів повинен вирішити, чи дозволяти створювати неоднорідні файли, чи ні. Компроміс полягає у виборі між ефективністю та гнучкістю.

Однорідна реалізація передбачає розміщення усіх записів однієї таблиці в одному файлі, а всі записи другої – в іншому. Таке розміщення спрощує відповіді на однотабличні SQL-запити – менеджеру записів потрібно сканувати лише сторінки одного файлу. Однак багатотабличні запити стають менш ефективними. Менеджеру записів доводиться здійснювати пошук між сторінками записів різних таблиць, шукаючи відповідні записи.

Неоднорідна організація зберігає записи обох таблиць в одному файлі, причому пов'язані записи обох таблиць зберігається поруч. Така організація вимагає менше звернень до сторінок для обчислення з'єднання, оскільки з'єднані записи знаходяться в одній сторінці або в сусідніх сторінках (рис. 7.3).

Students

Sid	SName	GradYear	DeptId
1	Олександр	2021	10
2	Ганна	2020	20
3	Микола	2022	10
4	Петро	2022	20
5	Андрій	2020	30
6	Олена	2020	20
7	Наталя	2021	10

Departments

Did	DName
10	СТММК
20	ММЗІ
30	ММАД

Сторінка 0

10 СТММК	1 Олександр 2021 10	3 Микола 2022 10

Сторінка 1

7 Наталя 2021 10	20 ММЗІ	2 Ганна 2020 20

Рис. 7.3. Кластерізоване зберігання даних

Кластеризація підвищує ефективність запитів, які об'єднують кластеризовані таблиці, оскільки відповідні записи зберігаються разом. Однак кластеризація призводить до того, що запити до однієї таблиці стають менш ефективними, оскільки записи для кожної таблиці розподіляються по більшій кількості блоків. Аналогічно, з'єднання з іншими таблицями також будуть менш ефективними. Таким чином, кластеризація ефективна лише тоді, коли найбільш часто використовувані запити потребують з'єднання кластерізованих записів.

Поля фіксованої і змінної довжини

Кожне поле в таблиці має визначений тип. На основі цього типу диспетчер записів вирішує, чи реалізовувати поле за допомогою структури фіксованої або змінної довжини. Структури фіксованої довжини використовують однакову кількість байтів для зберігання кожного значення поля, тоді як структури змінної довжини розширюються і стискаються залежно від значення даних, що зберігаються.

Більшість типів від природи мають фіксовану довжину. Наприклад, як цілі числа, так і числа з плаваючою комою можна зберігати у вигляді 4-байтових двійкових значень. Фактично, всі числові типи та типи дати/часу мають природне представлення фіксованої довжини.

Тип даних varchar є прикладом типу, який потребує структури змінної довжини. Зберігання в структурах змінної довжини може спричинити значні ускладнення. Наприклад, зміна такого запис може призвести до збільшення розміру запису. Тоді диспетчер записів може змінити розташування записів на сторінці. Тому, диспетчер записів намагається використовувати представлення фіксованої довжини, коли це можливо. Наприклад, диспетчер записів може вибирати з трьох різних представлень рядкового поля змінної довжини:

- Представлення змінної довжини, у якому диспетчер записів виділяє саме стільки місця у записі, скільки потрібно для рядка.
- Представлення фіксованої довжини, у якому менеджер записів зберігає рядок у області поза записом і зберігає посилання на це місце у запису фіксованої довжини. Ця область може бути окремим файлом. У будь-якому випадку, поле містить посилання на розташування рядка у цій області. Результатом такого представлення

є записи фіксованої довжини і невеликі за розміром. Недоліком такого представлення є те, що отримання значення рядка із запису вимагає додаткового доступу до диску.

- Представлення фіксованої довжини, у якому менеджер записів виділяє однакову кількість місця у записі для кожного рядка, незалежно від його довжини. Перевагою такої реалізації є те, що записи мають фіксовану довжину, а рядки зберігаються у записі. Однак недоліком є те, що деякі записи будуть більшими, ніж більшість інших. Якщо існує велика різниця у розмірах рядків, то цей марно витрачений простір буде значним, що призведе до більшого розміру файлу і, відповідно, до більшої кількості звернень до сторінок.

Жоден з цих способів не є однозначно кращим за інші. Щоб допомогти диспетчеру записів вибрати правильне представлення, в стандарті SQL передбачено три різні типи рядкових даних: char, varchar і clob. Тип char(n) визначає рядки, що складаються рівно з n символів. Типи varchar(n) і clob(n) визначають рядки, що містять не більше n символів. Різниця між ними полягає в очікуваному розмірі n. У varchar n досить мале, скажімо, не більше 4K. З іншого боку, значення n у clob(n)[4] може бути в діапазоні гігабайтів.

Поля типу char найбільш природно відповідають варіанту (с). Оскільки всі рядки будуть однакової довжини, всередині записів не буде зайвого простору, і представлення з фіксованою довжиною буде найефективнішим.

Поля типу varchar(n) найбільш природно відповідають варіанту (а). Оскільки n буде відносно невеликим, розміщення рядка всередині запису не призведе до того, що запис стане занадто великим. Більше того, різниця у розмірах рядків означає, що використання структур з фіксованою довжиною призведе до зайвих витрат місця.

Якщо n виявляється малим (скажімо, менше 20), то диспетчер записів може вирішити реалізувати поле varchar, використовуючи варіант (c). Причина полягає у тому, що втрачений простір буде незначним у порівнянні з перевагами представлення фіксованої довжини.

Для збереження записів типу clob використовується варіант (b), оскільки це представлення найкраще обробляє великі рядки. Зберігаючи великий рядок поза записом, самі записи стають меншими і більш керованими.

Розміщення полів у записах

Диспетчер записів визначає структуру записів. Для записів фіксованої довжини він визначає розташування кожного поля у записі. Найпростіша стратегія полягає у зберіганні полів поруч одне з одним. Розмір запису тоді стає сумою розмірів полів, а зміщенням кожного поля є кінець попереднього поля.

Проблема пов’язана з забезпеченням правильного вирівнювання значень у пам’яті. У більшості комп’ютерів машинний код доступу до цілого числа вимагає, щоб ціле число зберігалося в комірці пам’яті, кратній 4; кажуть, що ціле число вирівняне на межі 4-х байт. Таким чином, менеджер записів повинен переконатися, що кожне ціле число на кожній сторінці вирівняне за 4-байтовою межею. Оскільки сторінки ОС завжди вирівнюються за

[4] Абревіатура CLOB розшифровується як «символьний об’єкт великого розміру»

2N-байтовою межею для деякого достатньо великого N, перший байт кожної сторінки буде вирівняно належним чином. Таким чином, диспетчер записів повинен просто переконатися, що зміщення кожного цілого числа на кожній сторінці кратне 4. Якщо попереднє поле закінчувалося у місці, яке не є кратним 4, то диспетчер записів повинен доповнити його достатньою кількістю байтів, щоб воно стало таким.

Загалом, різні типи можуть вимагати різної кількості додаткових байтів. Наприклад, числа з плаваючою комою подвійної точності зазвичай вирівнюються на межі 8 байтів, а невеликі цілі числа – на межі 2 байти. Менеджер записів відповідає за забезпечення вирівнювання. Проста стратегія полягає у розташуванні полів у тому порядку, в якому вони були оголошені, з вирівнюванням кожного поля, щоб забезпечити правильне вирівнювання наступного поля. Більш складна стратегія полягає в тому, щоб змінити порядок полів так, щоб вимагати найменшу кількість пропусків.

7.2.2. Реалізація файлу записів

У попередньому розділі розглядалися різні рішення, які повинен приймати диспетчер записів. У цьому розділі ми розглянемо, як ці рішення реалізуються.

Розміщення записів фіксованої довжини

Той факт, що записи однорідні і мають фіксовану довжину, означає, що можна виділити простір однакового розміру для кожного запису на сторінці.

Диспетчер записів ділить сторінку на слоти, де кожен слот достатньо великий, щоб вмістити запис плюс один додатковий байт. Значення цього байта є прапором, який вказує на те, чи слот порожній, чи використовується. Скажімо, 0 означає «порожній», а 1 – «використовується» (рис. 7.4).

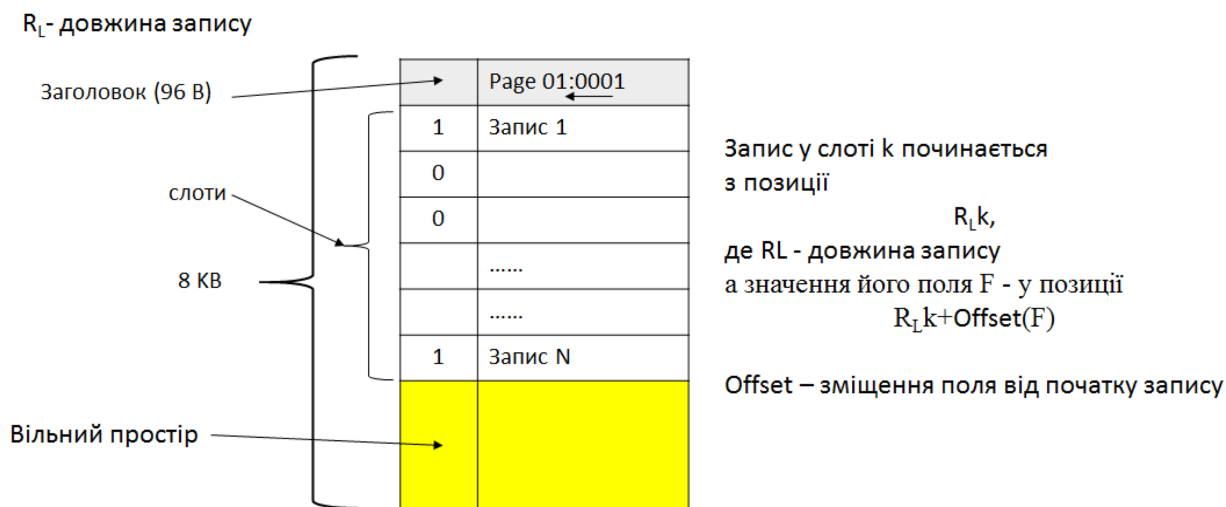


Рис. 7.4. Розміщення записів фіксованої довжини на сторінці

Диспетчер записів повинен мати можливість вставляти, видаляти та змінювати записи на сторінці запису. Для цього він використовує наступну інформацію про записи:

- розмір слоту,
- назва, тип, довжина і зміщення (**Offset**) кожного поля запису від її початку.

Ці значення складають схему запису. Маючи схему, диспетчер записів може визначити місце розташування кожного значення на сторінці. Запис у слоті **k** починається з позиції $R_L k$, де R_L – довжина запису. Прапорець порожнього/використання для цього запису знаходиться у позиції $R_L k$, а значення поля **F** запису – у позиції $R_L k + Offset(F)$. Диспетчер записів може досить легко обробляти вставки, видалення, модифікації та вилучення записів:

- Щоб вставити новий запис, диспетчер записів перевіряє прапори порожнього/зайнятого кожного слоту, доки не знайде 0. Потім він заповнює запис, встановлює прапор у 1 і повертає розташування цього слоту. Якщо всі значення прапорів дорівнюють 1, то блок заповнений і вставка неможлива.
- Щоб видалити запис, диспетчер записів просто встановлює його прапорець у 0.
- Щоб змінити значення поля запису, диспетчер записів визначає місцезнаходження цього поля і записує значення у це місце.
- Щоб отримати записи на сторінці, диспетчер записів перевіряє прапори кожного слоту. Кожного разу, коли він знаходить 1, він знає, що цей слот містить існуючий запис.

Диспетчеру записів також потрібен спосіб ідентифікувати запис у межах сторінки записів. Коли записи мають фіксовану довжину, найпростішим ідентифікатором запису є номер слоту.

Реалізація полів змінної довжини

В випадку полів змінної довжини зміщення всіх полів, наступних за полем змінної довжини, будуть відрізнятися від запису до запису. Єдиний спосіб визначити зміщення цих полів – це прочитати попереднє поле і подивитися, де воно закінчується. Якщо перше поле у записі має змінну довжину, то для визначення зсуву **n**-го поля потрібно прочитати перші **n-1** полів запису. Тому диспетчер записів зазвичай розміщує поля фіксованої довжини на початку кожного запису, щоб до них можна було отримати доступ за попередньо обчисленим зсувом. Поля змінної довжини розміщуються в кінці запису. Перше поле змінної довжини матиме фіксоване зміщення, а решта – ні.

Інша проблема полягає в тому, що зміна значення поля може призвести до зміни довжини запису. Якщо нове значення більше, то вміст блоку праворуч від зміненого значення має бути зсунутий, щоб звільнити місце. У крайньому випадку, зсунуті записи можуть зміститися за межі сторінки. Цю ситуацію вирішують шляхом виділення блоку переповнення. Блок переповнення – це нова сторінка, виділена в області, що має назву область переповнення. Будь-який запис, що виходить за межі початкової сторінки, видаляється з цієї сторінки і додається до блоку переповнення. Якщо таких модифікацій відбувається багато, то може знадобитися ланцюжок з декількох блоків переповнення. Кожен блок буде містити посилання на наступний блок переповнення у ланцюжку. Концептуально, початковий блок і блок переповнення утворюють одну (велику) сторінку запису.

Третя проблема стосується використання номера слоту як ідентифікатора запису. Неможливо помножити номер слоту на розмір слоту, як у випадку записів фіксованої

довжини. Єдиний спосіб знайти початок запису із заданим ідентифікатором – це прочитати записи, починаючи з початку блоку.

Вирішенням цієї проблеми є використання таблиці ідентифікаторів для відокремлення номера слоту запису від його розташування на сторінці. Таблиця ідентифікаторів – це масив цілих чисел, що зберігається на початку сторінки. Кожен слот у масиві позначає ідентифікатор запису. Значення у комірці масиву вказує на місцезнаходження запису з таким ідентифікатором; значення 0 означає, що наразі жоден запис не має такого ідентифікатора. На рис. 7.5 зображено розміщення даних з таблицею ідентифікаторів. Таблиця ідентифікаторів містить вказівники на записи і їх розмір.

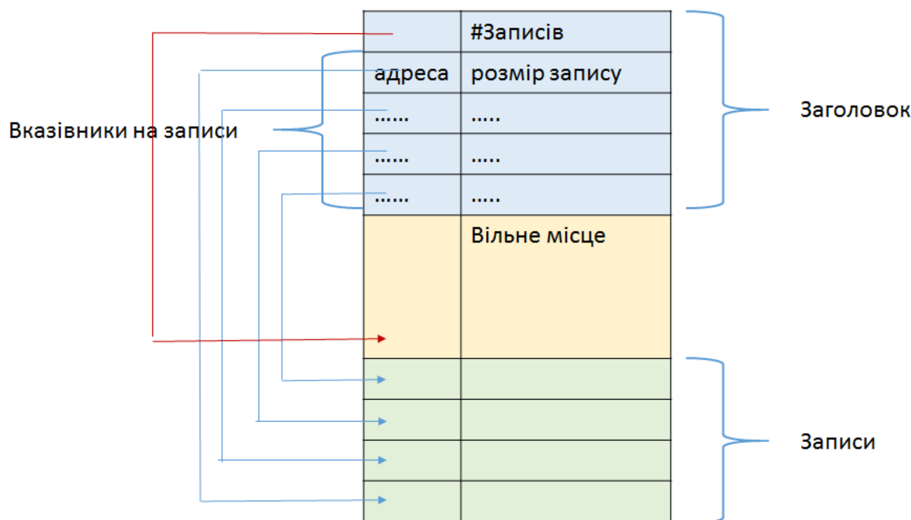


Рис. 7.5. Розміщення на сторінці записів змінної довжини

Таблиця ідентифікаторів забезпечує рівень непрямого зв’язку, який дозволяє диспетчеру записів переміщувати записи в межах сторінки. Якщо запис переміщується, його запис у таблиці ідентифікаторів відповідно змінюється; якщо запис видаляється, його вказівник встановлюється на 0. Коли вставляється новий запис, диспетчер записів знаходить доступний запис у таблиці ідентифікаторів і призначає його як ідентифікатор нового запису. Таким чином, таблиця ідентифікаторів дозволяє переміщувати записи змінної довжини в межах сторінки, надаючи кожному запису фіксований ідентифікатор.

Таблиця ідентифікаторів розширюється зі збільшенням кількості записів у блоці. Розмір масиву не фіксований, оскільки сторінка може містити різну кількість записів змінної довжини. Зазвичай таблиця ідентифікаторів розміщується на одному кінці сторінки, а записи – на іншому, і вони зростають одна до одної.

Таблиця ідентифікаторів робить непотрібними прапорці. Запис використовується, якщо на нього вказує запис у таблиці ідентифікаторів. Порожні записи мають ідентифікатор 0 (і фактично навіть не існують). Таблиця ідентифікаторів також дозволяє диспетчеру записів швидко знаходити кожен запис на сторінці. Щоб перейти до запису з певним ідентифікатором, диспетчер записів просто використовує місцезнаходження, що зберігається у цьому записі таблиці ідентифікаторів. Щоб перейти до наступного запису, диспетчер записів сканує таблицю ідентифікаторів, доки не знайде наступний ненульовий запис.

7.2.3. Організація записів у файлах

Досі ми вивчали, як записи представлені у файловій структурі. Відношення – це набір записів. Маючи набір записів, наступне питання полягає в тому, як організувати їх у файлі. Існує декілька можливих способів організації записів у файлах:

На рис. 7.6 наведено популярні типи організації файлів:

Рис. 7.6. Типи організації файлів

Організація файлів купи (Heap File)

Структура файлу, в якому записи можуть бути розміщені будь-де у цій області пам'яті має назву «купа» (Heap File). В такому файлі не підтримується жодного впорядкування, послідовності або індексування. Відповідальність за управлінням записами покладається на програмне забезпечення. Нові записи вставляються в кінець файлу і немає ніякого зв'язку між атрибутами запису та його фізичним розташуванням. Отже, додавання записів є досить ефективним, але пошук конкретного запису або набору записів за пошуковим ключем – ні. Єдиний варіант – виконати лінійний пошук, скануючи весь файл, і зберегти записи, які відповідають критеріям відбору. Якщо шукається один запис за первинним ключем, сканування триває доти, доки запис не буде знайдено, або доки не буде досягнуто кінця файлу, що означатиме, що такого запису немає. В середньому потрібно N/2 послідовних звернень до блоків де N кількість записів. Проте, чим більше запитів відбувається для записів, яких немає у файлі, тим більше пошуків буде відбуватися до кінця файлу, що вимагатиме N звернень до блоків. Пошук записів за неунікальним пошуковим ключем також вимагає сканування всього файлу, а отже, блочного доступу до файлу. Видалення запису часто зводиться до простого позначення його як «видаленого». Потім записи фізично видаляються при періодичній реорганізації файлу.

Послідовна організація файлів

У випадку послідовної організації файлів записи розміщуються у файлі в певному послідовному порядку на основі унікального ключового поля або ключа пошуку в порядку зростання або спадання пошукового ключа. Найчастіше це первинний ключ, але як критерій упорядкування можна використовувати й неунікальний пошуковий ключ (наприклад, неключовий атрибут або набір атрибутів). Перевагою цього методу є те, що пошук записів у порядку, визначеному цим ключем, стає набагато ефективнішим. Якщо послідовний файл зберігається на накопичувачі з прямим доступом, наприклад, на

жорсткому диску, можна використовувати бінарний пошук, який у випадку великих файлів є набагато ефективнішим, ніж лінійний пошук. Алгоритм бінарного пошуку застосовується рекурсивно, зменшуючи вдвічі інтервал пошуку з кожною ітерацією. Для унікального ключа пошуку K зі значеннями K_j алгоритм пошуку запису зі значенням ключа K_μ виглядає наступним чином (рис. 7.7):

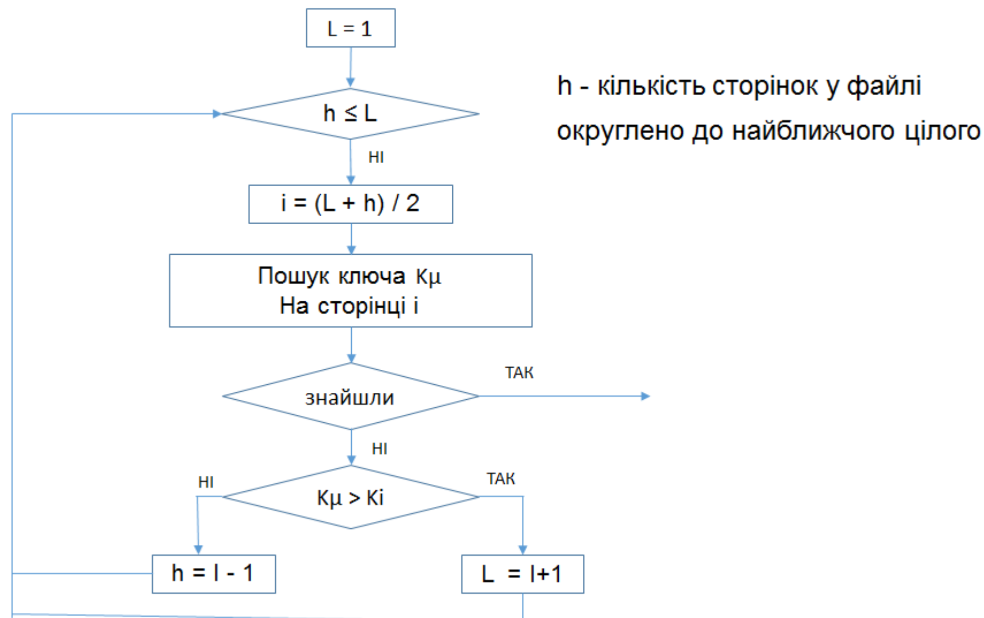


Рис. 7.7. Алгоритм бінарного пошуку в послідовному файлі

Якщо використовується бінарний пошук, то очікуване число становить $\log_2(N)$ випадкових звернень до сторінок, що є набагато ефективнішим для великих значень N.

Кластерна організація файлів

У цьому механізмі пов'язані записи з одного або декількох таблиць зберігаються в одному блоці на диску, тобто впорядкування записів не ґрунтується на первинному ключі або ключі пошуку. Така організація допомагає легко отримувати дані на основі певної умови з'єднання. Зазвичай для зберігання записів кожної таблиці використовується окремий файл або набір файлів. Однак у багатофайловій кластерній файловій організації записи кількох різних таблиць зберігаються в одному файлі, а фактично в одному блоці у файлі, щоб зменшити витрати на певні операції з'єднання.

Кластерна організація файлів не вважається хорошою для великих баз даних.

Організація хеш-файлів

Цей механізм використовує обчислення хеш-функції над деякими полями записів. Як ми знаємо, файл – це набір записів, який має бути відображений на деякий блок виділеного для нього дискового простору. Це відображення визначається обчисленням хешу за допомогою хеш-функції. Результат обчислення хешу визначає місце розташування блоку на диску, де можуть знаходитись записи. Хеш-функція обчислюється над деяким атрибутом кожного запису. Результат хеш-функції вказує, в якому блоці файлу слід розмістити запис.

Основним недоліком послідовної організації файлів є те, що для пошуку одного потрібного запису потрібно звертатися до багатьох інших записів. Цю проблему дещо

полегшує двійковий пошук, але навіть у цьому випадку кількість непотрібних записів може стати досить великою. При організації хеш-файлів існує прямий зв'язок між значенням пошукового ключа і фізичним розташуванням запису. Таким чином, запис може бути знайдений за допомогою одного або, щонайбільше, кількох звернень, якщо вказано значення ключа.

7.3. Індексування

7.3.1. Загальні відомості

У термінології баз даних індексування – це процес створення структури даних, яка дозволяє швидко отримувати записи з файлу бази даних. Індекс – це структура даних, яка встановлює відповідності значення атрибута, що індексується і місце розташування запису з таким атрибутом. Значення індексованого атрибута упорядковуються (найчастіше, по зростанню). Індекс зазвичай зберігається в окремому файлі або окремої області пам'яті. Індекси є структурою, яка повністю відноситься до внутрішньої схеми і, отже, не можуть бути видимі в логічній структурі БД.

Індекси мають вирішальне значення для ефективної обробки запитів у базах даних. Без індексів кожен запит зчитував би весь вміст кожної таблиці. Це було б невиправдано дорого для запитів, які обирають лише кілька записів. Індекси широко використовуються при проєктуванні баз даних через такі переваги:

- розмір індексів значно менший за розмір індексованих даних, що дає змогу розміщувати їх в оперативній пам'яті для прискорення доступу;
- структура індексів спеціальним чином оптимізована для виконання операцій пошуку;.

Планування і створення індексів вимагає врахування великої кількості різноманітних чинників, зокрема особливостей СКБД, апаратури, на якій працює сервер баз даних, і, головне, характеру запитів, що виконуються. Далеко не у всіх випадках індекси призводять до скорочення кількості ресурсів, необхідних для виконання запитів, а в деяких випадках наявність індексів призводить до помітного зниження продуктивності системи. Обов'язково слід зважати, щоб не створювати індекси там, де вони не потрібні:

- коли індексів стає багато, вони займають відчутний обсяг оперативної пам'яті;
- наявність індексів суттєво уповільнює операції модифікації даних (вставки, видалення, оновлення), тому що під час зміни даних СКБД необхідно оновити індекси відповідно до нових значень індексованих даних.

Виграш від застосування індексів досягається в тих випадках, коли час, витрачений на створення і підтримку індексу, компенсується економією часу завдяки прискоренню часто виконуваних запитів.

Існує дві основні категорії типів індексів (рис. 7.8):

1. **Впорядковані індекси.** Засновані на впорядкуванні значень за сортуванням.

2. **Хеш-індекси.** Засновані на рівномірному розподілі значень по діапазону областей. Діапазон, до якого відноситься значення, визначається функцією, яка називається хеш-функцією.

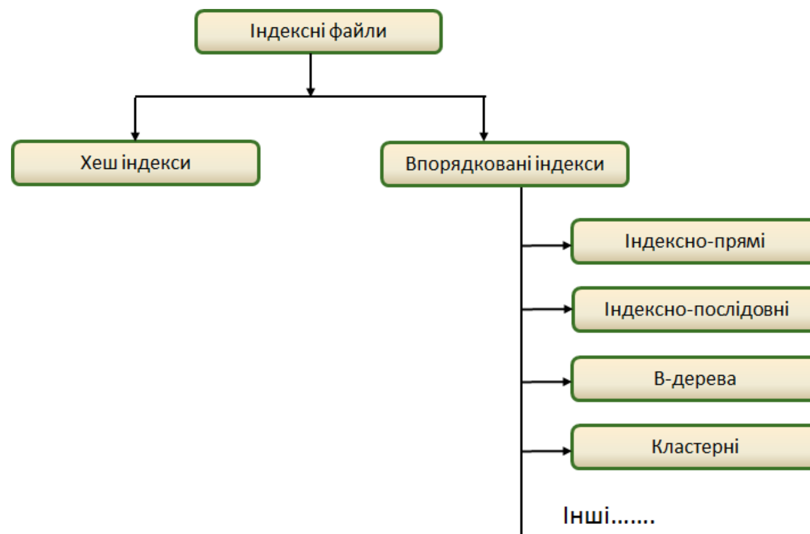


Рис. 7.8. Типи індексів

Атрибут або набір атрибутів, які використовуються для пошуку записів у файлі, називається ключем пошуку. Зауважте, що визначення ключа пошуку відрізняється від визначення первинного ключа. Таке дублююче значення терміну ключ є усталеним на практиці. Якщо у файлі є декілька індексів, то існує декілька ключів пошуку.

За властивостями ключа пошуку розрізняють такі види індексів:

- Простий індекс – індекс, побудований на одному полі таблиці.
- Складений індекс – індекс, побудований на двох і більше полях таблиці. Всі сучасні СКБД підтримують використання як простих, так і складених індексів.
- Унікальний індекс – індекс, побудований на полі (полях) таблиці, що містить унікальні значення. Неунікальний індекс – індекс, побудований на полі (полях) таблиці, що не містить унікальних значень. Усі сучасні СКБД підтримують використання як унікальних, так і неунікальних індексів.
- Індекс на наборі полів, який включає первинний ключ, називається первинним індексом; інші індекси називаються вторинними індексами. Терміни «первинний індекс» і «вторинний індекс» іноді використовуються у різних значеннях. Стандартної термінології в цьому питанні немає.

Будь-який індекс можна розглядати як колекцію, що складається з об’єктів спеціального виду (індексних записів), які містять два атрибути:

- ключ пошуку **K**, що представляє пошуковий образ (як правило, значення атрибута або атрибутів, для яких побудовано індекс);
- вказівник (або вказівники) на записи основної області, що містять достатньо інформації щоб знайти (один або декілька) записів даних зі значенням ключа

пошуку **K.** Вказівник позначається як **RID** (record Id) – ідентифікатор запису основних даних зі значенням ключа пошуку K.

Оскільки значення індексу впорядковано, можна ефективно здійснити пошук для знаходження потрібних вказівників, а потім використати їх для отримання записів даних.

Існує три основні варіанти того, що зберігати в індексі:

Варіант 1. Індексний запис є фактичним записом основних даних зі значенням ключа пошуку **K**. Можна розглядати такий індекс як спеціальну файлову організацію.

Варіант 2. Індексний запис – це пара **K, RID**, де **RID** – вказівник на запис основного файлу Такий індекс має назву щільний або індексно-прямий. Щільний індекс має індексний запис для кожного окремого запису основних даних. Якщо ключ пошуку містить атрибут, що не відноситься до первинного ключа, щільний індекс має індексний запис для кожної групи записів з однаковим значенням для цього атрибута.

Варіант 3. Індексний запис – це пара **K**, **RID-list**, де **RID-list** – це показник на блок записів основного файлу даних зі значенням ключа пошуку K. Такий індекс має назву нещільний або індексно-послідовний.

Варіанти (2) і (3), які містять індексні записи, що вказують на записи основних даних, не залежать від файлової організації, яка використовується для основного файлу. Варіант (3) забезпечує краще використання простору, ніж варіант (2), але індексні записи мають змінну довжину, залежно від кількості записів даних із заданим значенням пошукового ключа.

Якщо створюється більше одного індексу, то щонайбільше один з індексів може використовувати варіант (1), щоб уникати багаторазового формування індексних записів.

На практиці, індекс, який використовує варіант (1), називається первинним індексом (рис. 7.9), а той, який використовує варіант (2) або (3), називається вторинним індексом. Слід зазначити, що стандартної термінології в цьому питанні немає.

Employees

Empid	Empname	DOJ	DOB	
0012	Віктор	2013-06-15	1976-01-11	Page #1
0234	Петро	2013-06-15	1972-04-13	
0345	Ганна	2013-07-11	1981-01-10	
0478	Олена	2015-06-14	1986-04-11	
1625	Михайло	2016-06-12	1989-05-05	Page #2
1873	Михайло	2018-06-01	1996-05-11	
2791	Ганна	2019-09-15	1966-01-11	
5819	Петро	2021-08-13	1999-01-11	

Рис. 7.9. Організація первинного індексу

Два записи даних вважаються дублікатами, якщо вони мають однакове значення індексного поля. Первинний індекс гарантовано не містить дублікатів, але індекс на інших полях може містити дублікати. Загалом, вторинний індекс містить дублікати. Якщо дублікатів не існує, то ключ пошуку містить деякий ключ-кандидат.

Важливим питанням є те, як організовано записи даних в індексі для підтримки ефективного пошуку записів даних.

7.3.2. Впорядковані індекси

Щільні індекси

В індексно-прямих файлах основна область містить послідовність записів однакової довжини, розташованих у довільному порядку, а індексний запис містить значення первинного ключа і RID основної області, який має дане значення первинного ключа (варіант 2) [2]. RID визначається як зсув (у байтах) початку фізичного запису в основному файлі. Оскільки індексні файли будуються для первинних ключів, які однозначно визначають запис, то в індексно-прямих файлах для кожного запису в основній області існує тільки один запис з індексної області. Такий індекс називається щільним. Усі записи в індексній області впорядковані за значенням ключа (рис. 7.10).

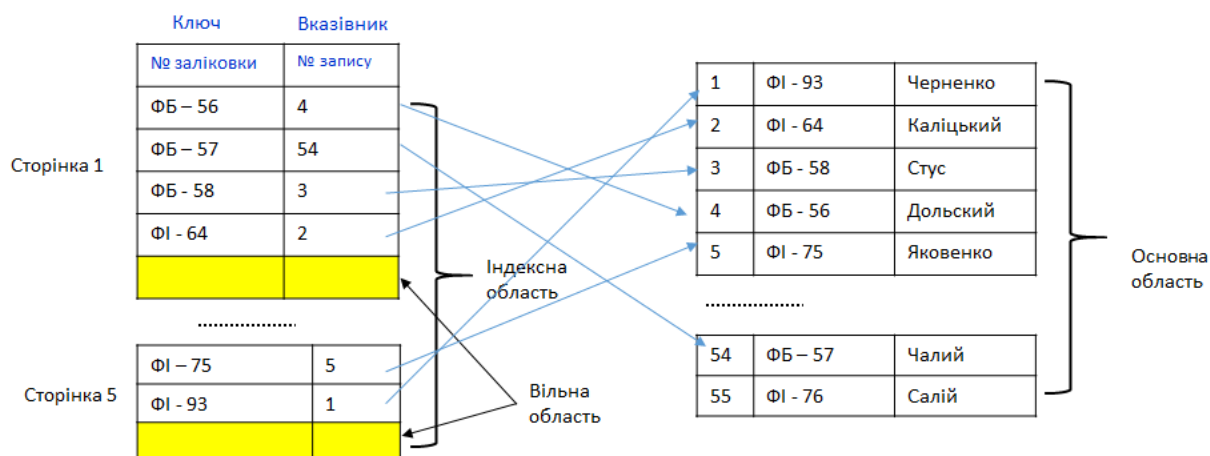


Рис. 7.10. Щільний індекс – кожному значенню ключа відповідає індексний запис

Щільні індекси зазвичай працюють швидше, але вимагають більше місця для зберігання і їх складніше підтримувати, ніж нещільні індекси.

Нещільні індекси

Основний файл має назву індексно-послідовний, якщо він підтримуються у відсортованому стані з моменту створення. В такому випадку можна створювати індексні записи з урахуванням сторінкової структури файлів на диску (варіант 3).

Індексний запис для таких файлів має містити: значення ключа першого запису на сторінці і RID. У цьому випадку RID формується як адреса початку сторінки фізичних записів у файлі. Тепер за заданим значенням первинного ключа в індексній області потрібно відшукати вже потрібну сторінку. Усі інші дії відбуваються в первинної пам’яті. Таким чином, кількість індексних записів суттєво скорочується.

За такого підходу новий запис має вноситися відразу в необхідний блок на необхідне місце. Природно, що для додавання записів сторінка основної області повинна мати вільне місце. Під час внесення нового запису індексна область не коригується.

Індексно-послідовні файли призначені для додатків, які вимагають як послідовної обробки всього файлу, так і довільного доступу до окремих записів.

На рис. 7.11 показано послідовний файл записів про викладачів. Записи зберігаються у відсортованому порядку за ідентифікатором викладача, який використовується як ключ пошуку.

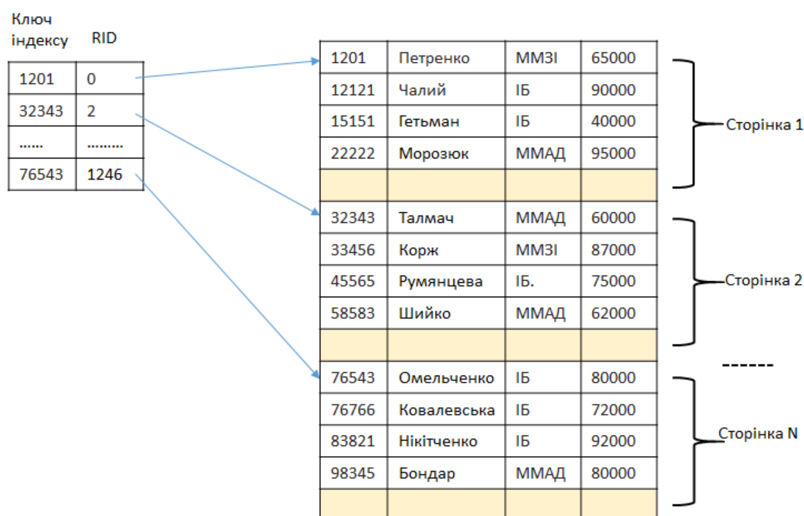


Рис.7.11. Організація нещільного індексу

Кластерний і некластерний індекси

Коли файл організовано так, що порядок записів даних збігається з порядком записів у деякому індексі, індекс має назву кластерний[5]. Пошуковий ключ кластерного індексу часто є первинним ключем, хоча це не обов'язково так. Індекси, ключ пошуку яких вказує порядок, відмінний від послідовного порядку у файлі, називаються некластерними. Індекс, який використовує варіант (1), є кластерним за визначенням. Індекс, який використовує варіант (2) або (3), може бути кластерним, тільки якщо записи даних відсортовано за ключовим полем пошуку. Але на практиці файли рідко зберігаються відсортованими за не ключовим атрибутом, оскільки це занадто дорого підтримувати при оновленні даних. Отже, на практиці, кластерний індекс – це індекс, який використовує варіант (1), а індекси, які використовують варіант (2) або (3) є некластерними.

На рис. 7.12 наведено приклад кластерного індексу, що організовано не за первинним ключем. Записи відсортовано за атрибутом Country.

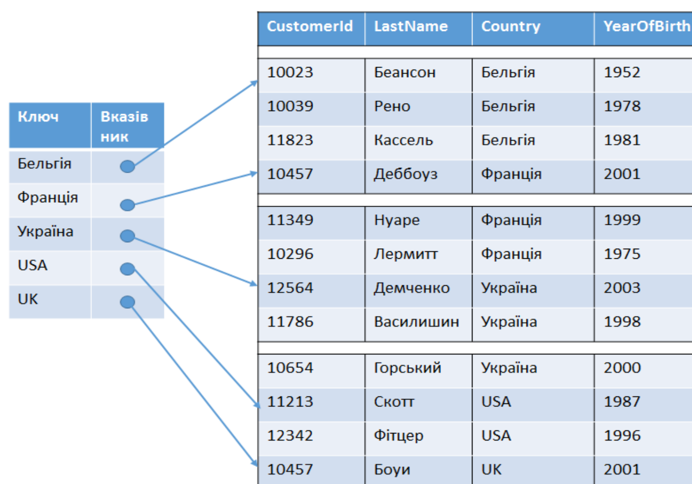


Рис. 7.12. Приклад кластерного індексу

[5] Не треба плутати з кластеризованою файловою структурою, розглянутою раніше

Якщо пошуковим ключем є не ключовий тип атрибута, значення пошукового ключа не можуть однозначно ідентифікувати окремий запис. Кожен запис індексу складається зі значення пошукового ключа, а також адреси першої сторінки, яка містить записи з цим значенням ключа. Процес пошуку такий самий, як і у випадку з первинним індексом, за винятком того, що після першого доступу до сторінки, можуть знадобитися додаткові доступи до інших наступних сторінок, щоб знайти всі записи з тим самим значенням пошукового ключа. У прикладі кластерний індекс визначено за пошуковим ключем «Країна». Для пошуку всіх записів зі значенням «Україна» у полі «Країна» потрібно виконати по індексу доступ до другої сторінки і доступ до третьої сторінки, яка також містить клієнтів з України.

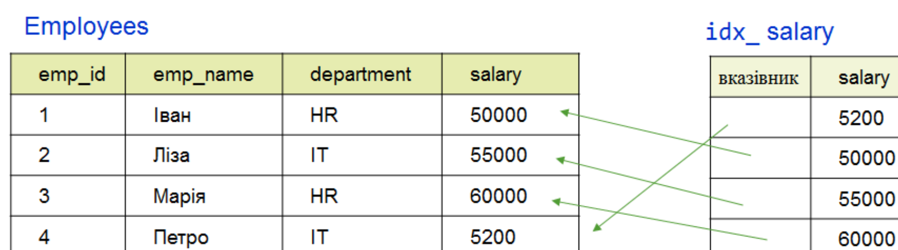
Отже , відмінності між кластерним і некластерним індексом [10]:

- Кластерний індекс визначає фізичний порядок рядків даних у таблиці. Кожна таблиця може мати лише один кластерний індекс.
- Коли таблиця має кластерний індекс, рядки зберігаються в порядку ключа індексу. Це означає, що дані фізично організовані на диску на основі значень в індексованому стовпчику.
- Завдяки фізичному впорядкуванню, пошук рядків за допомогою кластерного індексу відбувається швидше при виконанні запитів за індексованим стовпцем.
- Створення або перебудова кластерного індексу може зайняти більше часу, оскільки він впливає на фізичний порядок даних у таблиці.
- Некластерний індекс – це окрема структура даних, яка містить копії проіндексованих стовпців разом із вказівником на реальні рядки даних у таблиці.
- Таблиця може мати декілька некластерних індексів, що дозволяє використовувати різні стратегії індексування для оптимізації різних запитів.
- Некластерні індекси не впливають на фізичний порядок даних у таблиці; натомість вони забезпечують швидкий шлях пошуку до фактичних рядків даних.
- Некластерні індекси, як правило, швидше створюються або перебудовуються порівняно з кластерними індексами.

Приклад. Якщо створити некластерний індекс для стовпця salary (рис. 7.13):

```
CREATE NONCLASTERED INDEX idx_salary ON Employees(salary);
```

Буде створено індексну структуру, яка міститиме копію стовпця salary разом із вказівниками на реальні рядки в таблиці Employees. Будь-який запит, який використовує стовпець salary в пошуку або операції об'єднання, буде виконуватись швидше.



Employees

emp_id	emp_name	department	salary
1	Іван	HR	50000
2	Ліза	IT	55000
3	Марія	HR	60000
4	Петро	IT	5200

idx_ salary

вказівник	salary
	5200
	50000
	55000
	60000

Рис. 7.13. Приклад некластерного індексу

Якщо створити кластерний індекс для атрибуту emp_id:

```
CREATE CLASTERED INDEX idx_empid ON Employees(emp_id)
```

то дані зберігаються фізично впорядкованими по emp_id.

Вибір між кластерним і некластерним індексом залежить від конкретного дизайну бази даних і типів запитів, які потребують оптимізації.

Багаторівневі індекси

Зазвичай СКБД намагається зберегти індексну таблицю у первинній пам'яті. Основна таблиця зберігається у вторинній пам'яті через її розмір. Основна таблиця може містити мільйони записів, для яких навіть нещільний індекс стає настільки великим, що не можна зберігати його в первинної пам'яті, що призводить до багаторазових звернень до диску. А це означає втрату переваги швидкості доступу. Багаторівневий індекс допомагає розбити індекс на декілька менших індексів, щоб зробити зовнішній рівень настільки малим, що він може бути збережений в одному блоці на диску, який можна легко розмістити в будь-якому місці оперативної пам'яті.

Прикладом багаторівневих індексів [2,3] є так звані інвертовані списки (рис. 7.14).

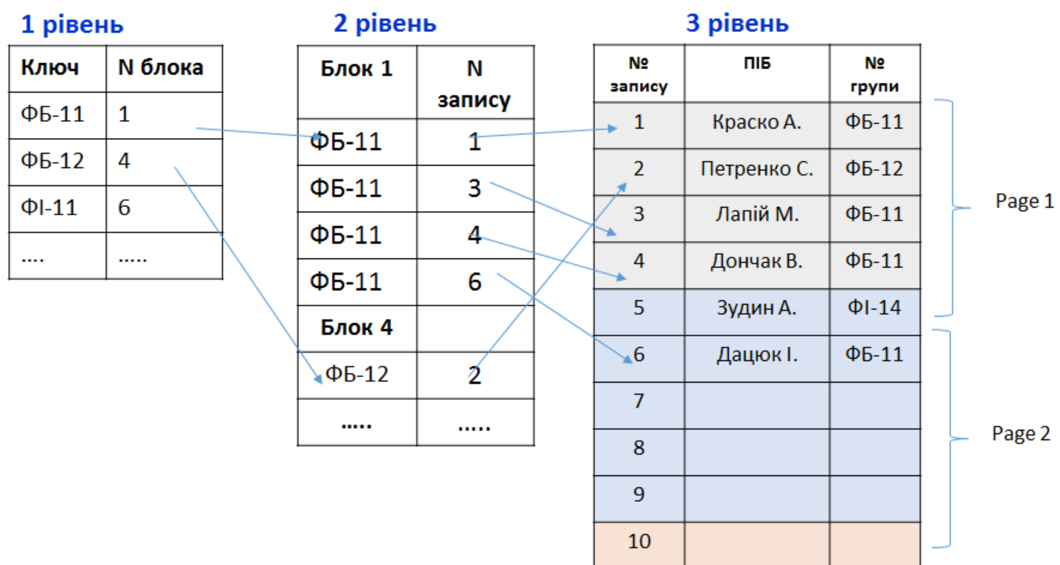


Рис. 7.14. Організація індексу в вигляді інвертованих списків

Інвертований список – це дворівнева індексна структура. Тут на першому рівні знаходиться файл, в якому упорядковано розташовані значення вторинних ключів (в нашому прикладі це номер групи). Кожен запис з вторинним ключем має посилання на номер першого блоку в ланцюжку блоків, що містять номери записів з даним значенням вторинного ключа. На другому рівні знаходиться ланцюжок блоків, що містять номери записів, що містять одне і те ж значення вторинного ключа. Блоки другого рівня також впорядковані за значеннями вторинного ключа. І нарешті, на третьому рівні знаходиться власне основний файл.

Самим популярним варіантом багаторівневої індексації є B+ дерева.

7.3.3. B+-деревоподібні індексні файли

Висока продуктивність індексів з бінарним пошуком забезпечується тим, що під час їхнього використання вдається уникнути обов'язкового перегляду всього вмісту

файлу. Однак, якщо індексований файл має великий розмір, то і його індекс також дуже великий і послідовний перегляд навіть тільки одного індексу займає значний час.

Багаторівнева індексація йде на крок далі в тому сенсі, що вона усуває необхідність бінарного пошуку, створюючи індекси до самого індексу.

Структура індексу B+-дерева є найпоширенішою з декількох структур індексів, які зберігають свою ефективність, незважаючи на додавання та видалення даних. Індекс B+дерева має вигляд збалансованого дерева, в якому кожен шлях від кореня дерева до листка дерева має однакову довжину.

Кожен нелистяний вузол дерева (крім кореня) має від **n/2** до **n** нащадків, де **n** є фіксованим для конкретного дерева; корінь має від **2** до **n** нащадків.

Середня кількість нащадків для не листкової вершини називається розгалуженістю (fan-out) дерева. Якщо кожна не листова вершина має **n** нащадків, то дерево висотою **h** має рівень розгалуженості **F = n * h**. На практиці **F** дорівнює щонайменше 100, що означає, що дерево висотою чотири містить 100 мільйонів сторінок. Таким чином, можна здійснити пошук у файлі зі 100 мільйонами сторінок і знайти потрібну сторінку за чотири операції вводу/виводу; на відміну від індексного пошуку, що зайняв би більше 25 операцій вводу/виводу.

Побудова B+-дерев пов'язана з простою ідеєю побудови індексу над уже побудованим індексом. Нехай основна область відсортована і має структуру нещільного індексу. Над основною областю побудуємо ще один рівень нещільних індексу, кожен запис якого вказує на сторінку основної області. А потім ще один рівень, доти поки нещільний індекс верхнього рівня не буде мати розмір в одну сторінку (рис. 7.15).

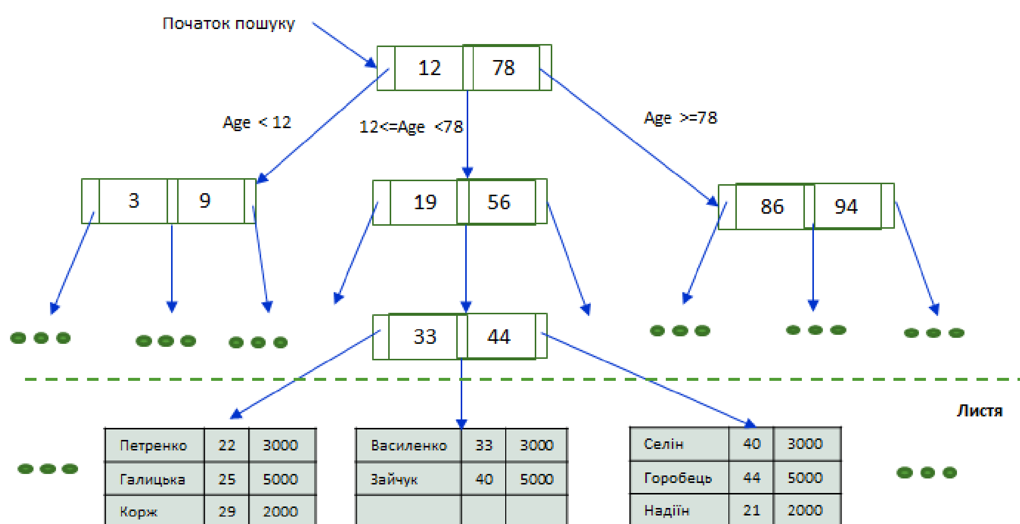


Рис. 7.15. Приклад B+-дерева

Кожна вершина дерева відповідає сторінці в просторі, виділеному для зберігання індексу. Індексні записи, включені в дерево, розміщуються відповідно до відношення порядку для індексних ключів. Таким чином, кожна сторінка містить деякий інтервал значень ключів; усі ключі, що потрапляють у цей інтервал, розміщуються на цій сторінці. Якщо ключі індексних записів унікальні, то інтервали, що відповідають різним сторінкам, не перетинаються.

Структура B+-дерева додає накладні витрати на продуктивність при вставці та видаленні, а також додає накладні витрати на простір. Ці витрати є прийнятними навіть для файлів, що часто модифікуються, оскільки не потрібно витрачатися на реорганізацію файлу. Крім того, оскільки вузли можуть бути наполовину порожніми (якщо вони мають мінімальну кількість дочірніх елементів), виникає деяка втрата простору. Ці витрати простору також є прийнятними, враховуючи переваги продуктивності структури B+дерева. Процедура побудови дерева гарантує, що всі сторінки індексу будуть заповнені не менше ніж на 50 %.

7.3.4. Індексування на основі хешування

Хешування є широко використовуваною технікою для створення індексів в основній (оперативній) пам'яті. Хешування також використовується як спосіб організації записів у файлі, хоча хеш-файли не дуже широко застосовуються.

Алгоритм хешування визначає перетворення ключа на адресу таким чином, що фізичну адресу запису можна обчислити за значенням його ключа. Кожного разу, коли додається новий запис, це перетворення застосовується до його ключа, повертаючи фізичну адресу, за якою має зберігатися запис.

Перетворення ключа в адресу складається з декількох кроків (рис. 7.16) [3].

Рис. 7.16. Послідовність створення хеш-індексу

По-перше, ключ перетворюється у цілий числовий формат, якщо він ще не є цілим числом. Наприклад, не цілі числові значення можуть бути округлені або алфавітно-цифрові ключі можуть бути перетворені в ціле число за допомогою їх ASCII-кодів. Потім застосовується власне алгоритм хешування, де цілочисельні значення ключів перетворюються на числа, розподілені приблизно в тому ж діапазоні, що й адреси. Чим більш рівномірно ці ключі розподілені в цьому діапазоні чисел, тим краще. Досить часто згенеровані адреси відносяться не до окремих адрес записів, а до області адрес записів, яка називається «кошик» (bucket). Кошик містить один або декілька слотів для записів, що зберігаються. Його розмір може бути визначений довільно, але також може бути узгоджений з фізичними характеристиками пристрою зберігання даних, наприклад, з цілим числом дискових блоків, доріжок або циліндрів.

Для дискових хеш-індексів, кошик може бути зв'язаним списком блоків на диску. В організації хеш-файлів замість вказівників на записи у кошиках зберігаються самі записи. Такі структури мають сенс лише для даних, що резидентно зберігаються на диску.

Формально, алгоритм пошуку адреси виглядає так: якщо **K** – множина всіх значень ключа пошуку, **B** – множина всіх адрес кошиків, то хеш-функція **h** – це перетворення значення **K** в **B**:

$$\mathbf{B} = \mathbf{h}\ (\mathbf{K}).$$

Щоб вставити запис з пошуковим ключем $\mathbf{K_i}$, обчислюється $\mathbf{h(K_i)}$, який дає нам адресу кошику для цього запису. Додається індексний запис для цього запису до списку зі зміщенням i.

Ключ пошуку для індексу може бути будь-якою послідовністю одного або декількох полів, і він не обов'язково повинен однозначно ідентифікувати записи. Наприклад, в індексі зарплат кілька записів мають однакове значення ключа пошуку (рис. 7.17).

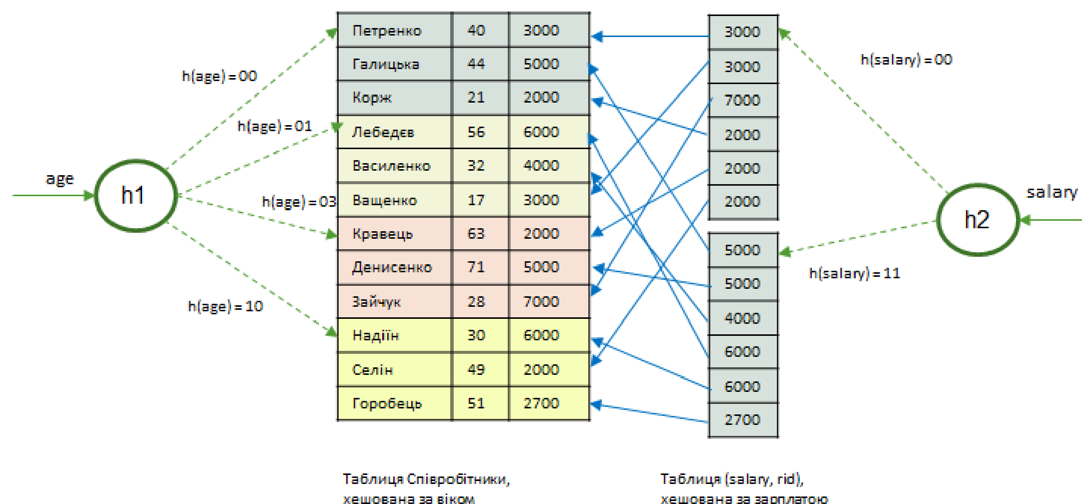


Рис. 7.17. Приклад хеш-індексу

Методи хешування

Для перетворення ключових значень в адреси було розроблено багато методів хешування, таких як ділення, порозрядний аналіз, середнє квадратичне, згортання та перетворення основи. Розглянемо підхід до хешування на прикладі одного з них:

а) Ділення.

Це одна з найпростіших технік хешування, але в більшості випадків вона працює дуже добре. За допомогою методу ділення числова форма ключа ділиться на ціле число **M**. Залишок від ділення стає адресою запису:

$$\mathbf{B\ (K_i) = K_i\ mod\ M}$$

де **M** – просте число (наприклад, M = 1009).

Вибір **M** дуже важливий. Якщо **M** не підходить для даного набору ключів, це призведе до численних колізій, а отже, і до значного переповнення. Слід уникати спільних множників між значеннями ключів і **M**. З цієї причини, вибране **M** часто є простим числом.

Для певного набору ключів одне просте число може виявитися кращим за інше, тому в ідеалі тестується декілька кандидатів. Найчастіше вибирають просте число, яке близьке до кількості доступних адрес, але трохи більше, щоб отримати приблизно стільки ж значень, скільки і кількість необхідних адрес. На наступному кроці перетворення ключа в адресу ці значення множаться на коефіцієнт (трохи) менший за 1, так що вони ідеально вписуються у реальний адресний простір.

б) Функція середнього квадрата.

В цьому методі числове значення ключа зводиться у квадрат. Якщо зобразити таке число у вигляді послідовності бітів, то в якості адреси вибираються біти в середині такої послідовності:

1. $\mathbf{K_i \rightarrow Z_i}$

2. $\mathbf{N_i = Z_i^2}$

3. $\mathbf{B_i}$ = середні біти $\mathbf{N_i}$ = домашня адреса запису в таблиці розміром **2b**.

в) Функція стиснення.

В цьому методі двійкове представлення числового ключа ділиться на кілька секцій, значення цих секцій складаються і найменші значущі біти беруться як адреса

1. $\mathbf{K_i \rightarrow Z_i}$

2. Двійкове представлення $\mathbf{Z_i}$ на ділиться декілька секцій довжиною **b** біт. Потім значення цих секцій складаються і найменші значущі біти беруться як адреса у таблиці розміром **2b**.

Колізії хеш-значень

Хоча метою хешування є рівномірний розподіл всіх ключів у доступному адресному просторі, декілька записів можуть мати однаковий хеш.. Така ситуація має назву колізія, а відповідні записи називаємо синонімами.

Розглянемо простий приклад (рис. 7.18), який ілюструє використання наведених вище хеш-функцій. Нехай значення ключа складається лише з двох символів. Використовуємо тільки великі літери і перетворюємо ключі в числові значення, замінюючи A на 1, B на 2, ... Z на 26. Потім застосуємо три хеш-функції до числового представлення ключа.

Розмір нашої таблиці дорівнює 10 для хеш-функцій середнього квадрата і стиснення і 11 для функції цілочисельного ділення, оскільки для хорошої продуктивності цілочисельне ділення вимагає, щоб розмір таблиці був простим числом.

Хеш-функція цілочисельного ділення мала значення **z mod 11**. Функція піднесення до квадрату полягала у піднесенні z до квадрату та вилученні лише однієї середньої цифри, оскільки таблиця має розмір 10. В прикладі вибрано 4-ту цифру справа (це одна з двох середніх цифр). Функція стиснення також повинна перетворити z в одну цифру, оскільки розмір таблиці 10. В прикладі перетворення здійснюється шляхом додавання цифр z, а потім вибір найменшої значущої цифри. Можна бачити, що виникло декілька колізій, незалежно від того, яка хеш-функція використовувалася

Key (k)	Numeric Representation (z)	Integer Divide	z2	Mid-digit of z2	Compression Function
AB	0102	3	0010404	0	3
AZ	0126	5	0015876	5	0
BN	0214	5	0045796	5	7
NB	1402	5	1965604	5	7
CD	0304	7	0092416	2	7
EF	0506	0	0256036	6	1
GH	0708	4	0501264	1	5
HK	0811	8	0657271	7	0

z mod 11 одна середня цифра 0+8+1+1 = 10

Рис. 7.18. Приклад колізії хеш-функцій

Колізії самі по собі не є проблемою, оскільки один кошик зазвичай складається з декількох слотів для записів, але якщо синонімів більше, ніж слотів для певного кошику, то вважається, що кошик переповнений. Існують різні способи боротьби з переповненням, але незалежно від фактичного методу обробки переповнення, воно неминуче призводить до того, що деякі записи зберігаються не там, де вони спочатку очікувалися відповідно до перетворення ключа до адреси.

Припустимо, що два ключі пошуку, K_5 і K_7, мають однакове хеш-значення, тобто $h(K_5) = h(K_7)$. Якщо ми виконуємо пошук за ключем K_5, то знайдемо і записи зі значеннями ключа пошуку K_7. Таким чином, необхідно перевірити значення ключа пошуку для кожного запису у кошику, щоб переконатися, що це саме той запис, який нам потрібен.

У дисковому хеш-індексі, коли вставляємо запис, і місця у кошику не вистачає, то відбувається переповнення кошика і створюється додатковий кошик. Якщо і додатковий кошик переповнений, система створює ще один і так далі. Всі додаткові кошики даного кошика з'єднуються разом у зв'язаний список. У випадку переповнення, якщо задано ключ пошуку K, алгоритм пошуку повинен шукати не лише кошик h(K), але й додаткові кошики, з'єднані з кошиком h(K) (рис. 7.19).

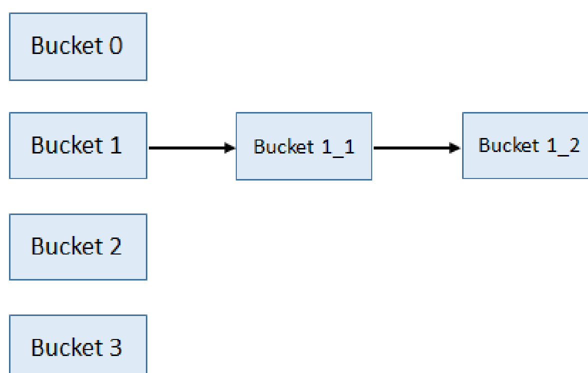


Рис. 7.19. Організація переповнення в хеш-індексах

Отже, для отримання записів, що переповнилися, потрібні додаткові звернення до блоків, і тому переповнення слід уникати, наскільки це можливо, щоб не ставити під загрозу продуктивність. З цієї причини слід обирати алгоритм хешування, який розподіляє ключі якомога рівномірніше за відповідними адресами в блоках, зменшуючи таким чином ризик переповнення.

Щоб зменшити ймовірність переповнення кошиків, кількість кошиків вибирається рівною

$$\mathbf{N = (n / f) * (1 + d)}$$

де **n** – кількість записів, **f** – кількість записів у кожному кошику, **d** – коефіцієнт заповнення, зазвичай близько 0.2. В такому випадку близько 20 відсотків простору у кошиках буде порожнім.

Хеш-індекси ефективно підтримують запити на рівність пошукових ключів. Щоб виконати пошук за значенням пошукового ключа $\mathbf{K_i}$, просто обчислюється $\mathbf{h(K_i)}$, а потім виконуємо пошук за цим адресом.

Видалення відбувається так само просто. Якщо значення пошукового ключа запису, який потрібно видалити, дорівнює $\mathbf{K_i}$, обчислюємо $\mathbf{h(K_i)}$, потім знаходиться цей запис у відповідному кошику і видаляється його з цього кошику.

Суттєвим недоліком хеш-індексу є відсутність підтримки запитів за діапазоном значень. Наприклад, на запит, який хоче отримати всі значення пошукового ключа **v** такі, що $\mathbf{l \le v \le u}$, не можна ефективно відповісти за допомогою хеш-індексу.

7.3.5. Вартість індексування

Жоден метод індексування не є найкращим. Скоріше, кожен метод найкраще підходить для конкретних застосунків баз даних. Кожну технологію слід оцінювати на основі таких факторів:

- Типи доступу, які підтримуються ефективно. Типи доступу можуть включати пошук записів із заданим значенням атрибута і пошук записів, значення атрибутів яких потрапляють у заданий діапазон.
- Час, необхідний для пошуку певного елемента даних або набору елементів за допомогою відповідного методу.
- Час, необхідний для вставки або видалення нового елемента даних.
- Додатковий простір, який займає структура індексу.

Оскільки основне призначення індексів – прискорення пошуку даних, найважливішим критерієм їхньої корисності є час пошуку в індексі. Тому для оцінки часу пошуку використовується показник кількості сторінок, які необхідно обробити, для того щоб виконати запит на пошук в індексі (таблиця на рис. 7.20).

В – кількість сторінок даних

	Файл купи	Сортований файл (100% заповнення)	B+ Tree Index (67% заповнення)
Сканування всіх записів	B	B	1.5 B
Пошук унікального ключа	0.5 B	$\log_2 B$	$\log_F 1.5B$
Пошук діапазону значень	B	($\log_2 B$) + к-ть сторінок з діапазону	($\log_F 1.5B$) + к-ть сторінок з діапазону
Insert	2	($\log_2 B$)+B	($\log_F 1.5B$)+1
Delete	0.5B+1	($\log_2 B$)+B	($\log_F 1.5B$)+1

Рис. 7.20. Вартість індексування

7.3.6. Рекомендації щодо створення індексів

Індекси можна створювати за допомогою наступного синтаксису, який підтримується більшістю СКБД [10]:

```
CREATE INDEX <index_name> ON <table_name> (<attribute_list>);
```

`attribute_list` – список атрибутів таблиці, які формують ключ пошуку для індексу.

Якщо значення атрибутів таблиці оголошено унікальними або первинним ключем, індекс на таких атрибутах створюється автоматично.

Нижче наведено кілька практичних правил вибору індексів для реляційних баз даних:

1. Індекси найбільш корисні для великих таблиць. Не треба створювати індекс на таблиці маленького розміру (менше п’яти фізичних сторінок).
2. Індекси найбільш корисні для стовпців, які часто з’являються в реченнях WHERE SQL-команд наприклад, індекс на атрибут ProductFinish:

```
WHERE ProductFinish = 'Дуб',
```

   або для з’єднання таблиць за якими захоплюють від 5 до 10% рядків таблиці. Наприклад, індекс на зовнішній ключ ProductID:

```
WHERE Product_T.ProductID = OrderLine_T.ProductID
```

3. Використовуйте індекс для атрибутів, на які є посилання в реченнях `ORDER BY` і `GROUP BY`. Однак необхідно бути обережними з цими реченнями. Переконайтеся, що СКБД дійсно буде використовувати індекси для атрибутів, перелічених у цих реченнях (наприклад, Oracle використовує індекси для атрибутів у реченнях `ORDER BY`, але не в реченнях `GROUP BY`).
4. Якщо в запитах активно використовується з’єднання таблиць, атрибути що використовуються для з’єднання треба індексувати. Наприклад, якщо є популярний запит:

```
Select e.EmpName, d.DeptName
From Employees e Join Departments d ON e.DeptId = d.DeptId;
```

   необхідно створити індекси:

```
CREATE INDEX idx_emp_deptid ON Employees(DeptId);
```

```
CREATE INDEX idx_dept_deptid ON Departments (DeptId);
```

інакше буде виконуватись повне сканування обох таблиць.

5. Хороші кандидати для індексування є будь-які колонки, які містять унікальні значення і колонки, які часто використовуються для перевірки правильності введення даних.
6. Погані кандидати для індексування колонки з низькою кардинальністю. Кардинальність (cardinality) колонки таблиці – кількість різних значень колонки. Її можна визначити за допомогою команди

```
SELECT COUNT (DISTINCT колонка) / COUNT (*) FROM таблиця
```

7. У багатьох СКБД на рядки з Null значенням не буде посилань в індексі (тому їх не можна знайти за допомогою пошуку в індексі на основі значення атрибута Null). Так що колонки з багатьма значеннями Null не слід індексувати. Пошук в таких колонках здійснюється шляхом сканування файлу.
8. Не слід індексувати колонки, в яких часто змінюються значення або якщо значна довжина ключа пошуку (більше 25 байтів).
9. Не слід створювати для таблиці кілька складових індексів, що містять одні й ті ж колонки, але в іншому порядку. Складовий індекс краще використовувати, якщо кілька стовпців з низькою кардинальні в комбінації з один з одним можуть дати набагато вищу кардинальність. Не можна допускати, щоб у кількох індексів для однієї таблиці була однакова лідируюча частина.

Сучасні СКБД мають в своєму складі інструментальні засоби, які дозволяють аналізувати запити, що потребують повного сканування великих таблиць, а також оцінюють ефективність створених індексів. Наприклад, це Explain для MySQL або Execution Plans для MS SQL Server. Створений за допомогою таких запитів план виконання демонструє кроки, які виконує сервер для отримання результату.

Контрольні запитання

1. Дайте визначення індексу в базі даних.
2. Перелічіть та охарактеризуйте різні типи індексів.
3. Опишіть процедуру встановлення та видалення індексу.
4. Сформулюйте основні завдання етапу фізичного проєктування.
5. Опишіть метод доступу – хешування. У чому полягає проблема колізії.
6. Опишіть метод доступу з щільним індексом. У чому переваги та недоліки методу.
7. У чому різниця між кластерним і некластерним індексами
8. Опишіть метод доступу з нещільним індексом.
9. Опишіть організацію індексів у вигляді B-дерев.
10. Поясніть, чому індекс корисний лише тоді, коли існує достатня різноманітності значень атрибута
11. Скільки кластерних індексів можна створити для таблиці?
12. Скільки некластерних індексів можна створити для таблиці?
13. У яких випадках слід використовувати хеш-індекс?
14. Які основні фактори впливають на вартість операцій з базами даних? Обговоріть просту модель витрат, яка відображає це

15. Що таке складений ключ пошуку? Які переваги та недоліки складених пошукових ключів?

Тестові завдання

1. Яка основна мета створення індексу?
   A. Документування відношень
   B. Виявлення проблем цілісності
   C. Додавання до таблиць функціональності тригерів
   D. Покращення швидкості пошуку та представлення даних
2. Що не відповідає дійсності стосовно індексів (2 відповіді):
   A. Повільніше послідовно скануються, ніж відповідні таблиці
   B. Зазвичай менші, ніж таблиці, на які вони посилаються
   C. Можуть використовуватися разом з обмеженнями первинного ключа
   D. Може використовуватися для покращення продуктивності вставки, оновлення та видалення
   E. Може використовуватися для покращення продуктивності запитів
3. Скільки стовпців у структурі індексу?
   A. 2
   B. 3
   C. 4
   D. 5
4. Оскільки первинні ключі відсортовано, операції ___ є досить ефективними через їх відсортований порядок.
   A. Додавання
   B. Оновлення
   C. Пошуку
   D. Видалення
5. Якщо кожне значення ключа пошуку у основному файлі даних відповідний має індексний запис, індекс має назву
   A. Нещільний
   B. Щільний
   C. І А, і В
   D. Жодне з перерахованих вище
6. При використанні щільного індексу немає різниці у кількості ___ в індексній та основній таблицях.
   A. Файлів
   B. Баз даних
   C. Записів
   D. Наборів даних
7. ___ індексний запис використовується для посилання на кілька елементів у файлі даних.
   A. Щільний

   B. Нещільний
   C. I A, і B
   D. Жодне з перерахованих вище
8. Елементи нещільного індексу вказують на ___.
   A. Файли
   B. Записи
   C. Сторінки
   D. Нічого з перерахованого
9. Що таке індекс у базі даних?
   A. Це структура даних, яка підвищує швидкість операцій пошуку даних у таблиці бази даних
   B. Це віртуальна таблиця, яка зберігає фактичні дані таблиці бази даних
   C. Це обмеження, яке забезпечує унікальність даних у стовпці або групі стовпців
   D. Це подання (view), яке представляє персоналізований результат запиту на основі певних критеріїв
10. Чи можна створити кілька індексів для однієї таблиці?
   A. Так, якщо індекс кластерний
   B. Так, якщо індекс некластерний
   C. Так, якщо індекс щільний
   D. Так, якщо індекс нещільний
11. Чи можна використовувати інструкцію `CREATE INDEX` для створення індексу для стовпця з Null значеннями?
   A. Так
   B. Ні
   C. Ні, якщо стовпець що індексуються є первинний ключ
   D. Ні, якщо стовпець що індексуються є частиною складеного індексу
12. Індекс можна створити на ____________ або на наборі стовпців у таблиці.
   A. Один стовпець
   B. Первинний ключ
   C. Зовнішній ключ
   D. Усі стовпці
13. Ви хочете створити індекс з назвою «idx_name» в таблиці «Employees» для стовпця «Name». Як слід написати інструкцію `CREATE INDEX`, щоб досягти цього?
   A. `CREATE INDEX idx_name ON Employees(Name);`
   B. `CREATE INDEX Employees_idx_ name ON Employees(Name);`
   C. `CREATE INDEX ON Employees(idx_name, Name);`
   D. `CREATE INDEX ON Employees(Name) INDEX_NAME idx_name;`
14. Файлова організація, яка забезпечує дуже швидкий доступ до будь-якого довільного запису файлу:
   A. Впорядкований файл
   B. Інвертований файл
   C. Хешований файл
   D. B+-дерево

15. У вас є таблиця під назвою « Orders» з декількома стовпчиками, які часто використовуються в операціях сортування та фільтрації. Як можна записати інструкцію `CREATE INDEX`, щоб створити індекс для кількох стовпців, таких як «OrderDate» і «CustomerID»?

    A. `CREATE INDEX idx_orderdate_customerid ON Orders (OrderDate, CustomerID);`

    B. `CREATE INDEX ON Orders (OrderDate, CustomerID);`

    C. `CREATE INDEX Orders_idx_orderdate_customerid ON (OrderDate, CustomerID);`

    D. `CREATE INDEX (OrderDate, CustomerID) ON Orders idx_orderdate_customerid;`

8. УПРАВЛІННЯ ТРАНЗАКЦІЯМИ

8.1. Поняття транзакцій

Незалежно від того, наскільки ретельно база даних розроблена і створена, вона може бути легко пошкоджена або зруйнована, якщо немає належного контролю паралелізму та методів відновлення. Контроль паралелізму – це можливість керувати одночасними процесами, пов'язаними з базою даних, так, щоб вони не заважали один одному. Відновлення бази даних – це процес відновлення бази даних до правильного стану в разі збою. Обидва ці засоби необхідні для захисту бази даних від небажаної зміни або втрати даних.

Поняття транзакції є центральним для розуміння як відновлення, так і контролю паралелізму. Транзакцію можна розглядати як логічну одиницю роботи, яка містить один або декілька операторів SQL. Логічною одиницею роботи може бути ціла програма з декількома операторами SQL, частина програми або одна команда SQL. Загалом, транзакція може включати будь-яку кількість операцій над базою даних.

Транзакція завжди повинна переводити базу даних з одного узгодженого стану в інший. Під час виконання транзакції допускається тимчасовий не узгоджений стан. Однак наприкінці транзакції всі зміни повинні бути зафіксовані в базі даних. Транзакція є атомарним процесом, не можна дозволити виконати лише частину транзакції – або весь набір кроків має бути виконаний, або жодного не може бути виконано, оскільки часткова транзакція залишить базу даних у неузгодженому стані.

Існує два способи завершення або припинення транзакції. Якщо транзакція успішно завершується, вона вважається зафіксованою, що означає, що всі внесені нею зміни зберігаються в базі даних, а база даних приводиться до нового узгодженого стану. Інша можливість полягає в тому, що транзакція не може бути успішно виконана. У цьому випадку транзакція переривається, що означає, що її зміни не зберігаються в базі даних. Якщо транзакцію перервано, дуже важливо, щоб база даних була відновлена до узгодженого стану, в якому вона перебувала до початку транзакції. Така транзакція відкочується, тобто всі зміни, внесені нею в базу даних, скасовуються. Зафіксовану транзакцію не можна скасувати. Якщо ми вирішили, що зафіксована транзакція була помилковою, ми повинні виконати компенсуючу транзакцію, щоб скасувати її наслідки. Однак перервана транзакція, яка була відкочена, в майбутньому, в залежності від причини збою, може успішно виконатися і бути зафіксована в цей час.

Можливі стани транзакції показано на рис. 8.1.

Активний стан. Це перший стан кожної транзакції. Коли всі операції читання і запису транзакції виконані, вона залишається в «Активному стані». Якщо всі операції виконані без помилок, транзакція переходить в «Частково зафіксований стан»; якщо відбувається збій, вона переходить в «Відхилений стан». Активний стан транзакції починається з оператора `BEGIN TRANSACTION` (або його еквівалента) і триває доти, доки прикладна програма не перерветься або не завершиться успішно, що є кінцем

транзакції. Зазвичай відповідальність за визначення початку та кінця кожної транзакції покладається на програміста. Команди `BEGIN TRANSACTION`, `COMMIT`, `ABORT`, `ROLLBACK` і `END TRANSACTION` або подібні доступні в мовах маніпулювання даними (DML) для розмежування транзакцій. Якщо ці команди не використовуються, вся програма може розглядатися як одна транзакція, і система автоматично виконує `COMMIT` після коректного завершення програми.

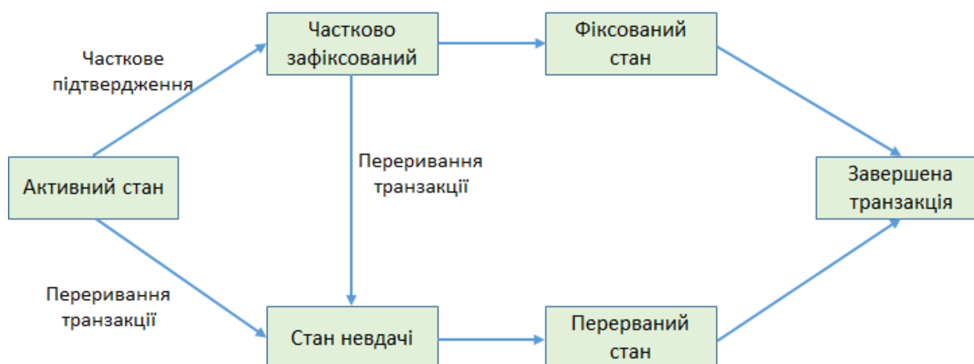


Рис. 8.1. Діаграма станів транзакції

Частково зафіксований стан. Коли транзакція виконує свою останню операцію, вона переходить у частково зафіксований стан, де зміни вносяться в основну пам'ять, а не в реальну базу даних. Якщо зміни вносяться в базу даних назавжди, вона переходить у фіксований стан. Якщо відбувається збій до того, як зміни стануть постійними в базі даних, вона перейде в стан невдачі («Failed State»). На цьому етапі СКБД перевіряє, чи транзакція не порушує протоколи контролю паралелізму, або будь-які обмеження, і що система здатна здійснити необхідні зміни до бази даних. Якщо проблем не виникає, транзакція фіксується, робиться запис у журналі транзакцій, і процес завершується.

Фіксований стан. Транзакція переходить з частково фіксованого стану в фіксований, коли транзакція успішно виконує всі свої операції, після чого всі зміни, зроблені в локальній пам'яті під час виконання транзакції, назавжди зберігаються в базі даних. Вважається, що база даних знаходиться на жорсткому диску або вторинному сховищі, і дані, що зберігаються в ній, зберігаються навіть після збою.

Стан невдачі. Якщо транзакція виконується і відбувається апаратний або програмний збій, транзакція з активного стану переходить в «Стан невдачі». Якщо транзакція знаходиться в частково зафіксованому стані і відбувається збій, вона також переходить в стан невдачі. Як і у випадку з частково зафіксованим станом, внесені зміни ще не відобразилися на постійному сховищі. Якщо виник такий стан, всі оновлення, зроблені в базі даних поточною транзакцією, слід відкотити.

Якщо транзакція зазнає невдачі під час виконання, вона переходить у «Перерваний стан» зі стану невдачі. В цей час зміни, внесені в локальну пам'ять, повинні бути відкочені до попереднього узгодженого стану. На цьому етапі система має два варіанти – перезапустити транзакцію або знищити транзакцію.

Програмне управління транзакціями

Розробники можуть програмно організовувати транзакції для виконання важливих операцій, використовуючи такі команди:

- `BEGIN TRANSACTION` – початок транзакції;
- `COMMIT` – завершення транзакції і збереження змін;
- `SAVEPOINT` *ім'я_точки_зберігання* – установка особливих точок повернення (`SAVE TRANSACTION` для SQL Server)
- `ROLLBACK [TO` *ім'я_точки_зберігання*`]` – скасування змін і повернення до точки зберігання.

Приклад. Переказ 100 грн. з рахунку А на рахунок В:

```
BEGIN TRANSACTION
UPDATE Accounts SET BAL = BAL-100 WHERE AccountId = 1
UPDATE Accounts SET BAL = BAL+100 WHERE AccountId = 2
-- фіксація змін
COMMIT
-- або, якщо відбулася помилка, повертаємось до початкового стану
ROLLBACK
```

Під час виконання транзакції можуть бути сформовано кілька точок зберігання. Якщо транзакція повертається до стану відповідно точки зберігання (рис. 8.2), вся зміни і точки зберігання, що були після неї скасовуються.

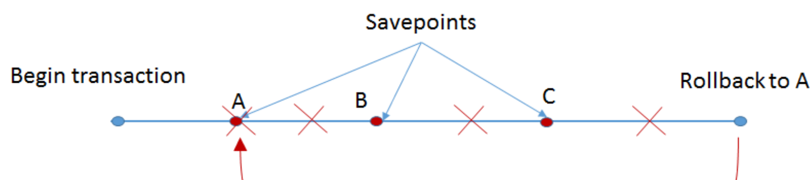


Рис. 8.2. Часткове скасування змін в транзакції

8.2. Базові властивості транзакцій

З міркувань продуктивності, СКБД повинна чередувати дії декількох транзакцій. Однак чередування виконується обережно, щоб гарантувати, що результат паралельного виконання транзакцій є еквівалентним (за своїм впливом на базу даних) деякому послідовному або почерговому виконанню того ж набору транзакцій. Те, як СКБД обробляє паралельне виконання транзакцій, є важливим аспектом управління транзакціями і предметом контролю паралелізму. Тісно пов'язане з цим питання про те, як СКБД обробляє часткові транзакції, або транзакції, які перериваються до їх нормального завершення. СКБД гарантує, що зміни, зроблені такими частковими транзакціями, не будуть помічені іншими транзакціями. Як це досягається, є предметом відновлення після збоїв.

Для збереження даних в умовах паралельного доступу та системних збоїв, СКБД повинна забезпечувати чотири важливі властивості транзакцій:

Атомарність (Atomicity). Користувачі повинні мати можливість розглядати виконання кожної транзакції як атомарне – або всі дії виконуються, або жодної.

Користувачі не повинні турбуватися про наслідки незавершених транзакцій (скажімо, при збої системи).

Транзакції можуть бути незавершеними з трьох причин. По-перше, транзакція може бути перервана або невдало завершена СКБД через те, що під час виконання виникла якась аномалія. Якщо транзакція переривається СКБД з якоїсь внутрішньої причини, вона автоматично перезапускається і виконується заново. По-друге, система може вийти з ладу (наприклад, через перебої з електроживленням) під час виконання однієї або декількох транзакцій. По-третє, транзакція може зіткнутися з неочікуваною ситуацією (наприклад, прочитати неочікуване значення даних або не мати доступу до якогось диска) і прийняти рішення про переривання.

Звичайно, транзакція, яка переривається посередині, може залишити базу даних у неузгодженому стані. Тому СКБД повинна знайти спосіб видалити з бази даних наслідки часткових транзакцій. Тобто, вона повинна забезпечити атомарність транзакцій: або виконуються всі дії транзакції, або не виконуються жодні.

Узгодженість або несуперечливість (Consistency). Кожна транзакція, що виконується сама по собі, без одночасного виконання інших транзакцій, повинна зберігати цілісність бази даних. Якщо кожна транзакція переводить узгоджений екземпляр бази даних в інший узгоджений екземпляр бази даних, то виконання декількох транзакцій (над узгодженим початковим екземпляром бази даних) призводить до узгодженого кінцевого екземпляра бази даних.

Ізольованість (Isolation). Користувачі повинні мати результат виконання транзакції без урахування впливу інших транзакцій, що виконуються паралельно, навіть якщо СКБД з міркувань продуктивності чередує дії декількох транзакцій. Цю властивість називають ізоляцією: Транзакції ізольовані, або захищені, від впливу інших транзакцій, що паралельно виконуються.

Властивість ізольованості забезпечується тим, що навіть якщо дії декількох транзакцій можуть чергуватися, чистий ефект буде ідентичним виконанню всіх транзакцій одна за одною в певному послідовному порядку. Наприклад, якщо дві транзакції $\mathbf{T_1}$ і $\mathbf{T_2}$ виконуються одночасно, чистий ефект гарантовано буде еквівалентний виконанню (всієї) $\mathbf{T_1}$ з подальшим виконанням $\mathbf{T_2}$ або виконанню $\mathbf{T_2}$ з подальшим виконанням $\mathbf{T_1}$ (СКБД не надає жодних гарантій щодо того, який з цих порядків буде ефективно обрано).

Довговічність (Durability). Після того, як СКБД повідомить користувача про успішне завершення транзакції, її наслідки повинні зберігатися, навіть якщо система вийде з ладу до того, як всі її зміни будуть відображені на диску. Ця властивість називається довговічністю.

СКБД забезпечує атомарність транзакцій, скасовуючи дії незавершених транзакцій. Щоб мати таку можливість, СКБД веде запис в журналі про всі операції з базою даних. Журнал також використовується для забезпечення довговічності: Якщо система виходить з ладу до того, як зміни, внесені завершеною транзакцією, будуть записані на диск, журнал використовується для запам’ятовування і відновлення цих змін при перезавантаженні системи.

Для позначення цих чотирьох властивостей транзакцій використовують абревіатуру **ACID**: атомарність, узгодженість, ізольованість і довговічність.

Компонент СКБД, який забезпечує атомарність та довговічність, називається менеджером відновлення.

8.3. Проблеми одночасного доступу до даних

Однією з головних цілей розробки бази даних є створення інформаційного ресурсу, який може бути спільним для багатьох користувачів. Якщо транзакції виконуються одна за одною, послідовно, причому кожна транзакція фіксується перед початком наступної, не виникає проблеми втручання в роботу інших транзакцій. Однак, користувачам часто потрібно отримати доступ до даних одночасно. Завданням системи контролю паралелізму є забезпечення якомога більшої кількості одночасної обробки, без втручання транзакцій одна в одну. Якщо всі користувачі тільки читають дані, вони не можуть заважати один одному, і немає необхідності в управлінні паралелізмом. Однак, коли два користувачі намагаються одночасно змінити одні і ті ж дані, або один оновлює, а інший читає ті ж дані, можуть виникнути конфлікти.

Деякі приклади потенційних проблем, спричинених паралелізмом: проблема втраченого оновлення, проблема нефіксованого оновлення, проблема непослідовного аналізу, проблема неповторюваного читання та проблема фантомних даних.

Для демонстрації проблем що виникають, розглянемо дві транзакції, $\mathbf{T_1}$ і $\mathbf{T_2}$, які працюють з деякими об’єктами бази даних, скажімо балансовим рахунком BAL.

Втрата результатів оновлення (Lost Update)

Час	Транзакція T_1	Транзакція T_2	Стан рахунку BAL
t1	BEGIN TRANSACTION		1000
t2	READ BAL (1000)	BEGIN TRANSACTION	1000
t3	. . .	READ BAL (1000)	1000
t4	BAL = BAL – 50 (встановити BAL 950)	. . .	1000
t5	WRITE BAL (записали 950)	BAL = BAL + 100 (встановити BAL 1100)	950
t6	COMMIT	. . .	950
t7		WRITE BAL (записали 1100)	1100
t8		COMMIT	1100

Рис. 8.3. Втрата результатів оновлення

Послідовність операцій транзакцій $\mathbf{T_1}$ і $\mathbf{T_2}$ наведена на рис. 8.3. Спочатку дані оновлює транзакція $\mathbf{T_1}$, а слідом за нею транзакція $\mathbf{T_2}$. Після закінчення обох транзакцій, рядок Р містить значення $\mathbf{P_2}$, занесене більш пізньої транзакцією $\mathbf{T_2}$. Транзакція $\mathbf{T_1}$ нічого не знає про існування транзакції $\mathbf{T_2}$, і природно очікує, що на рахунку міститься значення 950. Таким чином, транзакція $\mathbf{T_1}$ втратила результати своєї роботи.

Оскільки транзакції не ізольовані один від одної, кілька транзакцій можуть зчитувати і змінювати одні й ті ж дані. В результаті всі зміни крім останнього будуть втрачені.

Чорнове читання (Dirty read)

Час	Транзакція T_1	Транзакція T_2	Стан рахунку BAL
t1	BEGIN TRANSACTION		1000
t2	READ BAL (1000)		1000
t3	BAL = BAL + 100 (встановити BAL 1100)		1000
t4	WRITE BAL (записали 1100)	BEGIN TRANSACTION	1100
t5	. . .	READ BAL (1100)	1100
t6	ROLLBACK	. . .	1000
		COMMIT	1000

Рис. 8.4. Чорнове читання

Приклад чорнового читання наведено на рис. 8.4. Транзакція $\mathbf{T_1}$ змінює дані в рахунку. Після цього транзакція $\mathbf{T_2}$ читає змінені дані і працює з ними. Транзакція $\mathbf{T_1}$ відкочується і відновлює старі дані.

Одні і ті ж дані будуть одночасно читати і змінюватися різними транзакціями. Отримана інформація може виявитися суперечливою, оскільки:

- транзакція $\mathbf{T_1}$ ще не закінчила всі зміни даних (наприклад, коригування в зв'язаних таблицях – частина даних вже змінилася, а частина ні);
- оскільки транзакція $\mathbf{T_1}$ ще не закінчена, то взагалі не очевидно, що вона буде зафіксована. У разі помилки станеться її відкат. Вийде так, що інша транзакція прочитала модифіковані дані, які ні насправді не змінилися.

В результаті, транзакція $\mathbf{T_2}$ в своїй роботі використовувала дані, яких немає, і не було в базі даних! Дійсно, після відкату транзакції $\mathbf{T_1}$, повинна відновитися ситуація, так якщо б транзакція $\mathbf{T_1}$ взагалі ніколи не виконувалася.

Зчитування що не повторюється (Unrepeatable read)

Такий від порушення цілісності відбувається, коли перша транзакція читає елемент, інша транзакція записує нове значення для цього елемента, а потім перша транзакція повторно читає елемент і отримує значення, відмінне від першого.

Час	Транзакція T_1	Транзакція T_2	Стан рахунку BAL
t1	BEGIN TRANSACTION		1000
t2	READ BAL (1000)	BEGIN TRANSACTION	1000
t3	. . .	READ BAL (1000)	1000
t4	. . .	BAL = BAL + 100	1000
t5	. . .	COMMIT	1100
t6	READ BAL (1100)		1100
t7	COMMIT		1100

Рис. 8.5. Читання що не повторюється

Приклад читання що не повторюється наведено на рис. 8.5. Транзакція **T_1** двічі читає один і той ж рядок. Між цими читаннями вклинюється транзакція **T_2**, яка змінює значення в рядку. Транзакція **T_1** нічого не знає про існування транзакції **T_2**, і тому очікує, що після повторного читання значення буде тим же самим. В результаті транзакція **T_1** працює з даними, які, з точки зору транзакції **T_1**, мимоволі змінюються.

Фантомні зчитування (Phantom data)

Це відбувається, коли перша транзакція зчитує набір рядків, інша транзакція вставляє (або видаляє) рядки, а потім перша транзакція знову зчитує рядки і бачить новий рядок.

Ефект фіктивних елементів відрізняється від попередніх тим, що тут за один крок виконується досить багато операцій – читання одночасно декількох рядків, що задовольняють деякий умові.

Час	Транзакція T_1	Транзакція T_2
t1	BEGIN TRANSACTION	
t2	READ BAL WHERE α (зчитано ***n*** рядків за умовою α)	BEGIN TRANSACTION
t3	. . .	Додали 1 рядок, що відповідає умові α
t4	. . .	COMMIT
t5	READ BAL WHERE α (зчитано ***n+1*** рядок за умовою α)	
t6	COMMIT	

Рис. 8.6. Фантомне читання

Приклад фантомного читання наведено на рис. 8.6. Транзакція **T_1** двічі виконує вибірку рядків за однією і той же умовою. Між вибірками вклинюється транзакція **T_2**, яка додає новий рядок, що задовольняє умові відбору. Транзакція **T_1** нічого не знає про існування транзакції **T_2**, сама вона не змінює нічого в базі даних, і очікує, що після повторного відбору будуть отримані ті ж самі рядки.

В результаті транзакція **T**$_1$ в двох однакових вибірках рядків отримала різні результати.

Власне несумісний аналіз (Inconsistent Analysis)

Проблема неузгодженого аналізу виникає, коли транзакція зчитує кілька значень, а друга транзакція оновлює деякі з них під час виконання першої. Відрізняється від попередніх прикладів тим, що виконуються дві транзакції – одна довга, інша коротка.

Приклад несумісного аналізу наведено на рис. 8.7.

Час	Транзакція T_1	Транзакція T_2	BAL_A	BAL_B	BAL_C	SUM
t1	BEGIN TRANSACTION		5000	5000	5000	-
t2	SUM = 0	BEGIN TRANSACTION	5000	5000	5000	-
t3	READ BAL_A (5000)	. . .	5000	5000	5000	-
t4	SUM = SUM +5000 (5000)	READ BAL_A (5000)	5000	5000	5000	-
t5	READ BAL_B (5000)	BAL_A = BAL_A – 1000 (4000)	5000	5000	5000	-
t6	SUM = SUM +5000 (10000)	WRITE BAL_A (4000)	4000	5000	5000	-
t7	. . .	READ BAL_C (5000)	4000	5000	5000	-
t8	. . .	BAL_C = BAL_AC+ 1000 (6000)	4000	5000	5000	
t9	. . .	WRITE BAL_A (6000)	4000	5000	6000	
t10	READ BAL_C (6000)	COMMIT	4000	5000	6000	
t11	SUM = SUM +6000 (16000)		4000	5000	6000	
t12	WRITE SUM (16000)		4000	5000	6000	16000
t13	COMMIT		4000	5000	6000	16000

Рис. 8.7. Власне несумісний аналіз

Припустимо, що ми хочемо знайти загальний баланс усіх ощадних рахунків, але дозволяємо вносити, знімати та переказувати кошти під час виконання цієї транзакції. На рис. 8.6 наведено графік, який показує, як може бути отримано неправильний результат. Довга транзакція **T**$_1$ знаходить суму залишків на всіх рахунках. Ця транзакція не оновлює рахунки, а лише зчитує їх. Тим часом, коротка транзакція **T**$_2$ переказує 1000 з рахунку A на рахунок C. Транзакція **T**$_2$ виконується правильно і фіксується. Однак вона заважає транзакції **T**$_1$, яка додає ту саму 1000 двічі – один раз, коли вона знаходиться на рахунку A, і другий раз, коли вона потрапляє на рахунок C.

В результаті, хоча транзакція **T**$_2$ все зробила правильно – гроші переведені без втрати, але в результаті транзакція **T**$_1$ підрахувала невірну загальну суму.

8.4. Паралельне виконання транзакцій

8.4.1. Поняття серіалізації транзакцій

СКБД може підтримувати декілька поточних транзакцій одночасно. Транзакції запускаються відповідно до деякого графіку, що називаються розкладами (schedule). Розклад представляє хронологічний порядок, в якому виконуються інструкції. Очевидно, що розклад для набору транзакцій повинен зберігати порядок, в якому інструкції з'являються в кожній окремій транзакції.

Якщо контроль над паралельним виконанням повністю покласти на операційну систему, то можлива поява багатьох можливих розкладів, у тому числі таких, що залишають базу даних у неузгодженому стані, наприклад, як щойно описані.

Послідовне (serial) виконання транзакцій означає, що транзакції виконуються одна за одною, без будь-яких перерв між ними. Якщо A і B – це дві транзакції, то існує лише два можливих варіанти послідовного виконання: спочатку виконується вся транзакція A, а потім вся транзакція B, або спочатку виконується вся транзакція B, а потім вся транзакція A. При послідовному виконанні немає ніяких перешкод між транзакціями, тому що в кожен момент часу виконується тільки одна з них. Однак немає гарантії, що результати послідовного виконання всіх транзакцій заданого набору, будуть ідентичними. У банківській справі, наприклад, має значення, чи нараховуються відсотки на рахунок до того, як на нього покладено великий депозит, чи після. Однак, з точки зору коректності бази даних, порядок не має значення, оскільки будь-який порядок призведе до узгодженого стану бази даних.

Основна мета – знайти способи дозволити транзакціям виконуватися паралельно, при цьому переконавшись, що вони не заважають одна одній, і створити стан бази даних, який можна було б отримати при дійсно послідовному виконанні. Необхідно максимізувати паралельність, щоб максимізувати пропускну здатність, тобто кількість роботи, виконаної за певний проміжок часу, зберігаючи при цьому узгодженість бази даних.

Можна забезпечити узгодженість бази даних при паралельному виконанні, переконавшись, що будь-який виконаний розклад має такий самий ефект, як і розклад, який міг би відбутися без паралельного виконання. Тобто, розклад має бути, у певному сенсі, еквівалентним послідовному розкладу. Такі розклади називаються серіалізованими розкладами.

Для серіалізованого розкладу важливі наступні фактори:

- Якщо дві транзакції лише читають елементи даних, вони не конфліктують і порядок їх виконання не є важливим.
- Якщо дві транзакції оперують (або читають, або записують) абсолютно окремими елементами даних, вони не конфліктують і порядок не важливий.
- Якщо одна транзакція записує елемент даних, а інша або читає, або записує той самий елемент даних, то порядок виконання є важливим.

Отже, дві операції конфліктують лише тоді, коли всі наступні умови є істинними:

- Операції належать до різних транзакцій.
- Операції отримують доступ до одного і того ж елементу даних.
- Принаймні одна операція записує цей елемент.

Серіалізованість може бути досягнута кількома способами. Загалом, СКБД має підсистему управління паралелізмом, яка є «частиною пакета» і не контролюється безпосередньо ні користувачами, ні адміністратором бази даних.

8.4.2. Рівні ізоляції транзакцій

Серіалізованість є корисною концепцією, оскільки вона дозволяє програмістам ігнорувати проблеми, пов'язані з паралелізмом, коли вони кодують транзакції. Якщо кожна транзакція має властивість зберігає узгодженість бази даних, якщо виконується окремо, то серіалізованість гарантує, що одночасне виконання зберігає узгодженість. Однак, протоколи, необхідні для забезпечення серіалізованості, можуть допускати занадто малий рівень паралелізму для певних програм. У таких випадках використовуються слабші рівні узгодженості. Використання слабших рівнів узгодженості накладає додаткове навантаження на програмістів для забезпечення коректності бази даних.

Стандарт ANSI SQL (1992) дозволяє транзакції вказати, що вона може бути виконана таким чином, що вона стає не серіалізованою по відношенню до інших транзакцій. Стандарт визначає управління транзакціями на основі рівнів ізоляції транзакцій. Рівні ізоляції транзакцій означають ступінь, до якого дані транзакції «захищені або ізольовані» від інших паралельних транзакцій. Рівні ізоляції описуються на основі того, які дані можуть бачити (читати) інші транзакції під час виконання. Точніше, рівні ізоляції транзакцій описуються типом «читання», яке транзакція дозволяє або не дозволяє. В попередньому розділі були розглянуті такі типи операцій читання: чорнове читання, неповторне читання, фантомне читання.

На основі вищезазначених операцій ANSI визначив чотири рівні ізоляції транзакцій: Read Uncommitted, Read Committed, Repeatable Read і Serializable. На рис. 8.8 показано чотири рівні ізоляції транзакцій за стандартом ANSI. У таблиці також показано додатковий рівень ізоляції, що забезпечується базами даних Oracle і MS SQL Server.

	Рівень ізоляції	Чорнове читання	Читання що не повторюється	Фантомне читання
Менш обмежень ↕	Незафіксована операція читання (Read uncommitted)	Так	Так	Так
	Зафіксована операція читання (Read committed)	Ні	Так	Так
	Повторюване читання (Repeatable read)	Ні	Ні	Так
Більш обмежень	Упорядковування транзакцій (Serializable)	Ні	Ні	Ні
Oracle, SQL Server	Read Only / Snapshot	Ні	Ні	Ні

Рис. 8.8. Рівні ізоляції транзакцій

Рівень ізоляції Read Uncommitted дозволяє читати нефіксовані дані з інших транзакцій. На цьому рівні ізоляції база даних не накладає жодного блокування на дані, що підвищує продуктивність транзакцій, але за рахунок втрати узгодженості даних. Рівень Read Committed дозволяє транзакції читати лише зафіксовані дані. Це режим роботи за замовчуванням для більшості баз даних (включаючи Oracle і SQL Server). На цьому рівні база даних буде використовувати ексклюзивні блокування даних, змушуючи інші транзакції чекати, поки початкова транзакція не зафіксує дані. Рівень ізоляції Repeatable Read гарантує, що запити повертають узгоджені результати. Цей тип ізоляції використовує спільні блокування, щоб гарантувати, що інші транзакції не будуть

оновлювати рядок після того, як початковий запит його прочитає. Однак, нові рядки читаються (фантомне читання), оскільки ці рядки не існували під час виконання першого запиту. Рівень ізоляції Serializable є найсуворішим рівнем, визначеним стандартом ANSI SQL.

Формат команди встановлення визначеного рівня ізоляції:

```
SET TRANSACTION ISOLATION LEVEL [
{SERIALIZABLE|
REPEATABLE READ |
READ COMMITED |
READ UNCOMMITED }]
```

Приклад. Встановлення визначеного рівня ізоляції

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITED;
BEGIN TRANSACTION
SELECT Balance FROM Accounts WHERE AccounId = 1;
```

Результат – дані можна читати, навіть якщо інша транзакція змінює цей запис (чорнове читання).

```
SET TRANSACTION ISOLATION LEVEL READ COMMITED;
```

Результат – транзакція може читати тільки зафіксовані зміни, але результат може змінитись, якщо інша транзакція завершиться раніше поточної (Зчитування що не повторюється).

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Результат – Друга транзакція блокується, поки перша не завершиться.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Результат – Друга транзакція блокується, поки перша не завершиться, забезпечуючи відсутність її впливу на результат.

8.5. Механізми управління паралелізмом

Існують різні політики управління паралелізмом, які можна використовувати, щоб гарантувати, що навіть при одночасному виконанні декількох транзакцій генеруються тільки прийнятні розклади, незалежно від того, як операційна система розподіляє ресурси (наприклад, процесорний час) між транзакціями.

Найбільш важливі механізми контролю паралелізму:

- Протоколи на основі блокування;
- Протоколи на основі міток часу;
- Протоколи на основі валідації.

8.5.1. Протоколи на основі блокування

Блокування забезпечує серіалізацію, дозволяючи транзакції заблокувати об’єкт бази даних, щоб інша транзакція не могла отримати доступ до об’єкта або змінити його. Блокувати можна об’єкти різного розміру – від усієї бази даних до окремого елемента даних. Розмір об’єкта визначає тонкість, або гранулярність, блокування. Фактичне блокування може бути реалізоване шляхом вставки прапорця в елемент даних, запис,

сторінку або файл, щоб вказати, що частина бази даних заблокована, або шляхом ведення списку заблокованих частин бази даних.

Зазвичай розрізняють дві категорії блокування: спільні (shared) та ексклюзивні (exclusive). Якщо система використовує блокування, то будь-яка транзакція, яка хоче отримати доступ до елемента даних, повинна спочатку заблокувати цей елемент, запитуючи спільне блокування для доступу тільки на читання або ексклюзивне блокування для доступу на запис. Якщо елемент ще не заблоковано іншою транзакцією, блокування буде надано. Якщо елемент наразі заблоковано, система визначить, чи сумісний запит з існуючим блокуванням. Якщо запитується спільне блокування для позиції, яка вже має спільне блокування, запит буде задоволено; в іншому випадку, транзакція буде змушена чекати, поки існуюче блокування не буде знято. Блокування транзакції автоматично знімаються, коли транзакція завершується. Крім того, іноді транзакція може явно зняти блокування, які вона мала, до завершення.

Існує декілька протоколів блокування. Кожен протокол складається з наступного:

1. Набору типів блокування.
2. Набору правил, які вказують, які блокування можуть бути надані одночасно.
3. Набір правил, яким повинні слідувати транзакції при встановленні та звільненні блокування.

Якщо набір правил заданий невдало, використання блокувань не вирішує розглянутих раніше проблем одночасного доступу. На рис. 8.10 позначено XL – це запит на ексклюзивне блокування, а UL – це зняття блокування. SL – це запит на спільне блокування, A і B об'єкти доступу. У прикладі, показаному на рис. 8.10a, результат, показаний транзакцією 2, є неправильним, оскільки вона змогла прочитати B до оновлення, але прочитала A тільки після оновлення. Тому наведений вище розклад не можна серіалізувати.

Проблема з цим розкладом полягає у не послідовному отриманні даних. Проблема виникає через те, що у наведеному прикладі є два RW-конфлікти. Перший пов'язаний з об'єктом A, в якому транзакція 1 логічно з'являється перед транзакцією 2, оскільки транзакція 1 читає A перед транзакцією 2. Другий конфлікт пов'язаний з об'єктом B, в якому транзакція 2 логічно з'являється перед транзакцією 1, оскільки транзакція 2 зчитує B перед транзакцією 1. Таким чином, розклад не можна серіалізувати.

Двофазний протокол блокування

Цей протокол також відомий як протокол 2PL. 2PL – це найпоширеніший метод контролю паралелізму в комерційних базах даних, який забезпечує серіалізацію. У цьому протоколі транзакція не може вимагати нового блокування після зняття блокування.

2PL допомагає усунути проблему паралелізму в СКБД. Цей протокол блокування ділить фазу виконання транзакції на три різні фази, які виглядають наступним чином (рис. 8.9):

- **Фаза зростання**. У фазі зростання транзакція може здійснити нове блокування елемента даних, але жодне не може бути звільнене. Підвищення блокування (від

блокування на читання до блокування на запис для будь-якого елемента даних) дозволено у фазі зростання.

- **Фаза виконання**. У фазі виконання транзакція реалізує операції транзакції.
- **Фаза зменшення**. На фазі зменшення існуючі блокування, що утримуються транзакцією, можуть бути звільнені, але нові блокування не можуть бути отримані. Пониження блокування (від блокування на запис до блокування на читання будь-якого елемента даних) повинно відбуватися у фазі зменшення.

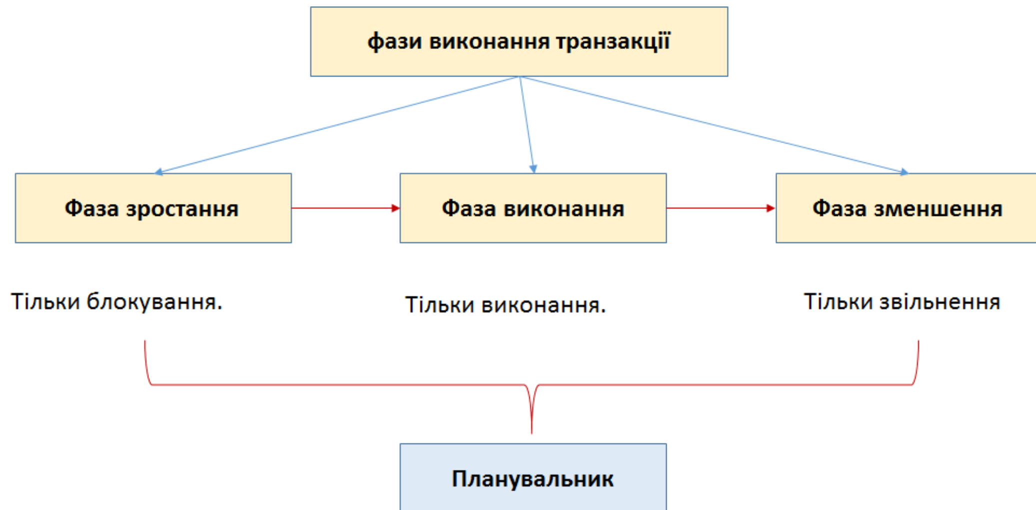


Рис. 8.9. Фази виконання протоколу 2PL

Існує три різні варіанти протоколів двофазного блокування: базовий, суворий, жорсткий. Базовий 2PL має фази зростання та зменшення. У фазі зростання та зменшення він не знімає та не отримує блокування на жодному елементі даних. У строгому 2PL він буде тримати блокування до завершення запису, а в жорсткому 2PL триматиме всі блокування до завершення транзакції або її переривання.

Транзакція 1	Час	Транзакція 2
XL(A)	1	–
Read(A)	2	–
A := A – 50	3	–
–	4	SL(B)
Write(A)	5	–
–	6	Read(B)
UL(A)	7	–
–	8	UL(B)
–	9	SL(A)
XL(B)	10	–
–	11	Read(A)
Read(B)	12	–
–	13	UL(A)
B := B + 50	14	–
–	15	Display (A + B)
Write (B)	16	–
UL(B)	17	–

a

Транзакція 1	Час	Транзакція 2
XL(A)	1	–
Read(A)	2	–
A := A – 50	3	–
–	4	SL(B)
Write(A)	5	–
–	6	Read(B)
XL(B)	7	–
UL(A)	8	–
–	9	SL(A)
–	10	UL(B)
–	11	Read(A)
Read(B)	12	–
–	13	UL(A)
B := B + 50	14	–
–	15	Display (A + B)
Write (B)	16	–
UL(B)	17	–

b

Рис. 8.10. Порівняння правил блокування

Використовуючи метод 2PL, можна модифікувати розклад, представлений на рис. 8.10a. Позначки на рисунку: XL означає X-lock, SL означає S-lock і UL означає unlocked. Модифікований розклад наведено на рис. 8.10b. Цей розклад отримує всі блокування до того, як будь-який блокування буде звільнено.

Привабливість алгоритму 2PL випливає з теореми, яка доводить, що 2PL завжди призводить до серіалізованих розкладів, які еквівалентні послідовним розкладам у порядку, в якому кожна транзакція отримує своє останнє блокування. Це є достатньою умовою серіалізованості, хоча і не обов’язковою.

На жаль, хоча наведений вище розклад не призведе до неправильних результатів, він має іншу складність, а саме, обидві транзакції блокуються в очікуванні одна одної. Транзакція 1 на рис. 8.10b чекає (на кроці 7), поки транзакція 2 розблокує B, в той час як транзакція 2 не може цього зробити, поки не отримає блокування на A (на кроці 9). Виникла класична ситуація тупика (deadlock) Тупики виникають через умови кругового очікування, в яких беруть участь дві або більше транзакцій, як описано вище. Система, яка не допускає виникнення тупиків, називається безтупиковою. У двофазному блокуванні потрібен механізм виявлення тупиків і схема виходу з тупика після його виникнення.

8.5.2. Тупикове блокування (deadlock)

У СКБД тупикова ситуація (deadlock) – це небажаний стан, який зазвичай виникає в середовищі зі спільними ресурсами, коли транзакція невизначено довго чекає на ресурс, який зайнятий іншою транзакцією. Таким чином, всі транзакції чекають одна на одну, щоб звільнити ресурси, і жодна з них не може завершити своє завдання.

Розглянемо концепцію тупика на наступному прикладі (рис. 8.11):

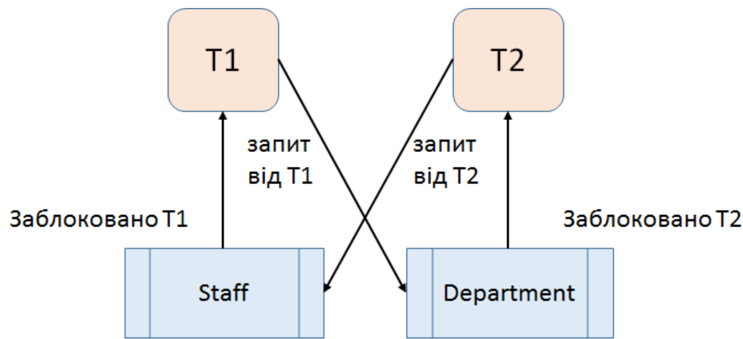


Рис. 8.11. Приклад тупикової ситуації

Нехай є дві транзакції $\mathbf{T_1}$ і $\mathbf{T_2}$. Припустимо, що транзакція $\mathbf{T_1}$ має блокування на таблиці Staff і їй потрібно додати кілька рядків до таблиці Department. У той же час, транзакція $\mathbf{T_2}$ утримує блокування на таблиці Department, але хоче отримати доступ до кількох рядків таблиці Staff, що утримується транзакцією $\mathbf{T_1}$, як показано на рис. 8.8. У цій ситуації обидві транзакції чекають одна на одну, щоб зняти блокування з таблиць, і жодна з них не може завершити своє завдання. В результаті, вони чекають невизначений час на ресурс і залишаються бездіяльними назавжди, якщо СКБД не визначить це як тупикову ситуацію і не вживе відповідних заходів, наприклад, не припинить одну з транзакцій.

Рівень ізоляції Serializable є найсуворішим рівнем, визначеним стандартом SQL. Однак, важливо зазначити, що навіть з рівнем ізоляції Serializable можливі тупикові ситуації. Більшість баз даних використовують підхід до управління транзакціями на основі виявлення тупикових ситуацій, а отже, вони виявлятимуть «тупикові ситуації» під час перевірки транзакції на коректність.

Зазвичай тупики виявляються шляхом побудови та аналізу графа очікування (WFG), як показано на рис. 8.12. Виявлення тупиків може відбуватися кожного разу, коли запит на блокування транзакції не задовольняється. Це називається безперервним виявленням. З іншого боку, можна віддати перевагу періодичній схемі виявлення, наприклад, раз на секунду, хоча результати експериментів вказують на те, що безперервна схема є більш ефективною.

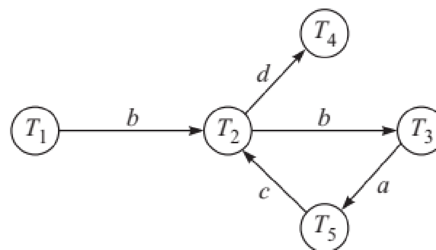


Рис. 8.12. Приклад графа очікування

Коли транзакція T_i запитує елемент даних, який в даний момент знаходиться у T_j, то ребро T_i до T_j вставляється у граф очікування. Це ребро видаляється тільки тоді, коли T_j більше не тримає елемент даних, необхідний T_i. Система на рис. 8.12 знаходиться у стані тупика, оскільки граф очікування має цикл, оскільки T_1 чекає на T_2, який чекає на T_3 і T_4, а T_3 чекає на T_5, який, у свою чергу, чекає на T_2. Літери на ребрах графу позначають назви

об'єктів, на які очікують транзакції. Наприклад, T_1 чекає на T_2 для об'єкта b. Таким чином, існує циклічне очікування між транзакціями T_2, T_3 і T_5.

Як зазначалося раніше, тупик можна виявити, лише періодично перевіряючи граф очікування на наявність циклів.

Безумовно, тупик є небажаним, але при паралельному управлінні, заснованому на блокуванні, тупики часто є неминучими. Двофазне блокування не позбавлене тупиків. Тому існує декілька аспектів роботи з тупиками, а саме: їх слід запобігати, якщо це можливо (запобігання тупикам), виявляти їх, коли вони виникають (виявлення тупиків) і вирішувати їх, коли тупик було виявлено (вирішення тупиків). Тупики можуть включати дві, три або більше транзакцій. Для вирішення тупиків необхідно відстежувати, які транзакції чекають і на яку транзакцію вони чекають.

Як тільки тупик виявлено, необхідно вибрати одну з транзакцій, що потрапила в тупик, і відкотити її (така транзакція називається жертвою), тим самим звільнивши всі блокування, які тримала ця транзакція, і таким чином вийти з тупика.

Спочатку обговоримо запобігання виникненню тупиків, а потім – виявлення та вирішення тупикових ситуацій.

Запобігання виникненню тупикових ситуацій

Запобігання тупикам передбачає забезпечення того, щоб не виникало циклічного очікування. Це можна зробити або визначивши порядок того, хто кого може чекати, або усунувши будь-яке очікування. Визначення порядку виконання транзакцій здійснюється за допомогою алгоритмів очікування-відмови або старший-молодший.

Алгоритм очікування-відмови (Wait-die). Коли виникає конфлікт між $\mathbf{T_1}$ і $\mathbf{T_2}$ ($\mathbf{T_1}$ – старша транзакція), якщо $\mathbf{T_1}$ володіє блокуванням, то $\mathbf{T_2}$ (молодша транзакція) не може чекати на ресурс, на який $\mathbf{T_1}$ має блокування. $\mathbf{T_2}$ повинна відкотитися і перезапуститися. Якщо, однак, $\mathbf{T_2}$ володів блокуванням під час конфлікту, то старшій транзакції дозволяється чекати. Таким чином, метод дозволяє уникнути циклічних очікувань і, як правило, уникає нездатності транзакції отримати необхідні їй ресурси. Однак старша транзакція може виявити, що їй доводиться чекати на кожен ресурс, який їй потрібен, і це може зайняти деякий час для завершення. Час очікування для транзакції можна задати командою

```
SET LOCK_TIMEOUT {Millisecond}
```

Алгоритм старший-молодшій (Wound-die). Щоб уникнути ситуації, коли старшій транзакції доводиться чекати на кожен елемент даних, який їй потрібен, схема wound-die дозволяє старшій транзакції негайно отримати необхідний їй елемент даних, навіть якщо він знаходиться під контролем молодшій транзакції, який має на ньому блокування. Молодша транзакція перезапускається. Молодшій транзакції, який очікує на блокування, що утримується старшою транзакцією, дозволяється зачекати.

Виявлення та вирішення тупикових ситуацій

Після виявлення тупика за допомогою WFG, його необхідно вирішити. Найпоширеніший метод розв'язання тупику вимагає, щоб одна з транзакцій у графі

очікування була обрана як жертва, відкотилася назад і була перезапущена. Жертва може бути обрана на основі одного з наступних критеріїв:

1. Випадково – транзакція з циклу вибирається випадковим чином.
2. Остання заблокована – транзакція, яка була заблокована останньою.
3. Наймолодша – оскільки ця транзакція, ймовірно, виконала найменшу кількість роботи.
4. Мінімальна робота – транзакція, яка записала найменше даних до бази даних, оскільки всі записані дані повинні бути скасовані при відкаті.
5. Найменший вплив – транзакція, яка, ймовірно, вплине на найменшу кількість інших транзакцій, тобто транзакція, відкат якої призведе до найменшої кількості інших відкатів (при каскадному відкаті).
6. Найменше блокувань – транзакція, яка має найменшу кількість блокувань.

Було виявлено, що техніка, яка вибирає жертву з найменшою кількістю блокувань для виходу з тупику, забезпечує найкращу продуктивність.

8.5.3. Інші протоколи контролю паралелізму

Протоколи на основі міток часу

Цей протокол використовує концепцію мітки часу для впорядкування транзакцій і застосовує так зване оптимістичне блокування. Оптимістичне блокування передбачає, що ймовірність конфліктів низька і більшість транзакцій не впливає одна на одну. Транзакції не блокують ресурси відразу, а роблять це під час фіксації. Якщо інша транзакція модифікувала дані, після того як перша її прочитала, перша транзакція відкочується.

Загалом, мітка часу може бути встановлена як системний час або логічний лічильник (номер версії). Транзакції виконуються у порядку зростання часових міток.

У цьому методі старша транзакція завжди має пріоритет над молодшою. Наприклад, припустимо, що є три транзакції, $\mathbf{T_1}$, $\mathbf{T_2}$ і $\mathbf{T_3}$. Транзакція $\mathbf{T_1}$ входить в систему в час 0010, $\mathbf{T_2}$ входить в систему в час 0040, а $\mathbf{T_3}$ входить в систему в час 0050. Тут $\mathbf{T_1}$ є найстарішою транзакцією і має найвищий пріоритет, тому виконується першою. Аналогічно, $\mathbf{T_2}$ отримує шанс на виконання, а потім $\mathbf{T_3}$, як наймолодша транзакція, буде виконана останньою.

В методі також зберігаються мітки часу останньої операції читання та запису над елементами даних бази даних, як показано нижче:

- W-timestamp(X) – означає найбільшу мітку часу будь-якої транзакції, яка успішно виконала операцію запису на 'X'.
- R-timestamp(X) – означає найбільшу мітку часу будь-якої транзакції, яка успішно виконала операцію читання на 'X'.

Ці мітки оновлюються щоразу, коли виконується нова операція читання або запису над елементом даних 'X'.

Приклад. Транзакція 1 читає залишок на Рахунку 1 і починає його змінювати. Транзакція 2 читає залишок на Рахунку 1 і також починає його змінювати.

```
-- Транзакція 1
```

```sql
SELECT Balance, Version FROM Accounts WHERE AccountId = 1
-- Транзакція 2
SELECT Balance, Version FROM Accounts WHERE AccountId = 1
-- Обидві транзакції зчитали номер версії = 10
-- Транзакція 1 змінює стан рахунку
UPDATE Accounts SET Balance = Balance - 100, Version = Version + 1
WHERE AccountId = 1 AND Version = 10;
-- Транзакція 2 також намагається змінити стан рахунку
UPDATE Accounts SET Balance = Balance - 100, Version = Version + 1
WHERE AccountId = 1 AND Version = 10;
```

Транзакція 2 відкочується внаслідок конфлікту міток.

Протокол впорядкування міток часу забезпечує серіалізованість. Він також гарантує відсутність тупикових ситуацій, тобто жодна транзакція ніколи не чекає.

Протоколи на основі валідації

Протокол на основі валідації також використовує оптимістичний метод контролю паралелізму. У цій техніці під час виконання транзакції не виконується ніяких перевірок. Всі оновлення, що відбуваються під час виконання транзакцій, не застосовуються безпосередньо до бази даних, а застосовуються до локальних копій елементів даних для транзакцій. Коли транзакція досягає кінця виконання, відбувається перевірка, чи не порушує оновлення транзакції принципу серіалізованості. Якщо немає порушень серіалізованості, то транзакція фіксується і, нарешті, база даних оновлюється; в іншому випадку, транзакція перезапускається.

Цей протокол блокування ділить фазу виконання транзакції на три різні фази:

- **Фаза читання**. На цій фазі транзакція **T** зчитує зафіксовані елементи даних з бази даних і зберігає їх у тимчасових локальних змінних. Операції запису можуть бути виконані над тимчасовими змінними без оновлення реальної бази даних.
- **Фаза перевірки**. На цій фазі виконується перевірка значень тимчасових змінних, щоб переконатися у відсутності порушень серіалізованості.
- **Фаза запису**. Якщо немає порушень серіалізованості, то оновлені значення тимчасових локальних змінних записуються в базу даних; в іншому випадку оновлення відкидаються, а транзакція відкочується.

8.6. Відновлення після збою

8.6.1. Базові компоненти відновлення

Відновлення транзакцій це процес повернення бази даних з неузгодженого стану до узгодженого. Мета відновлення – встановити стан зовнішньої пам’яті, таким що виник би при фіксації в зовнішній пам’яті змін всіх транзакцій що завершилися і яке не містило б ніяких слідів незакінчених транзакцій.

Коректному виконанню транзакції можуть перешкоджати кілька типів збоїв. Їх можна розділити на три великі категорії відповідно до їхніх причин: збій транзакції, збій системи та збій носія.

Збій транзакції виникає внаслідок помилки в логіці, яка керує операціями транзакції (наприклад, неправильне введення, не ініціалізовані змінні, некоректні

оператори тощо) та/або в логіці програми. Як наслідок, транзакція не може бути успішно завершена і повинна бути перервана. Це рішення зазвичай приймається застосунком або самою системою баз даних. Якщо під час транзакції були внесені будь-які попередні зміни, їх слід відкотити.

Системний збій відбувається, якщо операційна система або СКБД виходить з ладу, наприклад, через помилку або відключення електроенергії. Це може призвести до втрати вмісту первинного сховища і, відповідно, буфера бази даних.

Збій диску (носія) відбувається, якщо зовнішнє сховище (жорсткий диск або в деяких випадках флеш-пам'ять), яке містить файли бази даних, і, можливо, файл журналу, пошкоджено або недоступне через збій диска, збій в мережі зберігання даних тощо. Хоча транзакція могла бути виконана правильно, а отже, програма або користувач, який ініціював транзакцію, був проінформований про те, що транзакція була успішно завершена, фізичні файли можуть бути не в змозі зафіксувати або відобразити оновлення, викликані транзакцією.

Система керування базами даних надає такі основні засоби для резервного копіювання та відновлення бази даних (рис. 8.13):

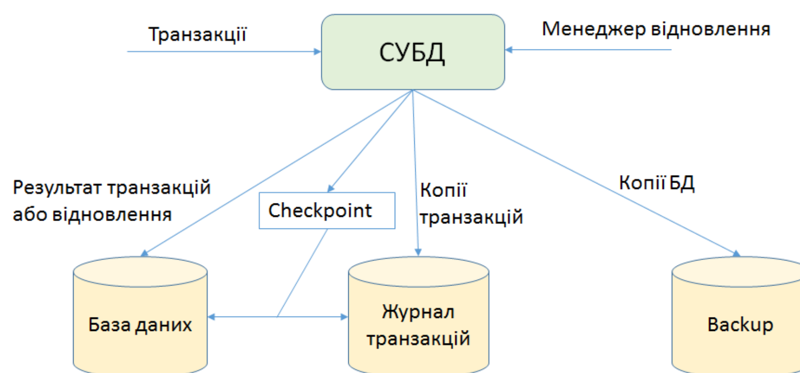


Рис. 8.13. Базові компоненти відновлення бази даних

Засоби резервного копіювання. Забезпечують періодичне резервне копіювання частин або всієї бази даних. На додаток до файлів бази даних, засоби резервного копіювання повинні створювати копії пов'язаних об'єктів бази даних, включаючи метадані (або системний каталог), індекси бази даних тощо.

Засоби ведення журналів змін у базі даних. У випадку збою, послідовний стан бази даних можна відновити, використовуючи інформацію в журналах разом з останньою повною резервною копією. Дані, які зазвичай записуються для кожної транзакції, включають ідентифікатор транзакції, дію або тип транзакції (наприклад, вставка), час транзакції, номер терміналу або ідентифікатор користувача, значення вхідних даних, таблицю і записи, до яких здійснювався доступ, записи, які модифікувалися, старі і нові значення полів.

Контрольні точки. Запис контрольної точки містить інформацію, необхідну для перезапуску системи, гарантуючи, що всі зміни, зроблені до проходження контрольної точки, були записані в довгострокове сховище. Контрольні точки слід робити часто

(скажімо, кілька разів на годину). При виникненні збоїв часто можна відновити обробку з останньої контрольної точки без повного відновлення бази даних.

Менеджер відновлення. Менеджер відновлення – це модуль СКБД, який відновлює базу даних до коректного стану після збою, а потім відновлює обробку запитів користувачів. Тип перезапуску залежить від характеру збою. Менеджер відновлення використовує журнали, а також резервну копію, якщо це необхідно.

8.6.2. Відновлення транзакцій

Відновлення після збою транзакції

Відкат транзакції після збою виконується наступним чином:

1. Журнал сканується у зворотному напрямку, для кожного знайденого запису транзакції створюється так званий компенсаційний запис і виконується операція, яка відновлює значення, що змінені під час транзакції.
2. Після того, як буде знайдено в журналі запис початку транзакції, зворотне сканування буде зупинене, і до журналу буде записано запис переривання транзакції.

Кожна дія оновлення, виконана транзакцією, включно з діями, виконаними для відновлення елементів даних до їхніх старих значень, тепер записана до журналу.

Відновлення після системного збою

В процедурі відновлення транзакцій після системного збою використовується такі засоби відновлення:

- Протокол с записом на випередження (write-ahead-log) гарантує, що журнали транзакцій завжди записуються до того, як будь-які дані в базі даних будуть оновлені. Цей протокол гарантує, що в разі збою базу даних можна буде пізніше відновити до узгодженого стану, використовуючи дані журналу транзакцій.
- Надлишкові журнали транзакцій (кілька копій журналу транзакцій) гарантують, що фізичний збій диска не вплине на здатність СКБД відновлювати дані.
- Буфери бази даних – це тимчасові області зберігання в первинній пам’яті, які використовуються для прискорення дискових операцій. Щоб зменшити час обробки, програмне забезпечення СКБД зчитує дані з фізичного диска і зберігає їх копію в «буфері» в первинній пам’яті. Коли транзакція оновлює дані, вона фактично оновлює копію даних у буфері, оскільки цей процес набагато швидший, ніж щоразу звертатися до фізичного диска. Пізніше всі буфери, що містять оновлені дані, записуються на фізичний диск за одну операцію, що значно економить час обробки.
- Контрольні точки бази даних – це операції, під час яких СКБД записує всі свої оновлені буфери в пам’яті на диск. Поки це відбувається, СКБД не виконує жодних інших запитів. Операція контрольної точки також реєструється в журналі транзакцій. В результаті цієї операції фізична база даних і журнал транзакцій будуть синхронізовані. Ця синхронізація необхідна, оскільки операції оновлення оновлюють копії даних у буферах, а не у фізичній базі даних. Фізично база даних

оновлюється тільки даними від зафіксованих транзакцій, використовуючи інформацію з журналу транзакцій.

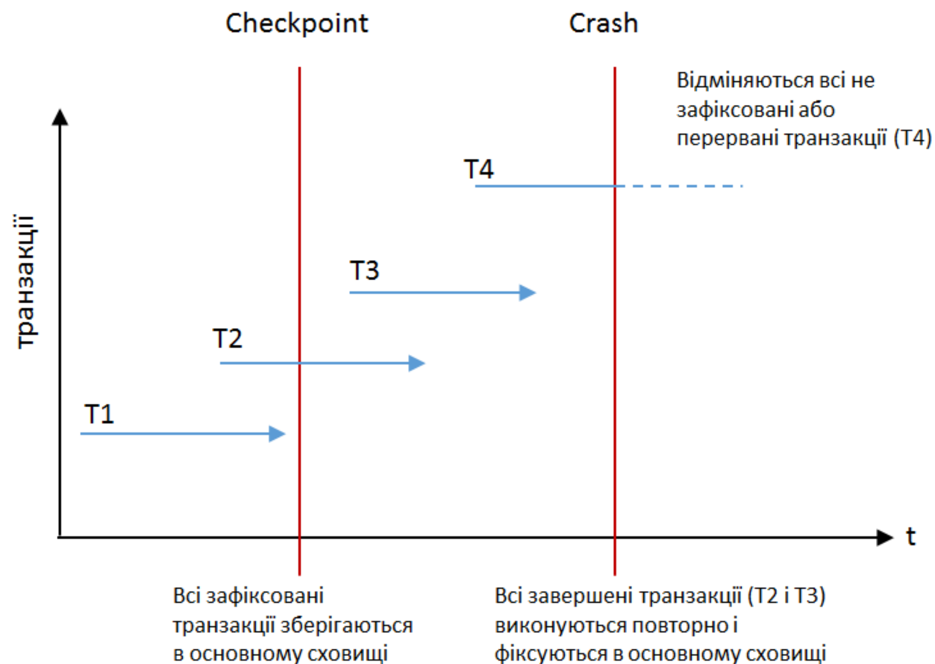


Рис. 8.14. Відновлення транзакція після системного збою

Процес відновлення наведено на рис 8.14. Для всіх розпочатих і зафіксованих транзакцій (до збою) він складається з наступних кроків:

1. Визнається остання контрольна точка в журналі транзакцій. Це останній раз, коли дані транзакцій були фізично збережені на диск.
2. Для транзакції, яка почалася і була зафіксована до останньої контрольної точки (**T_1**), нічого не потрібно робити, оскільки дані вже збережені.
3. Для транзакції, яка виконала операцію фіксації після останньої контрольної точки (**T_2 , T_3**), СКБД використовує записи журналу транзакцій, щоб повторити транзакцію і оновити базу даних, використовуючи значення в журналі транзакцій. Зміни вносяться у порядку зростання, від найстаріших до найновіших.
4. Для будь-якої транзакції, яка мала операцію `ROLLBACK` після останньої контрольної точки або яка залишалася активною (без `COMMIT` або `ROLLBACK`) до того, як стався збій (**T_4**), СКБД використовує записи журналу транзакцій для скасування операцій, використовуючи значення в журналі транзакцій. Зміни застосовуються у зворотному порядку, від найновіших до найстаріших.

8.6.3. Відновлення після збою носія

Відновлення після збою носія необхідне, якщо фізичні файли бази даних та/або файл журналу недоступні або пошкоджені через несправність носія або підсистеми зберігання. Незважаючи на те, що існує багато альтернатив, відновлення даних завжди базується на певному типі надлишковості даних: додаткові копії файлів або даних що зберігаються на автономних носіях (наприклад, на стрічковому сховищі) або онлайн-

носіях (наприклад, на резервному жорсткому диску в Інтернеті, або навіть у повноцінному резервному вузлі бази даних). Такі копії створюються за допомогою процедури резервного копіювання.

Типи резервного копіювання:

1. **Повне** резервне копіювання бази даних (full database backup).

До переваг повного резервного копіювання відноситься те, що копії не залежать одна від одної. Тобто втрата повної резервної копії, зробленої в один день не вплине на можливість відновлення з повної резервної копії, зробленої на другий день. Але час резервного копіювання є найбільшим для всіх типів рівнів резервного копіювання. Також, таке копіювання використовує максимально можливий обсяг носія і матиме найбільшу вартість.

2. **Інкрементальне** резервне копіювання (incremental database backup).

Зберігаються лише ті дані, які було змінено з часу останнього резервного копіювання (незалежно від того, на якому рівні воно було). Це оптимальний варіант копіювання, наприклад, коли база даних об'ємом 50 ТБ може змінюватись лише на 2-5% протягом дня. Режим резервного копіювання, що поєднує повне та інкрементальне резервне копіювання використовує значно менше носія. Час резервного копіювання значно коротше, ніж повного резервного копіювання. Можливий графік резервного копіювання показано на рис. 8.15.

Субота	Неділя	Понеділок	Вівторок	Середа	Четвер	П'ятниця
Full	Incr	Incr	Incr	Incr	Incr	Incr

Рис. 8.15. Приклад графіку інкрементального резервного копіювання

Але відновлення зі змішаного режиму повного та інкрементального резервного копіювання може вимагати більшої кількості змін носіїв (якщо використовується знімний носій, наприклад, стрічка). Також повне відновлення системи неможливо виконати без повних резервних копій і всіх інкрементальних резервних копій, створених в період між повним копіюванням і збоїм.

3. **Диференціальне** резервне копіювання (differential database backup).

Такий спосіб резервного копіювання зберігає ті дані, які було змінено з часу останнього повного резервного копіювання. Можливий графік диференціального резервного копіювання показано на рис. 8.16.

Просте диференціальне резервне копіювання просто створює резервну копію всіх змін, які відбулися з моменту останнього повного резервного копіювання, незалежно від того, які резервні копії було створено між ними.

Субота	Неділя	Понеділок	Вівторок	Середа	Четвер	П'ятниця
Full	Incr	Incr	Incr	**Diff**	Incr	Incr

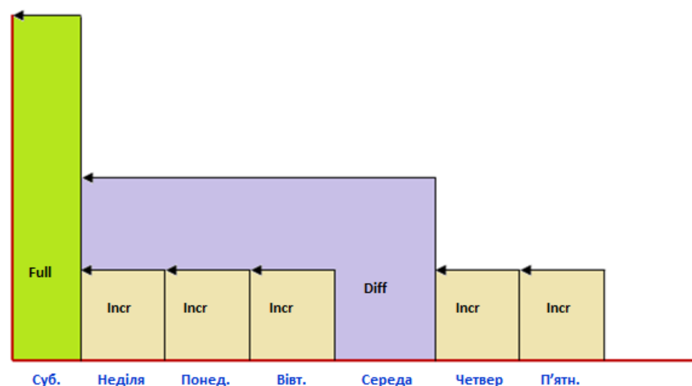


Рис. 8.16. Приклад графіку диференціального резервного копіювання

Багаторівневе диференціальне копіювання має різні рівні від 1 до 9. Диференціальний рівень x створює резервну копію змін з моменту останнього повного або останнього диференціального рівня з меншим або рівним номером.

Переваги диференціалів полягають у що зменшується кількості наборів резервних копій. Також, у стрічкових накопичувачах зменшується ризик того, що пошкоджена стрічка завадить повному відновленню (якщо вміст залишався незмінним протягом кількох днів, він з'явиться у кількох диференціальних резервних копіях).

Але методика диференціального копіювання по-різному інтерпретується постачальниками баз даних. Тому цей варіант створення копії бази даних використовується не часто.

Зберігання резервних копій

Основними носіями для зберігання резервних копій є:

- накопичувачі на жорстких дисках (HDD), якщо він знаходиться на іншому сервері;
- твердотільні накопичувачі (SSD) – хороше середовище для резервного копіювання;
- стрічкові накопичувачі (Tape drives) – відносно прості і як правило, служать тривалий час. Відновлення виконується повільніше.
- хмарний сервіс «Зберігання як послуга» (Storage as a Service) – користувачі хмари отримують доступ до дискових ферм. Оскільки послуга надається в оренду, авансові капітальні вкладення, необхідні для забезпечення резервних засобів, не потрібні, але витрати на обслуговування з часом накопичуються і цілком можуть випередити будь-які капітальні витрати, які могли бути понесені замість цього.

Для організації надійного зберігання резервних копій використовується **правило 3-2-1**. Це правило означає що для забезпечення надійного зберігання даних, необхідно мати як мінімум:

- **Три** резервні копії, що зберігаються в трьох фізично різних місцях;
- які мають бути збережені в **Двох** різних фізичних форматах зберігання. Зберігання копій у різних фізичних форматах має на меті знизити ймовірність одночасної

загибелі даних усіх копій через однорідний вплив (температура, єл. маг. імпульс…);

- причому **Одна** з копій має бути передана на позаофісне зберігання. Зберігання однієї копії поза офісом, вирішує те ж саме завдання (зниження статистичної залежності загроз різним копіям даних), тільки через географічний розподіл місць зберігання (пожар, крадіжка…).

8.7. Забезпечення доступності даних

Доступність бази даних один з основних показників безпеки бази даних. Формальне визначення доступності – це відсоток часу, який система може бути використана для продуктивної роботи.

Типові значення показників доступності наведені на рис. 8.17.

Доступність %	Час простою за рік	За місяць	За тиждень	Протягом доби
99% (“дві дев’ятки”)	3.65 доби	7.31 години	1.68 години	14.40 хвилини
99.9% (“три дев’ятки”)	8.77 години	43.83 хвилини	10.08 хвилини	1.44 хвилини
99.99% (“чотири дев’ятки”)	52.60 хвилини	4.38 хвилини	1.01 хвилини	8.64 секунди
99.999% (“п’ять дев’яток”)	5.26 хвилини	26.30 секунди	6.05 секунди	864.00 мілісекунди

Рис. 8.17. Показники доступності інформаційної системи

Існують три ключові аспекти забезпечення доступності бази даних:

1. Підтримання системи в робочому стані – негайне вирішення короткострокових проблем;
2. Планування резервного копіювання та відновлення;
3. Забезпечення високої доступності.

8.7.1. Підтримання системи в робочому стані

Розглянуті раніше питання відновлення транзакцій – це основний засіб вирішення короткострокових проблем. До цієї категорії можна віднести використання надлишкових масивів жорстких дисків. Використання надлишкових дискових масивів знижує ризик пошкодження основного сховища даних.

Резервний масив незалежних дисків (RAID) – це один із способів підвищення доступності. Більшість зовнішніх пристроїв зберігання даних підтримують RAID. Існує кілька типів RAID, відомих як рівні [2]. Залежно від використовуваного рівня, RAID може або підвищити продуктивність, або забезпечити вищу доступність.

Найчастіше використовуються такі рівні RAID: **RAID 1**, також відомий як дзеркальне відображення диска, і **RAID 5**. Рівень **RAID 1** є найпростішою формою реалізації високої доступності і потребує двох дисків. Якщо пара дисків доступна для паралельного читання, запити можуть виконуватися швидше, ніж до одного диска. Однак,

оскільки для зберігання даних потрібно вдвічі більше дискового простору, це не найефективніший механізм.

RAID 5 є популярним рівнем RAID для корпоративних систем, оскільки він пропонує кращу продуктивність, ніж **RAID 1**, а також високу доступність, і хоча він потребує використання щонайменше 3 дисків, він потенційно менш вимогливий до простору, оскільки не всі дані реплікуються. Це досягається за рахунок розміщення даних та інформації про парність на трьох (або більше) дисках. Окрім використання меншого дискового ресурсу, **RAID 5** може бути більш продуктивним, ніж **RAID 1**, оскільки ваші дані розподілені на більшій кількості дисків, а отже, з увімкненим паралельним читанням, потенційно можуть повертатися швидше. Недоліком, однак, є те, що запис буде повільнішим, ніж у **RAID 1**, оскільки для всіх даних, що записуються, потрібно буде виконувати перевірку на парність. Але для системи підтримки прийняття рішень він може виявитися дуже підходящим.

OLTP-системи з інтенсивним записом використовують **RAID 1** або комбінацію **RAID 1** і **RAID 0**, тобто **RAID 10**.

8.7.2. Планування резервного копіювання і відновлення

План аварійного відновлення визначає поточні та екстрені дії і процедури, необхідні для забезпечення доступності даних у разі виникнення катастрофи. Наприклад, план аварійного відновлення включає кроки для захисту даних організації від успішної кібератаки або фізичних збоїв, які можуть включати збої в роботі обладнання, втрату електроенергії або навіть повну втрату центру обробки даних внаслідок стихійного лиха. Ключові елементи успішного плану аварійного відновлення включають наступне:

Ключові елементи успішного плану аварійного відновлення включають наступні показники:

- **RPO** (recovery point objective) – середній період часу, протягом якого можна дозволити втрату даних, або як часто повинні виконуватися резервні копії працюючих додатків. Як правило, значення показника наведена в плані забезпечення безперервності бізнесу організації. **RPO** визначає максимальний вік даних або файлів у резервному сховищі, у разі збою інформаційної системи. Відповідно RPO визначає прийнятний інтервал резервного копіювання. Наприклад, **RPO** тривалістю 60 хвилин вимагає резервного копіювання системи кожні 60 хвилин. Якщо **RPO** становить 12 годин, а остання копія даних датується 10 годинами тому, система все ще відповідаємо параметрам RPO.
- **RTO** (recovery / recall time objective) – максимально допустимий час відновлення працездатності системи (максимальний час недоступності сервісу). для відновлення своїх процесів на прийнятному рівні обслуговування, щоб уникнути наслідків, пов'язаних із збоєм. **RTO** відповідає на запитання: «Скільки часу має пройти після повідомлення про порушення процесу обробки даних для відновлення нормальної роботи?».

Крім цих показників, використовуються ще такі:

- **Максимально допустимий час** простою (maximum tolerable downtime – MTD) для кожної бізнес-функції, який вказує на кількість часу, протягом якого бізнес може проіснувати без цієї функції.

  Приклади допустимих простоїв для типів функцій у бізнесі:

  – Неосновні послуги – 30 днів;
  – Звичайні пріоритетні послуги – 7 днів;
  – Важливі для бізнесу функції – 72 години;
  – Термінові функції – 24 години;
  – Критичні функції – протягом 3 годин.

- **Час відновлення роботи** (Work recovery time – WRT) – це максимальний проміжок часу, доступний для підтвердження функціональності та цілісності систем відновлювання і даних, щоб їх можна було знову запустити у процес обробки. RTO зазвичай має справу з відновленням інфраструктури та систем, а WRT – з тим, щоб користувачі могли повернутися до роботи з ними.

  Співвідношення показників наведено на рис. 8.18.

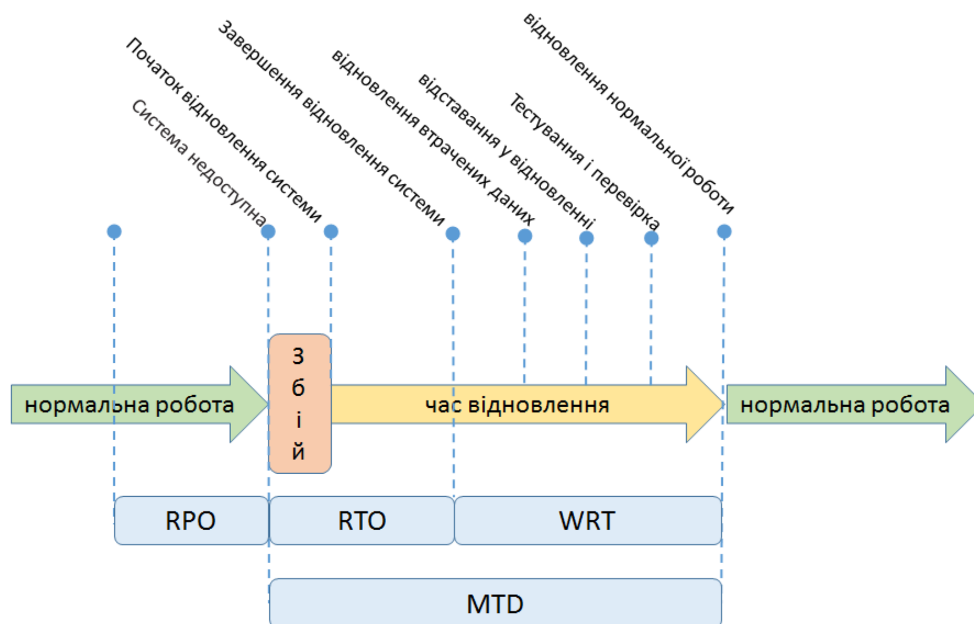


Рис. 8.18. Показники, що використовуються при аварійному відновленні

8.7.3. Висока доступність для критично важливих систем

Висока доступність (High Availability) – рівень доступності для ERP-систем, баз даних, поштових та інших сервісів, які забезпечують виробничі процеси. Коли сервіси стають недоступними, можлива втрата даних, що може істотно позначитися на ціні простою. Для цього рівня необхідний розрахунок метрик RTO і RPO. Наприклад, RTO = 1 – 4 години, RPO = 1 годину.

Постійна доступність (Continuously Available) потрібна для невеликої кількості сервісів, коли неприпустимо найменше час простою, таких як критично важливі SCADA і

MES-системи, бази даних для транспорту, безпеки, банків. Для таких систем RTO = 0, RPO = 0.

Для відновлення функціональності інформаційної системи високої доступності використовуються такі стратегії:

- **Гарячий резерв.** Сервери організовані в кластер і дані реплікуються між ними. Обидві копії працюють безперервно з ідеальною синхронізацією, тому альтернативна система може негайно взяти на себе відповідальність, якщо основна система вийде з ладу. Для кластера, RTO це час перемикання на інший сервер. Наприклад, VMware HA, переключається за 2 хвилини (з урахуванням старту віртуальної машини, її гостьовий ОС і додатків). Значить, таке рішення підходить для баз даних з цільовим значенням RTO від 2 хвилин.
- **Теплий резерв.** Копія змін в БД (наприклад за день) надсилається на інший сервері, наприклад, в хмарі. Дані повинні бути відновлені з резервної копії, перш ніж система може бути використана. RPO залежить від частоти резервного копіювання, RTO залежить від розміру системи та технології резервного копіювання.
- **Холодний резерв** мало чим відрізняється від стратегії резервного копіювання та відновлення, за винятком того, що відновлення відбувається на інший сервер в іншому місці. Резервний сервер запускається, коли це необхідно. RPO такий самий, як і для «теплого резерву», RTO тепер також залежить від часу, необхідного для організації нового серверу.

Контрольні запитання

1. Що таке транзакція? Які її властивості? Чому транзакції є важливими одиницями роботи в СКБД?
2. Які існують основні операції в транзакціях?
3. Намалюйте діаграму станів транзакції та обговоріть кожен стан, який проходить транзакція під час її виконання.
4. Перерахуйте властивості ACID.
5. Що ви розумієте під атомарністю? Чому вона важлива? Поясніть на прикладі.
6. Дайте визначення проблеми брудного читання.
7. Які існують типи блокування?
8. Що таке протокол двофазного блокування?
9. Що таке взаємне блокування (deadlock)? Як можна уникнути тупиків?
10. Перелічіть різні схеми запобігання тупикам.
11. Що таке Checkpoint і як він використовується для відновлення?
12. Що таке серіалізованість?
13. Що означає паралелізмом?
14. Які існують способи резервного копіювання даних?
15. Що таке граф очікування?

Тестові завдання

1. Що з наведеного є властивістю транзакції?

   A. Atomicity
   B. Consistency
   C. Durability
   D. Все перераховане вище

2. Що в базі даних забезпечує транзакція на її початок і по її закінченню?

   A. Consistency
   B. Redundancy
   C. Latency
   D. Anonymity

3. _______ вказує, що всі операції транзакції повинні відбуватися цілком, інакше транзакцію буде перервано.

   A. Atomicity
   B. Consistency
   C. Isolation
   D. Durability

4. Що є ІСТИНА щодо стану транзакції Committed?

   A. Транзакція виконує свою останню операцію
   B. Дані все ще не зберігаються в базі даних
   C. Всі зміни тимчасово зберігаються в базі даних
   D. Нічого з перерахованого вище

5. Стан транзакції ___ означає, що всі зміни було збережено у базі даних постійно

   A. Committed
   B. Partially Committed
   C. Active
   D. Aborted

6. Система відновлення бази даних гарантує, що база даних повернеться до _______, якщо транзакція буде невдалою.

   A. Попередній узгоджений стан
   B. Пост-послідовний стан
   C. Неузгоджений стан
   D. Жодне з перерахованих вище

7. ___ розклад – це тип розкладу, в якому одна транзакція виконується повністю перед початком іншої транзакції.

   A. Serial
   B. Non-serial
   C. Serializable
   D. Non-serializable

8. Яка умова(и) має бути виконана для того, щоб дві транзакції конфліктували?

A. Обидві транзакції є незалежними.
B. Транзакції мають доступ до одного елементу даних.
C. Операція запису присутня хоча б в одній з них.
D. Усе перераховане вище

9. У випадку тупика (deadlock) або відсутності ресурсів, активна транзакція _____?

A. Committed
B. Rollbacked
C. Aborted
D. Sent

10. Під час одночасного виконання кожна операція повинна виконуватися в певному порядку, щоб не створювати перешкод іншим операціям. Таким чином в базі даних підтримується ___.

A. Узгодженість
B. Надмірність
C. Паралельність
D. Нічого з перерахованого

11. Потрібно забезпечити виконання серії операторів SQL як єдиного атомарного блоку, і якщо будь-який оператор у цій групі зазнає невдачі, то вся група операторів повертається до початкового стану. Яку команду мови управління транзакціями (TCL) слід використати для виконання цього завдання?

A. BEGIN TRANSACTION
B. COMMIT
C. ROLLBACK
D. SAVEPOINT

12. Що таке deadlock?

A. Це ситуація в багатопотоковому або багатопроцесному середовищі, коли два або більше процесів не можуть продовжувати роботу, оскільки кожен з них чекає, поки інший звільнить ресурс
B. Це помилка, яка виникає через синтаксичну помилку в операторі SQL
C. Це операція з базою даних, яка об'єднує декілька запитів в один набір результатів
D. Це механізм безпеки, який використовується для захисту конфіденційних даних у базі даних

13. ________ означає, що транзакція повинна виконуватися рівно один раз повністю або не виконуватися взагалі.

A. Довговічність
B. Узгодженість
C. Атомарність
D. Ізольованість

14. Припустимо, що транзакція A має спільне блокування таблиці R. Якщо транзакція B також запитує спільне блокування на R

A. Це призведе до тупикової ситуації

B. Вона буде негайно відхилена
C. Буде негайно задоволено
D. Його буде надано, як тільки він буде відпущений A

15. ________ означає, що після завершення транзакція повинна залишити базу даних в узгодженому стані.

A. Data isolation
B. Data duration
C. Data consistency
D. Data non-reputability

9. BIG DATA І БАЗИ ДАНИХ NoSQL

Традиційні застосунки реляційних баз даних мають справу з даними, обмежених рамками одного підприємства і мають чітко визначену структуру. Але сучасні програми для управління даними часто мають справу з даними, які не обов'язково мають реляційну форму; крім того, такі програми також мають справу з обсягами даних, які набагато більші, ніж ті, що генеруються одним підприємством. У цьому розділі розглядаються методи управління такими даними, які часто називають великими даними.

9.1. Великі дані (Big Data)

Традиційні джерела даних, з якими мають справи реляційні СКБД характеризуються досить високою щільністю корисної інформації і можуть бути представлені в структурованому вигляді. Але в останні роки стає актуальним проблема обробки даних нового покоління які відрізняються великою різноманітністю, об'ємом і швидкістю надходження. У той же час частка інформації, на основі якої приймаються будь-які рішення, в даних нового покоління набагато нижче і становить зазвичай не більше 10%. Прикладом даних нового покоління можуть бути (рис. 9.1):

- дані, що надходять безперервним потоком з різних пристроїв (Internet of Things);
- дані з GPS пристроїв, які вимагають високу швидкість обробки;
- повідомлення і різного роду контент з соціальних мереж.

За даними Forbes у 2018 році[6] за нинішніх темпів щодня створюється 2,5 квінтильйона байт даних, але ці темпи лише прискорюється зі зростанням Інтернету речей (IoT). За останні два роки було створено 90 відсотків даних у світі.

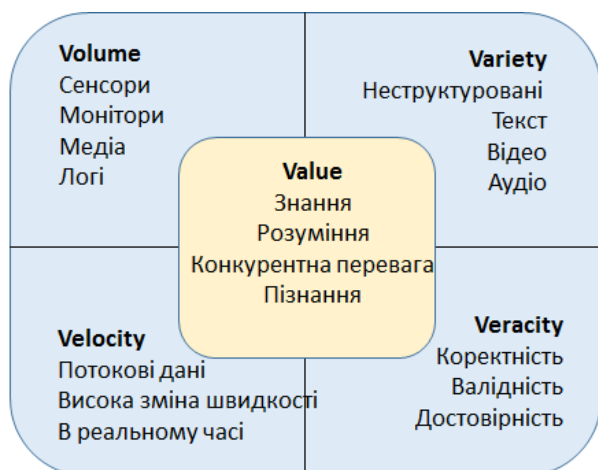
Рис. 9.1. Сучасні джерела потоків даних

Через свій масштаб, різноманітність, розосередженість та потреб своєчасній обробки такі дані потребують застосування нових технічних засобів та аналітики. Для

[6] https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyoneshould-read/

обробки таких даних були розроблені різні технології, що мають загальну назву Big Data – великі дані. Термін Big Data ширший за просте поняття «великі дані». Big Data в інформаційних технологіях, це серія підходів, інструментів і методів обробки структурованих і неструктурованих даних величезних обсягів і значного різноманіття для отримання результатів, ефективних в умовах безперервного приросту, розподілу по численних вузлів обчислювальної мережі, альтернативних традиційним системам управління базами даних.

Існують різні визначення великих даних, але більшість з них базується на концепції «5V» – до Big Data відноситься дані і технології, що відповідають наступним вимогам (рис. 9.2):



- Рис. 9.2. Характеристики великих даних

- **Volume** (Об `єм) Дані обсягом > 1 ТБ (аж до пета-, екса- байтів). Наприклад, мільярди пристроїв Internet of Things генерують гігантські обсяги даних.
- **Variety** (Різноманітність). Соціальні медіа (Facebook, Twitter, Instagram ..) створюють інформацію вид текстових коментарів до відео або зображень. Дані як структуровані так і неструктуровані.
- **Velocity** (Швидкість). Наприклад, мільйони мобільних пристроїв одночасно отримують доступ до Google Maps.
- **Veracity** (Правдивість). Довіра джерелам даних. Питання якості даних є особливо складними в контексті великих даних. Звичайно, в умовах відмови окремих пристроїв, ненадійності мережевих з’єднань, питання довіри джерелам даних стоїть надзвичайно гостро.
- **Value** (Цінність). Великі дані не мають сенсу, якщо вони не забезпечують цінності для досягнення певної значущої мети. Великі обсяги, висока швидкість надходження та широкий спектр типів даних разом не гарантують, що дані дійсно є цінними для підприємства.

Технології з управління Big Data ґрунтуються на певних принципах:

1. Горизонтальна масштабованість.

Оскільки даних може бути дуже багато, то будь-яка система, яка передбачає обробку великих даних, повинна мати можливість розширюватися.

2. Відмовостійкість.

Методи роботи з великими даними повинні враховувати можливість виходу з ладу окремих комп'ютерів (а їх може бути багато – до кількох тисяч) і здатність долати ці проблеми без будь-яких значущих наслідків.

3. Локальність даних.

У великих розподілених системах дані розосереджені по великій кількості вузлів. Принцип локальності даних полягає в тому, щоб за можливості обробляти дані на тому самому вузлі, де вони зберігаються.

Для того щоб дотримуватися цих принципів, необхідні технології (включають різні методи і способи) засобів обробки і використання на різних етапах життєвого циклу даних. Процес використання великих даних можна розглядати за допомогою метафори конвеєру, як показано на рис. 9.3.

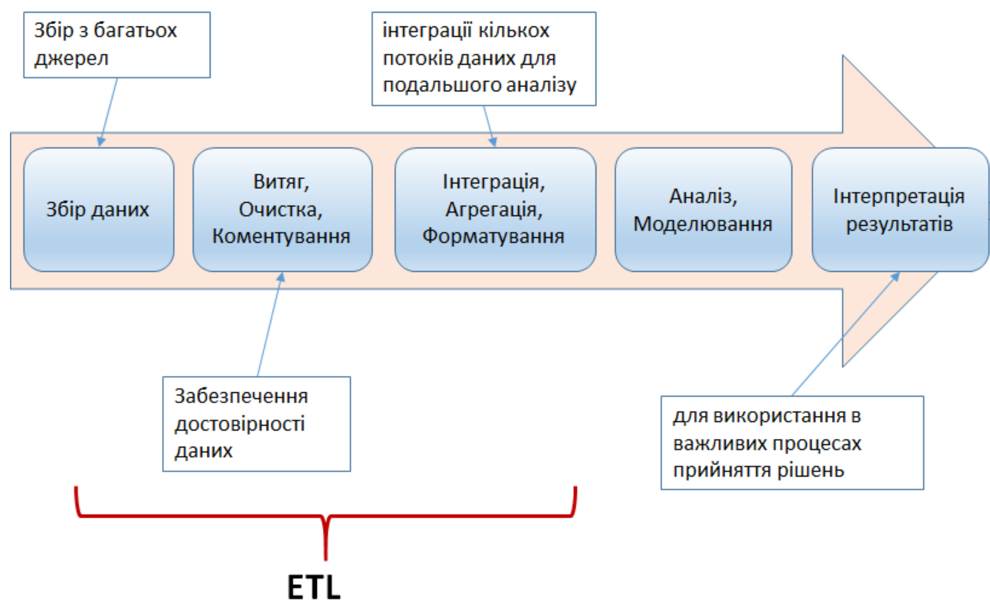


Рис. 9.3. Процес використання великих даних

Спочатку дані повинні бути зібрані з можливо багатьох різних джерел. Для забезпечення достовірності дані повинні бути зібрані, очищені та анотовані. Анотація може бути використана для створення метаданих про значення даних і походження, про історію джерел даних. Через великий обсяг даних часто потрібні методи для інтеграції декількох потоків даних або для агрегування даних у представлення. Ці етапи попередньої обробки мають загальну назву ETL (Extract, Transform, Load). Підготовлені дані краще підтримують наступний етап конвеєра, що передбачає застосування методів аналізу та моделювання. Останній етап конвеєра передбачає інтерпретацію результатів аналізу, можливо, для використання в критично важливих процесах прийняття рішень.

Масштабність і неоднорідність даних створюють виклики для конвеєра великих даних. Своєчасність і складність обробки великих обсягів неструктурованих даних також ускладнюють можливість їх аналізу та надання результатів, коли вони потрібні. Особливе занепокоєння викликає конфіденційність, особливо з огляду на те, що значна частина зібраних даних є персональними за своєю природою. Від транзакцій за кредитними

картками до постів у соціальних мережах – особиста ідентичність, місцезнаходження, медичні записи, звички та переконання приватних осіб потенційно можуть бути викриті та порушені.

Для реалізації всього конвеєру обробки розроблені різні технології. Розглянемо два принципово різних підходу до обробки великих даних. Перший підхід, Hadoop, орієнтований на пакетну обробку даних[7]. Інший, NoSQL, призначений для обробки даних в режимі реального часу.

9.2. Hadoop

Великі дані вимагають особливого підходу до розподіленого зберігання даних. Хоча можливі різні технології реалізації такого підходу, Hadoop став стандартом де-факто для більшості систем зберігання та обробки великих даних. Hadoop – це не база даних, це фреймворк на основі Java для розподілу та обробки дуже великих наборів даних на кластерах комп'ютерів. Хоча фреймворк Hadoop складається з багатьох частин, двома найважливішими компонентами є розподілена файлова система Hadoop (HDFS) та MapReduce. HDFS – це низькорівнева розподілена система обробки файлів, що означає, що її можна використовувати безпосередньо для зберігання даних. MapReduce – це модель програмування, яка підтримує обробку великих масивів даних у високопаралельний, розподілений спосіб. Хоча HDFS і MapReduce можна використовувати окремо, ці дві технології доповнюють одна одну, тому вони краще працюють разом як система [7].

9.2.1. Розподілена файлова система HDFS

Фреймворк для зберігання даних Hadoop підтримується розподіленою файловою системою HDFS. Перевага розподілених файлових систем (distributed file system – DFS) полягає в тому, що вони здатні представляти набори даних, які занадто великі, щоб поміститися в обсязі пам'яті одного комп'ютера, розподіляючи дані по мережі комп'ютерів. Недоліком DFS є складність управління всіма компонентами файлів і метаданими, необхідними для зберігання та отримання даних з розподілених місць. Однак розподілена файлова система, така як HDFS, також має ряд додаткових переваг. Зокрема, HDFS забезпечує відмовостійкість, оскільки здатна розділяти файл на блоки, а потім реплікувати копії кожного блоку. Якщо один блок недоступний, HDFS може отримати доступ до іншої копії цього ж блоку на іншому комп'ютері.

Підхід розподіленої файлової системи Hadoop (HDFS) до розподілу даних базується на кількох ключових припущеннях:

1. Великий обсяг.

Дані в HDFS організовані у фізичні блоки, розмір блоку за замовчуванням 64 MB, і його можна налаштувати на ще більші значення. В результаті, кількість блоків у файлі значно зменшується. Великі розміри блоків мінімізують час пошуку файлів на диску –

---

[7] Пакетна обробка - це метод, що використовується комп'ютерами для періодичного виконання великих обсягів повторюваних завдань обробки даних

чим менше блоків, тим менше запитів на диск, що призводить до швидкого отримання даних.

2. Запис – один раз, читання – багато.

Використання моделі «запис один раз, читання багато» спрощує проблеми паралелізму та покращує загальну пропускну здатність даних. За цією моделлю файл створюється, записується у файлову систему, а потім закривається. Після закриття файлу до його вмісту не можна вносити зміни. Це підвищує загальну продуктивність системи і добре підходить для типів завдань, що виконуються багатьма додатками для роботи з великими даними.

3. Потоковий доступ.

На відміну від систем обробки транзакцій, де запити часто отримують невеликі фрагменти даних з декількох різних таблиць, додатки для роботи з великими даними зазвичай обробляють цілі файли. Замість того, щоб оптимізувати файлову систему для випадкового доступу до окремих елементів даних, HDFS оптимізований для пакетної обробки цілих файлів як безперервного потоку даних.

4. Відмовостійкість.

HDFS призначена для реплікації даних між багатьма різними пристроями, щоб у випадку збою на одному пристрої, дані були доступні з іншого пристрою. За замовчуванням Hadoop використовує коефіцієнт реплікації три, що означає, що кожен блок даних зберігається на трьох різних пристроях. За бажанням, для кожного файлу можна вказати різні коефіцієнти реплікації.

Розподіл блоків файлу між декількома комп’ютерами також підтримує паралельну обробку, що призводить до більш швидких обчислень. У Hadoop паралелізм досягається за допомогою парадигми програмування MapReduce. Як приклад обчислювальної потужності, на яку здатна HDFS, Hadoop зміг відсортувати 500 гігабайт даних за 59 секунд і 100 терабайт даних за 173 хвилини.

Рис. 9.4 ілюструє організацію HDFS і спосіб, у який файл розподіляється і реплікується. Кластер HDFS складається з вузлів і стійок. Вузол – це один комп’ютер. Стійка – це сукупність приблизно 30-40 вузлів на одному мережевому комутаторі. Кластер – це сукупність стійок.

Також на рис. 9.4 наведено приклад набору даних, який поділено на три блоки: B1, B2 і B3. При подачі даних на вхід Hadoop, HDFS розподілить і реплікує блоки відповідно до протоколу реплікації. За замовчуванням, кожен блок має три репліки, які розподілені на двох стійках. Перша репліка для B1 розміщується на вузлі 2 в стійці 1. Друга завжди розміщується на вузлі в іншій стійці. У цьому випадку копія B1 розміщується на вузлі 3 в стійці 2. Третя копія розміщується на випадковий вузол у тій самій стійці, що і друга, на рис. 9.4 це вузол 4 у стійці 2. Потім HDFS створює конвеєр для з’єднання реплік блоку. Репліки B2 розподілено між стійкою 2 і стійкою 3, а репліки B3 розподілені між стійкою 3 і стійкою 1. Користувач завжди має можливість збільшити кількість копій блоку.

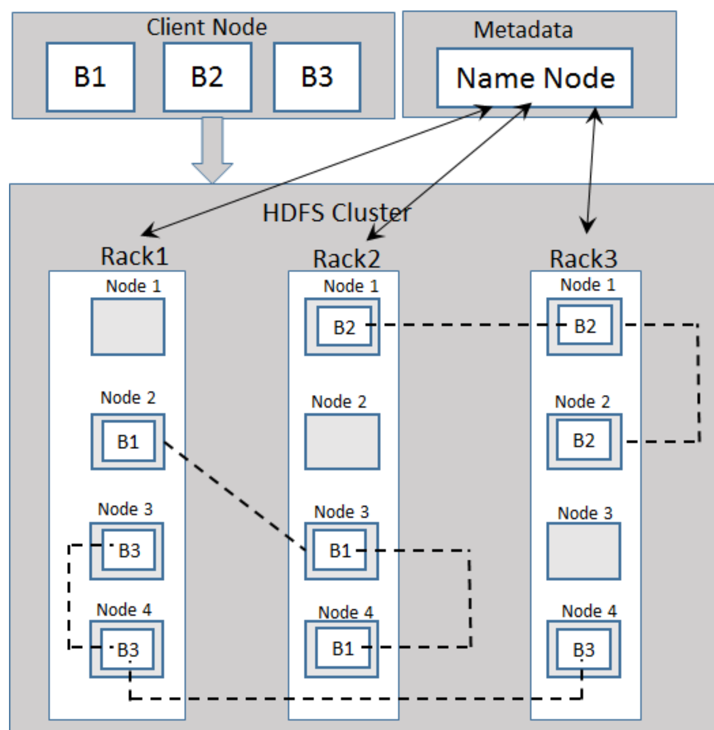


Рис. 9.4. Організація блоків наборів даних у кластері HDFS

Вузли, які використовуються для зберігання блоків, називаються вузлами даних. У HDFS також є вузол імен, який керує простором імен файлової системи. Вузол імен є основним компонентом метаданих у HDFS, який використовується для пошуку та отримання блоків і реплік, пов'язаних з кожним набором даних.

9.2.2. MapReduce

Набори даних, що використовуються в додатках для роботи з великими даними, є надзвичайно великими. Передача цілих файлів з декількох вузлів на центральний вузол для обробки вимагала б величезної пропускної здатності мережі і поклала б неймовірне навантаження на центральний вузол. Тому замість того, щоб обчислювальна програма отримувала дані для обробки в центральному місці, копії програми «розсилаються» на вузли, що містять дані для обробки. Кожна копія програми видає результати, які потім агрегуються по вузлах і надсилаються назад клієнту. Це дзеркальне відображення розподілу даних у HDFS.

MapReduce – це обчислювальний фреймворк, який використовується для обробки великих масивів даних на кластерах. MapReduce бере складну задачу, розбиває її на набір менших підзадач, виконує всі підзадачі одночасно, а потім об'єднує результат кожної підзадачі для отримання кінцевого результату для початкової задачі.

Як випливає з назви, це комбінація функції відображення (map) і функції зведення (reduce). Функція відображення приймає набір даних, сортує і фільтрує їх у набір пар ключ-значення. Функція відображення виконується програмою, яка називається маппером. Функція зведення приймає набір пар ключ-значення, всі з однаковим значенням ключа, і підсумовує їх в один результат. Функція зведення виконується програмою, яка називається редуктором.

Етап зведення використовується для виконання певних обчислень або агрегування даних, отриманих на етапі відображення. Може існувати декілька екземплярів однієї і тієї ж процедури відображення і процедури зведення, що виконуються паралельно над різними блоками великого набору даних, щоб досягти більш швидкого обчислення результатів по всьому набору даних.

Як правило, фреймворк Hadoop розподіляє маппер для кожного блоку на кожному вузлі даних, які потрібно обробити. Це може призвести до дуже великої кількості мапперів. Наприклад, якщо потрібно обробити 1 ТБ даних, а HDFS використовує блоки по 64 МБ, це означає, що потрібно понад 15 000 мапперів. Кількість редукторів налаштовується користувачем, але найкращі практики передбачають приблизно один редуктор на вузол даних.

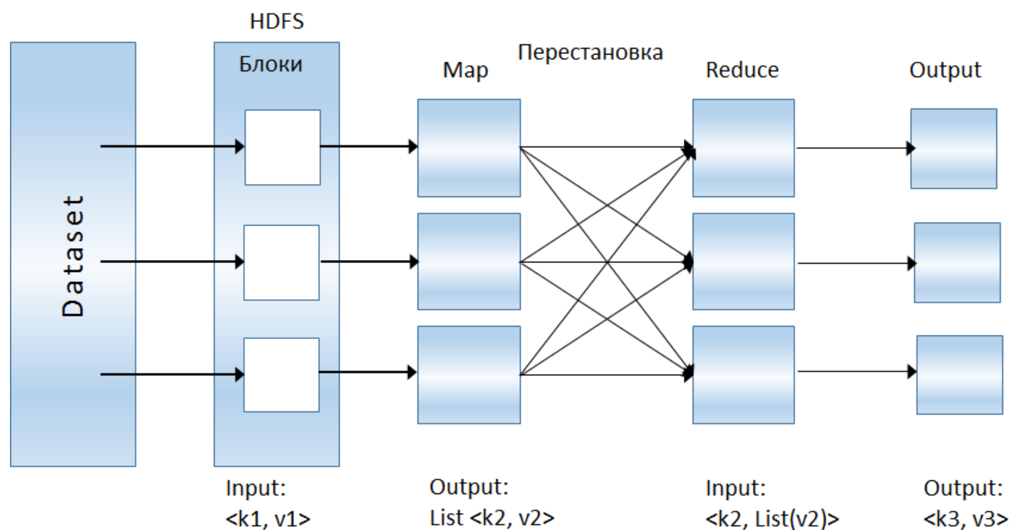


Рис. 9.5. Потік даних у MapReduce

На рис. 9.5 показано графічне представлення потоку даних у MapReduce. Коли набір даних потрапляє до HDFS, він розбивається на блоки. Дані також перетворюються у пари ключ–значення, $<k_1,v_1>$, для підготовки до введення у процедуру map. Ключ k_1 призначається HDFS. Значення v_1 зазвичай є окремим рядком вихідного файлу. Після виконання обчислень процедура map видає результат у вигляді list($<k_2, v_2>$), де кожне k_2 у списку є ключем, визначеним процедурою map, а кожне v_2 є відповідним значенням, визначеним логікою процедури map. На цьому етапі на виході кожного екземпляра процедури відображення можуть бути набір пар $<k_2,v_2>$, тобто list($<k_2, v_2>$).

Потім list($<k_2, v_2>$) проходить процес перестановки, який розподіляє кожне $<k_2,v_2>$ з вихідного списку кожного екземпляра процедури map до відповідної процедури зведення reduce, готуючи вхідні дані зведення до вигляду $<k_2,list(v_2)>$. У цьому перетворенні кожне $<k_2,v_2>$ з **однаковим значенням** k_2 об’єднується для створення списку значень, пов’язаних з тим самим ключем. Процедура зведення оперує з вхідними даними $<k_2,list(v_2)>$ для отримання вихідного списку ($<k_3,v_3>$), де $k_3 = k_2$, а v_3 – це, як правило, деяке обчислення або об’єднання list(v_2).

Приклад. Як приклад, припустимо, що сторінка власників домашніх тварин в Facebook проводить конкурс на визначення найпопулярніших імен собак, і що відвідувачі

сторінки можуть робити записи, які містять їхні логіни в Facebook та імена їхніх собак, причому для кожного собаки, яким вони володіють, є окремий запис. Файл для аналізу може виглядати приблизно як на рис. 9.6а. HDFS перетворює набір даних у наступний формат пари ключ–значення, а потім розподіляє рядки між трьома різними блоками (рис. 9.6б).

Припустимо, що процедура map перетворює кожен рядок у пару ключ–значення, що містить ім’я собаки як ключ і число 1 як значення, що вказує на єдине входження імені. Тоді вивід з кожного блоку буде таким як на рис. 9.6в.

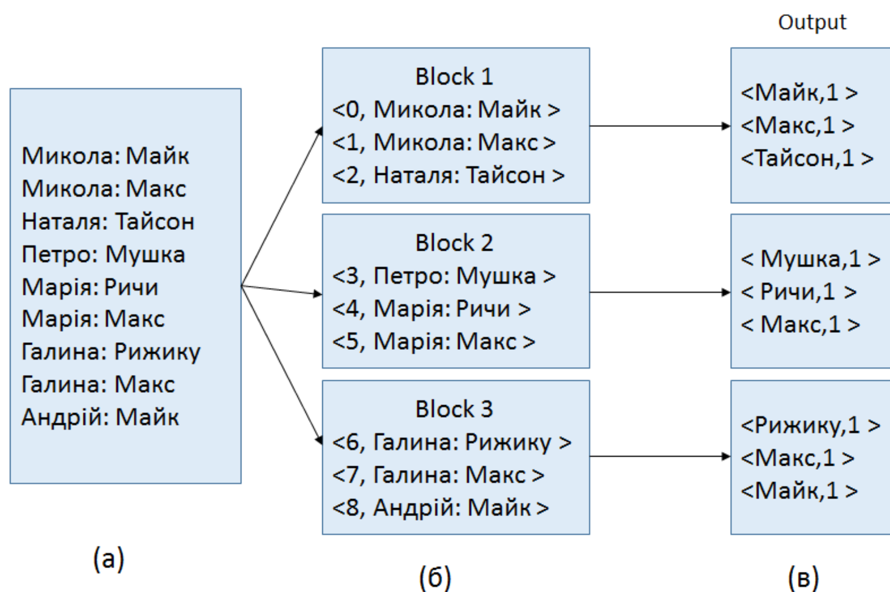


Рис. 9.6. Приклад процесу map-reduce

На рис. 9.7 показано, як вихідні дані map перемішуються для створення вхідних даних для процедури reduce, яка створює список значень, пов’язаних з кожним ключем. Функція reduce приймає декілька пар ключ-значення з **однаковими ключами** та списками. Дані у кожному блоці організовано у вигляді відсортованого списку ключів. У блоці 3 для процедури reduce вхідний запис для має вигляд <Ричи, list(1)>, оскільки був лише один запис для ключа Ричи. Запис для Майк в блоці 1 має вигляд < Майк, list(1,1)>, оскільки блок отримує два записи для ключа Майк. Значення list(1,1) створюється як об’єднаний список значень для кожного ключа Майк. Якщо процедуру reduce написати для підсумовування значень у списку, пов’язаних з кожним ключем, то на виході з процедури reduce кожне ім’я собаки буде ідентифіковано частотою входження кожного імені, причому Макс має найбільшу частоту.

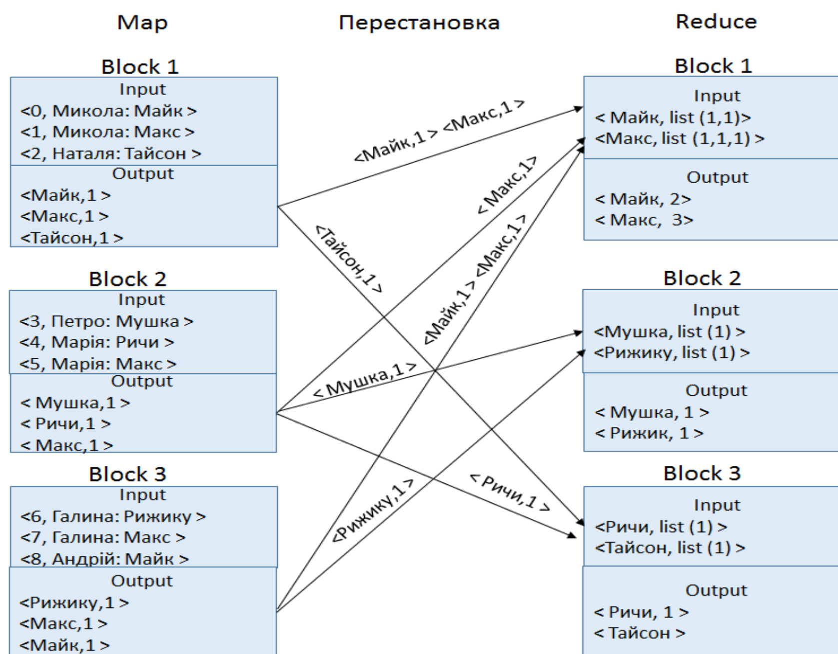


Рис. 9.7. Приклад MapReduce: Пошук найпоширеніших імен собак

Підсумовуючи, можна сказати:

- Завдання процесу map зчитувати дані і зводить їх до відповідних пар ключ-значення. Map читає лише один запис за раз, тому його можна розпаралелити і запустити на вузлі, який зберігає цей запис.
- Завдання процесу reduce обробити багато значень для одного ключа, отриманих від завдань map, і підсумовують їх в один результат. Кожен процес reduce працює з результатом одного ключа, тому його можна розпаралелити за ключами. Процеси reduce, які мають однакову форму для входу і виходу, можна об'єднувати в конвеєри. Це покращує паралелізм і зменшує обсяг даних, що передаються.
- Операції map-reduce можна об'єднати у конвеєр, де вихід однієї операції є входом для map іншої операції.
- Якщо результат обчислень map-reduce широко використовується, його можна зберігати у вигляді матеріалізованого подання. Матеріалізовані подання можна оновлювати за допомогою інкрементних операцій map-reduce, які обчислюють лише зміни у поданні, а не обчислюють все з нуля.

9.3. Засоби масштабування сховищ

Система Hadoop орієнтована на обробку великих даних в пакетному режимі. Однак існує багато завдань, коли обробку великих масивів інформації треба здійснювати в режимі реального часу, в тому числі і додавання, оновлення і вилучення даних, які к тому ж не мають заздалегідь визначеної фіксованої схеми. Бази даних NoSQL розроблені спеціально для вирішення таких завдань.

Але розробка таких баз даних вимагає рішення проблем забезпечення масштабованості, доступності, відмовостійкості і узгодженості даних .

Масштабованість – це фундаментальний аспект сучасного проєктування баз даних, який гарантує, що система зможе ефективно керувати зростаючими робочими навантаженнями. Масштабованість в системах баз даних зазвичай зводиться до двох основних варіантів: горизонтальне масштабування і вертикальне масштабування.

Вертикальне масштабування – це процес додавання додаткових ресурсів до існуючого сервера, таких як процесор, оперативна пам'ять і сховище. Хоча це простий метод, він має обмеження, в багатьох випадках вимагає великих поетапних інвестицій в апаратне забезпечення Горизонтальне масштабування, з іншого боку, передбачає додавання нових серверів для розподілу навантаження. Цей метод є більш адаптивним і, теоретично, нескінченно масштабованим, але він вносить ускладнення щодо розподілу та узгодженості даних. Тому додається завдання балансування навантаження. Балансування навантаження – це процес розподілу вхідних запитів до бази даних між декількома серверами так, щоб жоден сервер не був перевантажений.

У 2006 році Oracle оприлюднила версію 10g своєї бази даних. Буква «g» означає «grid». Грід – це просто пул комп'ютерів (дешевих, готових до використання), які можуть поділити між собою обробку даних. Масштабованість досягається за рахунок того, що додаткові комп'ютери можуть бути додані до (або видалені з) сітки. Звичайно, для того, щоб це працювало, повинен існувати певний механізм управління грід, який керує потоком завдань і даних.

У грід-рішенні Oracle, незважаючи на можливість додавання додаткових вузлів до грід-мережі, вони фактично надають лише додаткові місця для обробки даних. Дані не належать жодному окремому вузлу. Замість цього вони отримують доступ до єдиної спільної бази даних. З точки зору СКБД вона розглядається як єдина база даних, до якої вузли звертаються, коли їм потрібні дані. Всі вузли мають спільний доступ до бази даних, звідси і одна з назв цього підходу – спільний диск (shared disk) (рис.9.8).

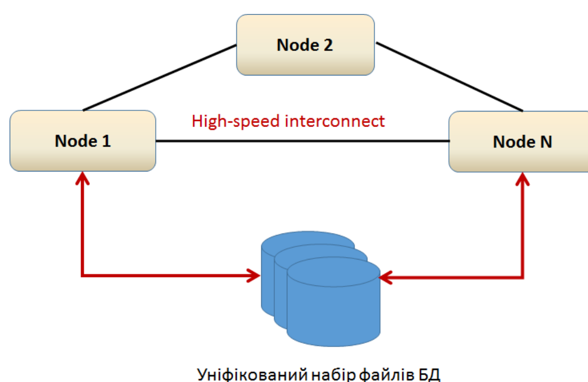


Рис. 9.8 .Технологія Shared Disk (Shared everything)

Хоча цей підхід має певні переваги у продуктивності, той факт, що кожен вузол має доступ до одних і тих самих даних, означає, що буде набагато більше блокувань, і ці блокування потрібно синхронізувати між усіма вузлами. Можна стверджувати, що це робить такий підхід не дуже масштабованим.

Підхід «нічого спільного» (shared-nothing) (рис. 9.9) означає, що кожен вузол контролює свої власні дані і має власну пам'ять і дисковий ресурс.

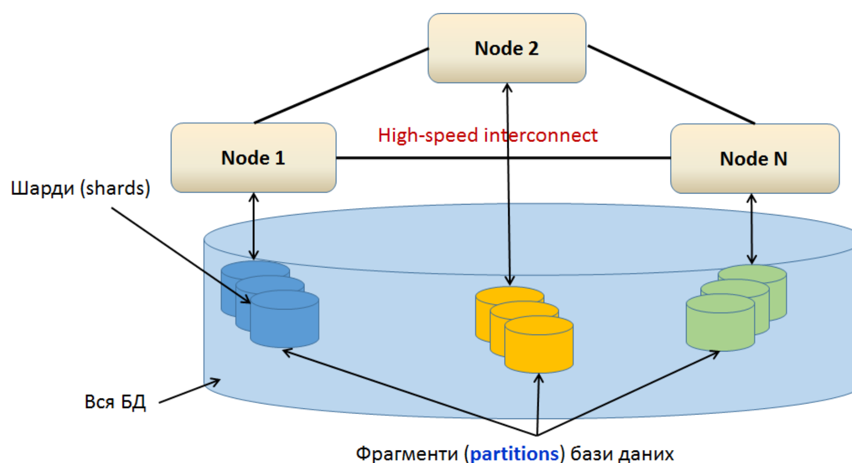


Рис. 9.9. Технологія Shared Nothing (шардінг)

Тут вся база даних фактично є логічною конструкцією, що складається з окремих баз даних на кожному вузлі. На кожному вузлі працює власний екземпляр бази даних.

Шардінг можна визначити як поділ бази даних на декілька менших баз даних, які разом складають одну логічну базу даних. Розподілені запити дозволяють швидше обробляти їх завдяки реалізації паралельного виконання на кожному шарді.

Шардінг набув популярності завдяки значному зростанню розміру баз даних у таких сервісах, як програмне забезпечення як послугу (SaaS), та вебсайти соціальних мереж. Ця модель може бути легко реалізована в хмарному середовищі, в якому сервери (реальні або віртуальні) розташовані в дата-центрі постачальника послуг, доступному через загальнодоступний Інтернет. Багато NoSQL-систем забезпечують автоматизований шардінг, тобто розподіл даних між декількома вузлами таким чином, щоб кожен сервер міг працювати незалежно з даними, що знаходяться на ньому. Це дає можливість реалізувати архітектуру «нічого спільного».

Шарди можуть знаходитися на одному сервері або на декількох серверах. Мета полягає в тому, щоб розділити набір даних таким чином, щоб кожен шард був більш керованим і до нього можна було отримати незалежний доступ або оновити. Ця стратегія особливо ефективна для підвищення швидкості читання і запису великих розподілених баз даних.

Реалізувати розподіл даних за шардами можна на рівні застосунків або на рівні проміжного програмного забезпечення (Distrbuted Database Midlware – DDM), що входить до складу СКБД, як показано на рис. 9.10.

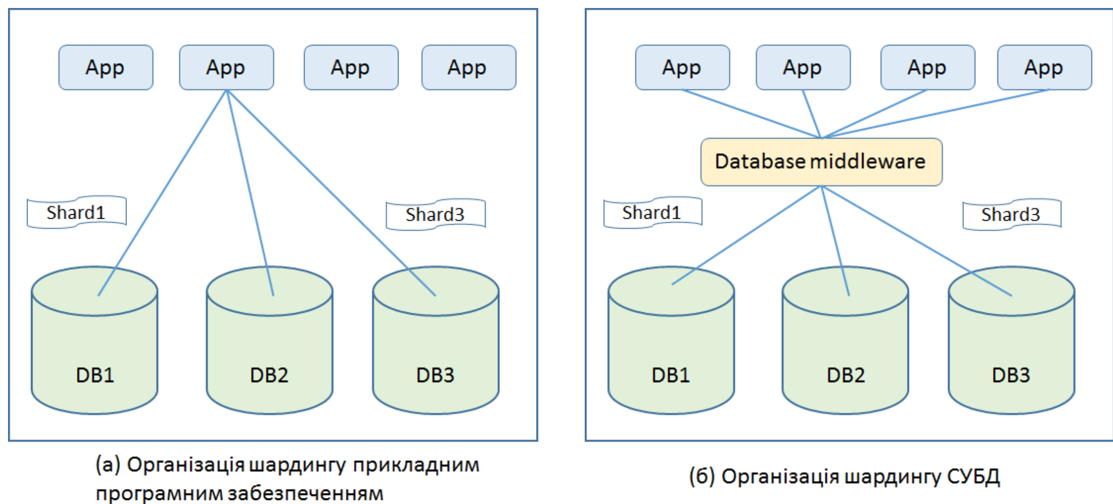


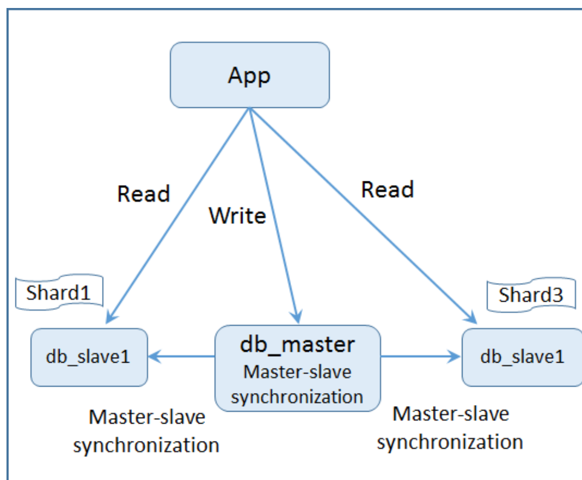
Рис. 9.10. Способи організації шардингу

Перевагою рішень для шардінгу на рівні застосунків (таких як Sharding-JDBC від Dangdang, розподілений фреймворк даних Taobao тощо) є те, що вони безпосередньо підключаються до бази даних і мають менше додаткових накладних витрат. Його недоліками є складність в організації великої кількості з'єднань, необхідність в модернізації та оновленні і високої вартості експлуатації та обслуговування.

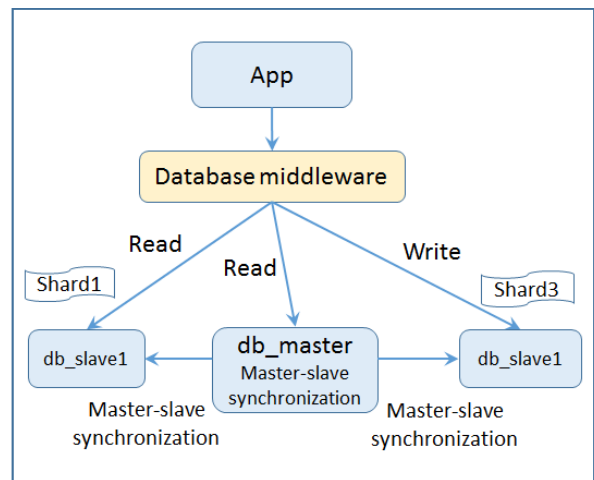
Коли розподіл виконується у коді програми, програма повинна відстежувати, які ключі зберігаються у якій базі даних, і спрямовувати запити до відповідної бази даних. Запити, які читають або оновлюють дані з декількох баз даних, не можуть бути оброблені простим способом, оскільки неможливо подати один запит, який буде виконано в усіх базах даних. Замість цього програмі потрібно буде прочитати дані з декількох баз даних і обчислити остаточний результат запиту. Оновлення між базами даних викликають додаткові проблеми.

Перевагами рішень для організації шардінгу на рівні проміжного програмного забезпечення (наприклад, Mycat, Cobar, Ekoson) є відсутність змін у застосунку, незалежність від мови, повна прозорість для застосунку при масштабуванні бази даних та ефективне управління кількістю з'єднань завдяки спільному використанню з'єднань (рис. 9.11). Недоліком є можливість додаткових затримок (<4%).

Хоча шардінг, що виконується шляхом модифікації коду програми, забезпечує простий спосіб масштабування додатків, незабаром стали очевидними обмеження цього підходу. Процес розширення може призвести до простою програми та переривання роботи сервісу. На сьогоднішній день шардінг даних реалізується засобами СКБД без необхідності змін в додатку.



(а) Організація читання/запису прикладним програмним забезпеченням



(б) Організація читання/запису СУБД

Рис. 9.11. Читання / запис даних з шардів

Горизонтальне масштабування бази даних за допомогою DDM дозволяє автоматично балансувати дані, досягати необмеженого розширення (необмежена кількість підтримуваних шардів, легкість роботи з великими обсягами даних), повної автоматизації (розширення в один клік, автоматичний відкат аномалій) та незначного впливу на прикладні сервіси.

Розподіл даних за шардами

Розподіл даних за шардами можна за такими принципами:

- шардінг даних по їх **географічному** розташуванню (випадок, коли заздалегідь вибирається ключ фрагментації)
- автоматичне фрагментація по синтетичному ідентифікатору (ключу розбиття), коли стоїть завдання **рівномірно розподілити** дані за фрагментами.

Наприклад, застосунок шукає інформацію про клієнта на основі його прізвища (рис. 9.11). Можливо, доцільно мати два шарди, один з яких містить всі прізвища, що починаються з А до Л, а інший – з М до Я. І якщо єдиний запит до даних стосується однієї особи, то таке розділення зменшить обсяг пошуку, який потрібно здійснити. Але якщо потрібно знайти всіх клієнтів, які живуть у певному місті, цей запит може призвести до необхідності виконання з’єднань, а управління цим процесом може зайняти більше часу, ніж якби всі дані були в одній таблиці. Зрозуміло, що вибір ключа шардингу є ключем до успіху або неуспіху цього підходу. В ідеальному випадку, дані повинні бути згруповані таким чином, щоб один користувач здебільшого отримував свої дані з одного сервера.

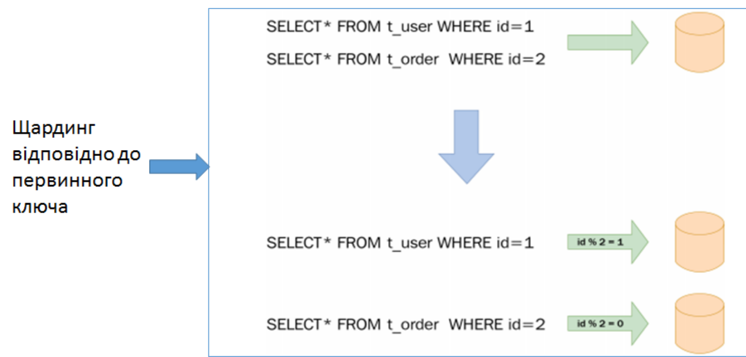


Рис. 9.12. Горизонтальний шардинг

Як альтернатива поділу таблиці на менші таблиці по горизонталі, вертикальне розбиття ділить таблицю на менші таблиці по вертикалі (рис. 9.13). У цьому випадку замість рядків кожен розділ містить підмножину стовпців..

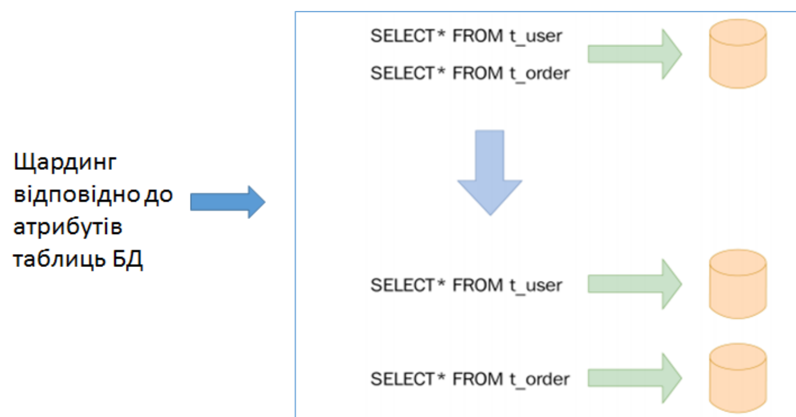


Рис. 9.13. Вертикальний шардинг

Вертикальне розбиття часто використовується, щоб відокремити стовпці, до яких часто звертаються, від тих, до яких звертаються рідше. Це дозволить оптимізувати дисковий ввід/вивід, оскільки більшість запитів будуть сканувати лише ті стовпці, до яких часто звертаються

9.4. Засоби забезпечення доступності і відмовостійкості

Основним засобом підвищення доступності і відмовостійкості сховищ є реплікація. Реплікація баз даних – це постійне копіювання даних між вузлами розподіленого кластера баз даних. Реплікація розподіленої бази даних передбачає копіювання даних між кількома вузлами, щоб вихід з ладу одного вузла не призвів до недоступності системи та/або втрати даних.

Основні види реплікацій – головний-підлеглий і однорангова реплікація.

Реплікація головний-підлеглий (Master-slave)

В такому типі реплікації створюється головний екземпляр бази даних (майстер-копія) і зберігається як ключове джерело даних. Будь-які оновлення, які можуть знадобитися, вносяться до цієї головної копії, а потім передаються на підлеглі копії. Для підтримки високої продуктивності, всі запити на читання обробляються

підпорядкованими копіями (рис. 9.14). У випадку виходу з ладу головної копії, одна з підлеглих копій автоматично призначається новою головною копією.

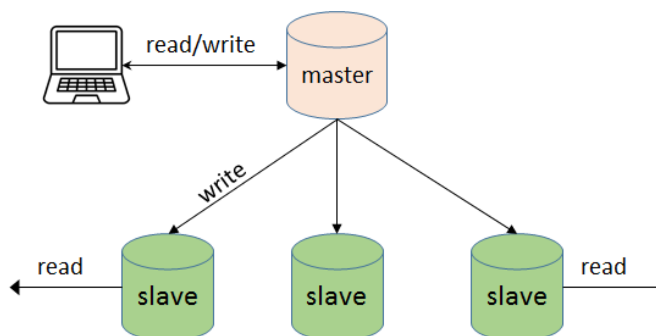


Рис. 9.14. Реплікація головний-підлеглий

Такий спосіб реплікації добре підходить для наборів даних з великою кількістю запитів на читання. За необхідністю кількість підлеглих вузлів збільшується. Якщо головний вузол вийде з ладу, підлеглі все одно зможуть обробляти запити на читання.

Але головний вузол (майстер) є потенційною точкою відмови. Його вихід з ладу позбавляє можливості обробляти запити на запис до тих пір, поки майстер не буде відновлено, або не буде призначено нового майстра. Це означає, що у випадку збою головної копії певні зафіксовані транзакції будуть втрачені, і жодна підпорядкована копія не буде містити цю інформацію.

Технологія Master-slave не підтримує високе масштабування запитів на запис. Якщо необхідно масштабувати такі запити, знадобиться додаткова обчислювальна потужність на головному вузлі.

Однорангова реплікація

У таких базах даних дані реплікуються між декількома вузлами, всі з яких є рівноправними і жоден вузол не може вивести з ладу весь кластер. Типова топологія без головного передбачає три або більше реплік для кожного набору даних.

Однорангова реплікація даних може працювати тільки тоді, коли кожна копія містить ідентичний формат схеми і зберігає той самий тип даних. Крім того, відновлення бази даних є ключовою перевагою цієї техніки реплікації даних. Всі репліки мають однакову вагу, всі вони можуть приймати записи (рис. 9.15). Втрата будь-якої з них не перешкоджає доступу до сховища даних. Однорангова реплікація дозволяє записувати на будь-який вузол, вузли координують свої копії даних для синхронізації.

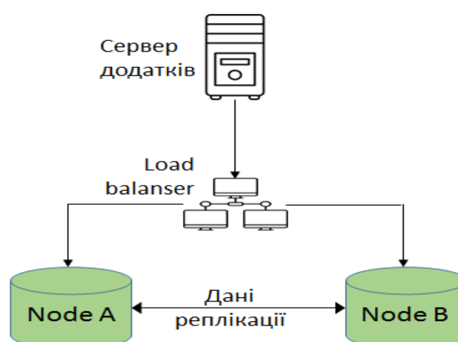


Рис. 9.15. Однорангова реплікація

Оскільки дані зберігаються на декількох вузлах, продуктивність однорангової реплікації даних залишається незмінною, навіть якщо навантаження зростає. Якщо вузол виходить з ладу, балансувальник навантаження СКБД може переадресовувати запити на читання з цього вузла на інші сусідні вузли і підтримувати середовище обробки без втрати доступність даних.

Метод однорангової реплікації полегшує обслуговування вузлів, оскільки дозволяє виводити окремі вузли в автономний режим для оновлення або технічного обслуговування, не знижуючи загальну продуктивність системи.

До недоліків однорангової реплікації можна віднести:

- Одночасна модифікація певного рядка в декількох вузлах бази даних може призвести до неузгодженості даних, яку складно виправити.
- Повільне поширення змін до різних вузлів. Крім того, якщо реплікації виконуються у реальному часі, потрібно динамічно виконувати складне завдання балансування навантаження між різними вузлами.

9.5. Бази даних NoSQL

9.5.1. Особливості технології NoSQL

В реляційних БД підтримка властивостей ACID вимагає значних витрат. І цими накладними витратами стає надзвичайно важко керувати, якщо дані розподілені на багатьох серверах по всьому світу. Зазвичай це проявляється в дуже низькій продуктивності. Реляційні СКБД найкраще проявляють себе при виконанні інтенсивних операцій читання/запису над малими або середніми базами даних, або при виконанні великих пакетних процесів, але з обмеженою кількістю одночасних транзакцій. Зі збільшенням обсягів даних або кількості паралельних транзакцій потужність може бути збільшена шляхом вертикального масштабування, тобто збільшенням ємності сховища та/або потужності процесора сервера бази даних. Однак, існують апаратні обмеження для вертикального масштабування.

Тому подальше збільшення потужності має бути реалізовано шляхом горизонтального масштабування, коли декілька серверів СКБД об'єднуються в кластер. Відповідні вузли в кластері можуть балансувати робочі навантаження між собою, а масштабування досягається додаванням вузлів до кластера, а не розширенням потужності окремих вузлів.

Системи NoSQL були розроблені для вирішення проблем щодо ефективного розподілу та доступу до даних, гнучкого та динамічного визначення схем. Важливо розуміти, що NoSQL не є заміною систем RDBMS. Навпаки, технологія NoSQL може використовуватися як доповнення до технології RDBMS для обробки великих обсягів даних і задоволення потреб користувачів в Інтернеті, які вийшли за межі можливостей традиційної технології баз даних, орієнтованої на транзакції.

Ключовими особливостями технології NoSQL є:

- Зберігати та обробляти петабайти і терабайти даних в режимі реального часу. На основі потреб сучасних інтернет-додатків, таких як Amazon, Facebook, Twitter і Google, NoSQL була розроблена для ефективної обробки великих обсягів даних, які включають читання і запис у реальному часі з низькою затримкою.
- Горизонтальне масштабування з реплікацією та розподілом на окремих серверах. У той час як технологія реляційних СКБД вимагає дорогого спеціального обладнання для розширення, системи NoSQL були розроблені для горизонтального масштабування, тобто вони мають можливість розподіляти і реплікувати дані на звичайному обладнанні. Термін «звичайне обладнання» означає недорогі комп'ютери, які широко доступні і можуть бути легко об'єднані для формування розподілених обчислювальних кластерів.
- Гнучка схема. Системи NoSQL здатні обробляти структуровані, напівструктуровані та неструктуровані дані. На відміну від схем реляційних баз даних, дані, зібрані з датчиків або з Інтернету, часто відрізняються за своєю природою і не завжди відповідають заздалегідь визначеній структурі. Системи NoSQL достатньо гнучкі, щоб визначати і змінювати схеми даних у міру їх збору.
- Гнучка модель одночасного доступу. Системи NoSQL не відповідають властивостям ACID систем RDBMS, жертвуючи узгодженістю даних на користь доступності та масштабованості. Як наслідок, репліковані копії даних можуть не синхронізуватися миттєво при оновленні даних.
- Простий інтерфейс звернення до даних. Доступ до даних у NoSQL-системах здійснюється переважно через використання API, вбудованих у процедурний код. Деякі системи починають також включати SQL-подібні мови.
- Паралельна обробка даних з багатьох вузлів. NoSQL-системи використовують технології HDFS і MapReduce від Hadoop для підтримки ефективної паралельної обробки даних.

9.5.2. Теорема CAP (Брюєра)

В зв'язку з розподілом даних на багато вузлів, виникають проблеми з підтримкою цілісності і узгодженості даних. Протоколи забезпечення узгодженості, такі як двофазний протокол фіксації транзакцій мають обмежену масштабованість. Тому, невідповідність властивостям ACID є однією з найбільш помітних відмінностей між NoSQL системами та реляційними системами. Складність організації розподіленої бази даних може відноситись до дотримання таких властивостей:

- Узгодженість (**C**onsistency): на кожному вузлі одні й ті ж дані в будь-який момент часу (так звана сувора узгодженість).
- Доступність (**A**vailability): система продовжує працювати, навіть якщо вузли в кластері виходять з ладу або модернізуються.
- Стійкість до поділу (**P**artition Tolerance): відноситься до здатності бази даних знаходити альтернативні маршрути через мережу, щоб отримати доступ до даних з різних вузлів у разі розриву зв'язку.

Обмеження розподілених баз даних можна описати за допомогою так званої теореми CAP (Consistency, Availability, Partition Tolerance): будь-яка розподілена база даних зі спільним доступом до даних може мати щонайбільше дві з трьох бажаних властивостей: узгодженість, доступність або толерантність (рис. 9.16).

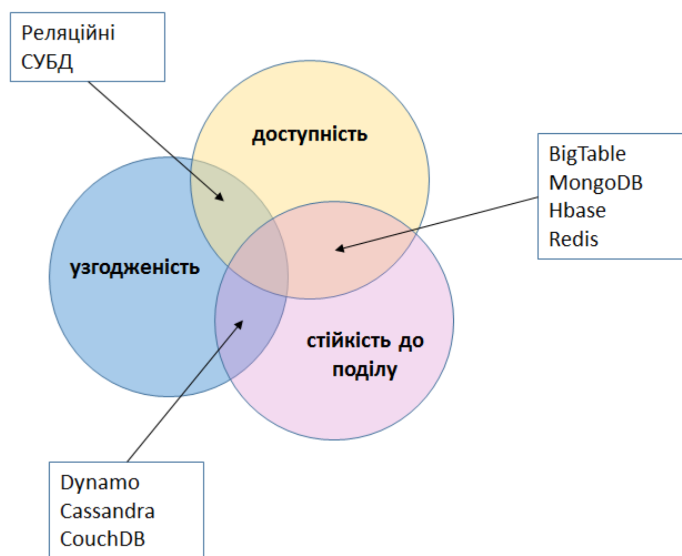


Рис. 9.16. Ілюстрація теорема CAP

Коли впроваджуються великомасштабні бази даних, в яких :

- ключовий фактором є доступність 24/7 і кілька хвилин простою означають втрачений дохід;
- при горизонтальному масштабуванні баз даних до тисяч машин, ймовірність відмови вузла або мережі надзвичайно зростає,

то, виходячи з з теореми CAP, щоб мати надійні гарантії доступності та толерантності розділів, доводиться жертвувати «суворою» узгодженістю. Тому більшість систем NoSQL надають перевагу доступності та толерантності до розділів, а не до узгодженості.

Підтримка узгодженості повинна балансувати між суворістю узгодженості та доступністю і масштабованістю.

Це призвело до появи баз даних з послабленими ACID-гарантіями, в яких задовольняються умови так званої кінцевої узгодженості [8]. Зокрема, в таких базах даних підтримуються властивості BASE (Basically Available, Soft-state, Eventually consistent):

- під базової доступністю (Basically Available) мається на увазі такий підхід до проєктування застосунків, щоб збій в деяких вузлах приводив до відмови в обслуговуванні тільки для незначної частини сесій при збереженні доступності в більшості випадків;
- нестійкий стан (Soft-state) має на увазі можливість жертвувати довгостроковим зберіганням стану сесій (таких як проміжні результати вибірок, інформація про навігацію, контексті), при цьому концентруючись на фіксації оновлень тільки критичних операцій;
- кінцева узгодженість (Eventually consistent), трактується як можливість суперечливості даних в деяких випадках, але при забезпеченні узгодження в

практично значимому часі. База даних називається кінцево узгодженою, якщо всі репліки поступово стають узгодженими за відсутності оновлень.

Щоб забезпечити узгодженість, використовуються такі прийоми:

- Виправлення читання: Коли відбувається операція читання і виявляється, що деякі дані на вузлі застарілі (тобто мають більш ранню мітку часу, ніж дані на інших вузлах), застарілі дані оновлюються.
- Відкладене виправлення: Програма поверне знайдене значення, навіть якщо воно може бути застарілим, але позначить його як таке, що потребує оновлення. Час оновлення залежить від керуючої або головної бази даних. Це називається слабкою узгодженістю. Це підхід, орієнтований на продуктивність.

Багато з доступних сьогодні баз даних NoSQL використовують філософію проєктування BASE. Багато з них також дозволяють керувати ступенем узгодженості, необхідним для будь-якої програми.

Існують різні варіанти побудови сховищ NoSQL (рис. 9.17). Всі вони мають різні підходи до зберігання даних (моделі даних), оскільки спрямовані на задоволення різних потреб , але їх об’єднує те, що традиційні правила і практики СКБД не завжди підтримуються, або підтримуються з деякими обмеженнями.

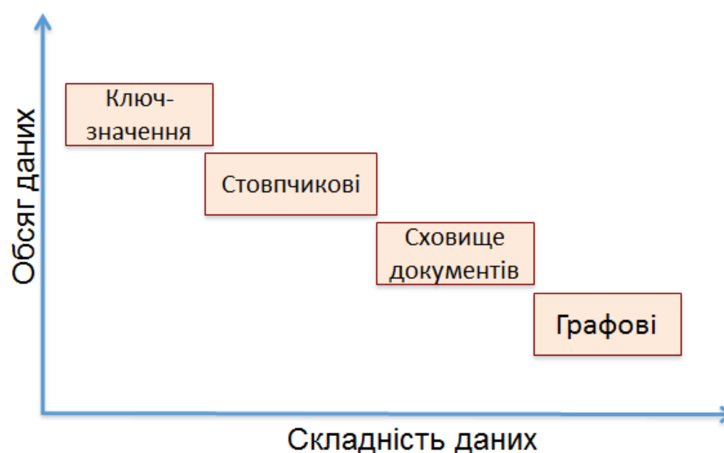


Рис. 9.17. Моделі даних NoSQL

9.5.3. Сховища «ключ-значення»

Сховища «ключ-значення» це найпростіший тип NoSQL-сховищ, що складаються з простих пар ключа і пов’язаного з ним набору (колекції) значень. База даних «ключ-значення» має структуру, яка дозволяє зберігати та отримувати доступ до «значень» використовуючи ключ. Ключі є унікальними і являють собою єдиний «пошуковий» критерій для отримання відповідного значення. Як правило, це рядок, з певним значенням і багато в чому він схожий на первинний ключ у реляційній таблиці. База даних не знає про вміст окремих колекцій «значень»; якщо потрібно змінити якусь частину «значення», потрібно буде оновити всю колекцію. Для таких бази даних «значення» – це довільна колекція байтів (рис. 9.18), і будь-яка обробка вмісту «значення» залишається за прикладною програмою. Єдині операції, які пропонує типове сховище ключ-значення – це

put (для зберігання «значення»), **get** (для отримання «значення» на основі «ключа») та **delete** (для видалення певної пари ключ-значення). Не існує операції оновлення - оновлення «значення», пов'язаного з певним «ключем», вимагає видалення пари і використання команди put для вставки нового запису. Оскільки сховища «ключ-значення» завжди використовують доступ по первинному ключу, вони, як правило, мають високу продуктивність і можуть бути легко масштабовані.

Деякі з популярних баз даних ключ-значення: Riak, Redis, Memcached DB, Amazon DynamoDB (код закритий).

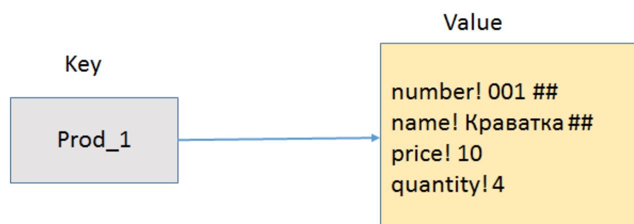


Рис. 9.18. Приклад пари Ключ - Значення

Як ключі, так і значення можуть являти собою що завгодно: від простих до складних об'єктів, наприклад, число, рядок, структуру JSON або масив. Сховища даних ключ-значення забезпечують безпрецедентне горизонтальне масштабування, недосяжне при використанні інших типів баз даних.

Сховища ключ-значення мають різний функціонал. У деяких, таких як Redis, значення, що зберігається, не обов'язково має бути об'єктом домену – це може бути будь-яка структура даних. Redis підтримує зберігання списків, множин, хешів і може виконувати операції з ранжирування, різниці, об'єднання та перетину. Ці можливості дозволяють використовувати Redis більш різноманітне, ніж стандартне сховище ключів і значень.

Сховище Riak дозволяє зберігати ключі у кошиках (bucket), які є лише способом сегментації ключів. Наприклад, в Riak в одному кошику з єдиним ключем можна зберігати дані про сесії користувача, налаштування користувача і дані щодо обраних товарів (рис. 9.19).

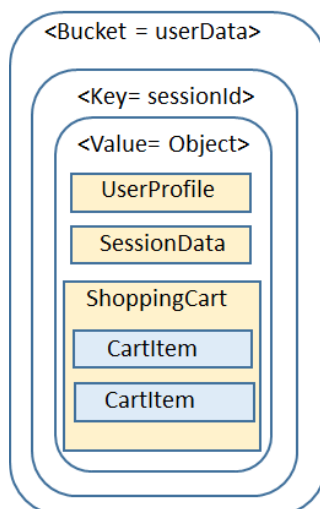


Рис. 9.19. Приклад кошику сховища Riak

Переваги моделі «ключ-значення»:

- сховища забезпечують швидкий доступ з незначними витратами;
- немає жорсткої схеми відносин між даними, тому в таких БД часто зберігають одночасно дані різних типів;
- розробник відповідає за визначення схеми іменування ключів і за те, щоб значення мало відповідний тип;
- дуже простий інтерфейс.

Недоліки бази даних ключ-значення

- ця модель не забезпечує жодних традиційних можливостей бази даних, таких як атомарність транзакцій або узгодженість при одночасному виконанні декількох транзакцій. Якщо в транзакції використовуються кілька ключів, і сталася помилка при збереженні одного з них, немає можливості повернути або відкотити решту операцій. Такі можливості повинні бути забезпечені прикладним програмним забезпеченням.
- зі збільшенням обсягу даних зберігати унікальні значення в якості ключів може стати складніше. Вирішення цієї проблеми вимагає введення певної складності в генерацію символьних рядків (наприклад, хешування), які залишатимуться унікальними серед надзвичайно великого набору ключових значень.
- якщо потрібно мати зв’язки між різними наборами даних або співвідносити дані між різними наборами ключів, такий тип сховища не є найкращим рішенням, навіть якщо деякі сховища ключ-значення надають функції переходу за посиланнями.
- запити за не ключовими атрибутами. Неможливо знайти ключі на основі чогось, знайденого в частині значення пари ключ-значення.
- операції над множинами. Оскільки операції обмежуються одним ключем за раз, не існує способу оперувати декількома ключами одночасно. Якщо потрібно оперувати з декількома ключами, доведеться робити це на стороні клієнта.

9.5.4. Сховища документів

Сховища документів не мають справу з «документами» у звичайному розумінні цього слова. Натомість документ у цьому контексті – це структурований набір даних, відформатований за допомогою такого стандарту, як JSON.

Різним значення ключа можуть відповідати набори даних різної структури, тобто такий тип сховища також без фіксованої схеми. Доступ до кожного «документу», як і раніше, здійснюється на основі «ключа». Головна відмінність між сховищами ключ-значення та сховищами документів полягає в тому, що сховище документів має можливість доступу та модифікації вмісту конкретного документа на основі його структури. Внутрішня структура «документа» (визначена в ньому за допомогою JSON) може бути використана для доступу та маніпулювання його вмістом.

Оскільки сховища документів повністю вільні від схем, відповідальність за схему перекладається на програму обробки. Недоліком відсутності фіксованої схеми є

відсутність посилальної цілісності та нормалізації. Однак відсутність обмежень на схему дозволяє гнучке зберігання широкого спектру даних, що і означає різноманітність в контексті великих даних. Це також полегшує фрагментацію та розподіл даних.

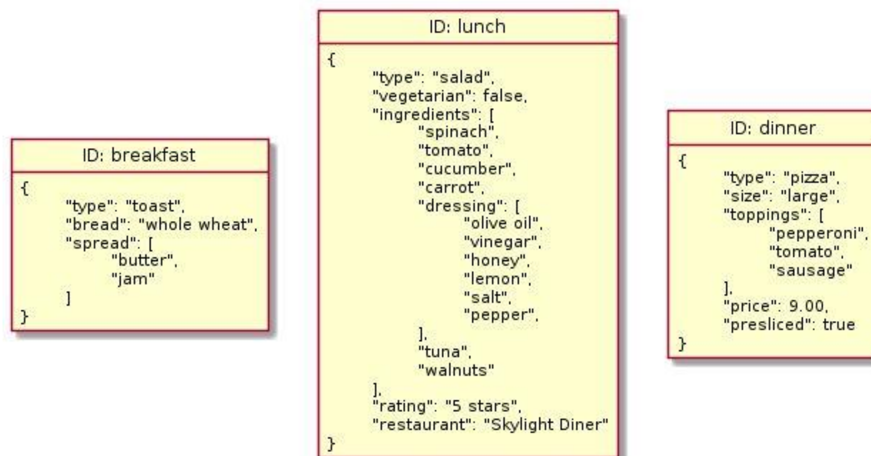


Рис. 9.20. Приклад документів в базі даних

У прикладі на рис. 9.20 ключ «lunch» використовується для доступу до документа, що складається з опису набору інгредієнтів. Кожним з них можна маніпулювати окремо в контексті сховища документів.

Сховища документів були розроблені спеціально для використання у вебсервісах. Тому їх можна легко інтегрувати з вебтехнологіями, такими як JavaScript і HTTP. Крім того, вони легко піддаються горизонтальному масштабуванню за допомогою шардінгу. Основна увага приділяється обробці великих обсягів різнорідних даних, тоді як для більшості даних, наприклад, із соціальних мереж, пошукових систем або новинних порталів, не потрібно забезпечувати сувору узгодженість даних.

Розглянемо особливості сховищ документів на прикладі найбільш популярного сховища такого типу MongoDB [8].

Модель даних MongoDB

Модель даних сховища наведено на рис. 9.21.

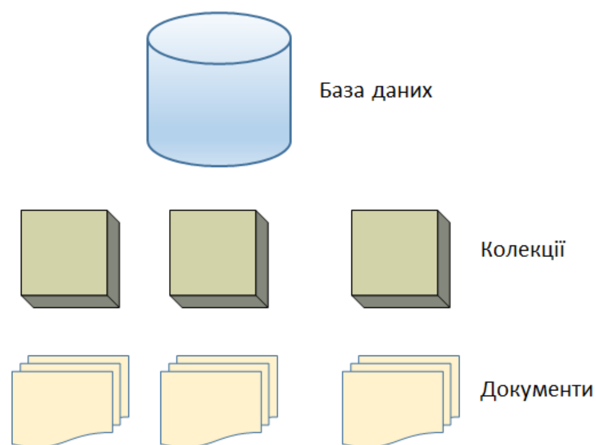


Рис. 9.21. Модель даних MongoDB

База даних – це фізичний контейнер для колекцій. Один сервер MongoDB зазвичай має декілька баз даних.

Колекція – група документів MongoDB. Це еквівалент таблиці в РСКБД. Колекція існує в межах однієї бази даних. Колекції не застосовують схему. Документи в колекції можуть мати різні поля. MongoDB додає до кожного документу поле _id типу ObjectID (12 байт), що грає роль первинного ключа. Все, що забезпечується – це те, що в межах однієї колекції поле «_id» має унікальне значення.

В інших СКБД, наприклад в DynamoDB від Amazon кінцевий користувач може вказати, який атрибут слід використовувати як унікальний ключ.

Документ – це набір пар ключ-значення. Документи мають динамічну схему. Динамічна схема означає, що документи в одній колекції не обов'язково повинні мати однаковий набір полів або структуру, а спільні поля в документах колекції можуть містити різні типи даних. Але, як правило, всі документи в колекції мають схоже або споріднене призначення.

Документ у контексті сховища документів – це файл зі структурованими даними, в вигляді списку пар «атрибут-значення». Усі значення атрибутів у цій структурі даних можуть рекурсивне містити власні списки пар «атрибут-значення». Вбудовування підлеглих документів як вкладених всередині документів, забезпечує легкий доступ і кращу продуктивність. Тобто документи можуть мати ієрархічну структуру, і вони не посилаються один на одного.

У документах немає порожніх атрибутів, якщо певний атрибут не знайдено, вважається, що він не був встановлений або не є релевантним для документа. В документі можна створювати нові атрибути без необхідності змінювати існуючі документи. Коли створюється новий документ, необхідно вказати, до якої бази даних і колекції належить цей документ (рис. 9.22).

Колекція

```
> db.menus.insert
  ({
    "type":  "toast",
    "brad":  ""whole wheat",
    "spread":  ["butter", "jam"],
  })
```

Документ в форматі JSON

Рис. 9.22. Додавання нових атрибутів в документ

Особливості організації взаємозв'язків в базі даних

Основним типом зв'язку між двома сутностями є зв'язок «один до багатьох». Розглянемо на прикладі, як зв'язок такого типу реалізується в MongoDB.

В базі даних зберігається інформація щодо науковців і нагороди, що вони отримали.. Основними сутностями, про які вам потрібно зібрати дані, є Науковці та Нагороди. Більшість користувачів цих даних цікавляться насамперед науковцями. Отже,

Scientists є центральною колекцією. Зразок документа в колекції Scientists показано на рис. 9.23.

```
{
"_id" : 1,
"name" : {
   "first" : "John",
   "last" : "Backus"
   },
"birth" : ISODate("1924-12-03T05:00:00Z"),
"death" : ISODate("2007-03-17T04:00:00Z"),
"contribs" : [
   "Fortran",
   "ALGOL",
   "Backus-Naur Form",
   "FP"
   ],
"awards" : [
   {
      "award" : "W.W. McDowell Award",
      "year" : 1967,
      "by" : "IEEE Computer Society"
   },
   {
      "award" : "National Medal of Science",
      "year" : 1975,
      "by" : "National Science Foundation"
   },
   {
      "award" : "Draper Prize",
      "year" : 1993,
      "by" : "National Academy of
Engineering"
   }
   ]
}
```

Ідентифікатор документа

Вкладені документи

Рис. 9.23. Реалізація зв’язку «один до багатьох»

Зв’язок між Науковці та Нагороди – це зв’язок «один до багатьох». У цьому випадку можна використовувати поняття вбудовування, щоб відобразити цей зв’язок «один до багатьох» шляхом створення масиву «нагород», як структуру вкладених документів. Основна перевага вбудовування полягає в тому, що пов’язані документи доступні в межах батьківського документа, що робить обробку запитів дуже ефективною.

Запити в MongoDB

Запити в MongoDB за своєю філософією схожі на запити до SQL, але мають синтаксис, який трохи складніший для розуміння. Кожен запит MongoDB виконується до певної колекції і складається з критерію відбору (аналог речення `WHERE`) і, за бажанням, порядку сортування (аналог `ORDER BY`) і специфікації підмножини полів (еквівалент проекції команди `SELECT` в SQL) які треба відобразити.

Приклад. Команда пошуку за умовою.

```
const cursor =
db.collection('Scientists').find({name.last:'Codd'});
```

Хоча метою створення бази даних NoSQL є масштабованість і швидкість обробки, виконання окремих запитів з багатьма критеріями запиту, операторами сортування або агрегації все одно може бути відносно повільним, навіть якщо не виконується з’єднання. Причиною цього є те, що кожен фільтр (наприклад, «`name.last = Baesens`») передбачає сканування кожного документа в колекції, щоб з’ясувати, які документи відповідають умовам запиту. Первинний ключ, визначений для кожної одиниці

зберігання, є винятком, оскільки цей ключ функціонує як унікальний індекс, що робить можливим ефективний пошук одиниць зберігання.

9.5.5. Сховища на основі стовпців

В реляційних базах даних дані зберігаються таким чином, щоб максимально ефективно зчитувати рядки, а не стовпці. Суть проблеми полягає в тому, що СКБД повинна прочитати кожен рядок, відкинувши непотрібну інформацію (або принаймні перемістивши позицію читання так, щоб пропустити непотрібні дані), щоб дістатися до даних у потрібному стовпчику. Необхідно пам’ятати, що пошук і читання з диска є одним з найповільніших процесів будь-якої системи баз даних.

Основна ідея стовпчикових сховищ – зберігання даних в стовпцях, а не в рядках. Елементи даних для кожного атрибуту зберігаються один за одним. Таким чином, процес читання дуже спрощується, і турбуватися потрібно лише про вказівники початку читання і кінця читання, і не потрібно переходити до різних сегментів диска, щоб перестрибнути через непотрібні дані. На рис. 9.24 наведені дані що зберігаються рядками.

1. Знайти перший рядок

2. Зміститись до потрібного стовпчику і зчитати значення

{A6U6-P5JF42JJ-8325XUAJ}	01.07.2010	14:48:57	PC-1689	Logoff	1	DTAA/AAA0371
{B3U0-S3EH07DL-9967NJSQ}	01.04.2010	8:08:17	PC-2282	Logon	0	DTAA/AAA0371
{H4H3-S7OL29QZ-3927MLAZ}	01.04.2010	16:05:15	PC-2282	Logoff	1	DTAA/AAC0344
{O7Q5-V1UP34UJ-9607OXME}	01.05.2010	8:03:26	PC-2282	Logon	0	DTAA/AAC0344
{W5S6-T1OP19BQ-1739SAHO}	01.04.2010	8:23:44	PC-0280	Logon	0	DTAA/AAH0734
{R9U5-Z3TI86QV-9158EEUQ}	01.06.2010	8:05:09	PC-2282	Logon	0	DTAA/AAC0344
{T9S4-F5RX22XN-8264ZNMV}	01.05.2010	8:09:42	PC-1800	Logon	0	DTAA/AAC0599
{J1Z9-M3EZ02WM-6809DHAO}	01.07.2010	8:37:46	PC-3233	Logon	0	DTAA/AAK0658
{T5H4-F0YD64XU-7868VISS}	01.07.2010	15:56:18	PC-2282	Logoff	1	DTAA/AAC0344
{C5N3-V2PC16JB-2018LZVJ}	01.04.2010	8:08:31	PC-1800	Logon	0	DTAA/AAC0599

3. Повторити для кожного рядку

Рис. 9.24. Зберігання даних рядками

На рис. 9.25 ті ж дані, але збереження стовпчиками. Підхід, орієнтований на стовпці, покращує ефективність читання, оскільки немає необхідності в ідентифікаторі рядка, а розміщення схожих даних разом дозволяє алгоритмам стиснення зменшити обсяг необхідного дискового сховища.

{A6U6-P5JF42JJ-8325XUAJ}, {B3U0-S3EH07DL-9967NJSQ}, {H4H3-S7OL29QZ-3927MLAZ} ......
01.07.2010, 01.04.2010, 01.04.2010,.....
DTAA/AAA0371, DTAA/AAC0344, DTAA/AAC0344

Читання значень послідовно

Рис. 9.25. Зберігання даних в стовпчиках

У такого способу зберігання існують і недоліки. Якщо є потреба створити запис (або рядок даних), його доведеться відтворювати, використовуючи позиційну інформацію для пошуку інших елементів даних зі стовпців. Очевидно, що це було б дуже неефективно. Друга проблема полягає в тому, що найкраща продуктивність з точки зору

стиснення буде досягнута, коли схожі значення знаходяться поруч один з одним. Якщо дані, що записуються, є мінливими, і відбувається багато вставок та оновлень, ця ефективність зрештою буде втрачена. Одна вставка може призвести до багатьох перезаписів, оскільки дані перемішуються, щоб забезпечити сусідство схожих значень. Тож стовпчикові бази даних краще використовувати в областях, де швидкість доступу до незмінних даних є важливою, наприклад, в деяких додатках для підтримки прийняття рішень.

Існують багато рішень на основі стовпців. Для подальших прикладів буде використовуватись СКБД Cassandra [8].

Модель даних Cassandra

Простір ключів у Cassandra приблизно еквівалентний схемі бази даних у реляційній СКБД і являє собою групу пов’язаних сімейств стовпців (рис. 9.26). Для горизонтального масштабування існує атрибут під назвою стратегія розміщення, який дозволяє користувачеві визначити, як розподілити репліки навколо вузлів, які він буде використовувати.

Сімейство стовпців приблизно еквівалентне таблиці в у реляційній СКБД. Кожне таке сімейство зберігається в окремому фізичному файлі, який відсортовано за ключами. Щоб зменшити кількість зчитувань з диска, а отже, підвищити продуктивність, стовпці, до яких регулярно звертаються разом, слід зберігати разом, в одному сімействі стовпців.

Стовпчик – це найменша одиниця зберігання даних у Cassandra. Стандартний стовпець складається з унікального імені (ключа), значення і мітки часу. Ключ ідентифікує рядок у сімействі стовпців. Мітка часу використовується для забезпечення вирішення конфліктів. Зазвичай вона визначається як різниця між поточним часом і 00:00:00 UTC 1 січня 1970 року, також відомого як епоха Unix. Рівень деталізації зазвичай вимірюється в мілісекундах або микросекундах. Мітка часу надається клієнтською програмою обробки. Це може бути проблемою у випадках, коли дані нестабільні, а точність клієнтської мітки часу невизначена. Вирішити цю проблему можна за допомогою зовнішнього сервера часу, але це може вплинути на продуктивність.

Суперстовпець – це контейнер для стовпців.

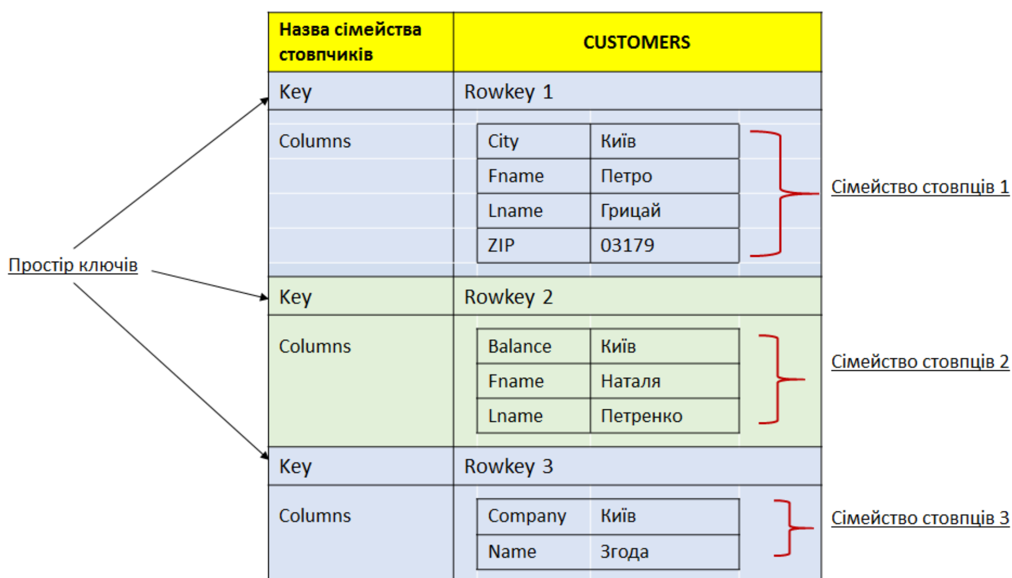


Рис. 9.26. Приклад стовпців в Cassandra

У Cassandra використовується мови запитів Cassandra Query Language (CQL), синтаксисом дуже схожим на SQL. У наступному прикладі (рис. 9.27) створюється простір ключів Flights і сімейство стовпців FlightDetails, а потім додаються дані з CSV-файлу DataOnly.csv:

```
CREATE KEYSPACE Flights WITH strategy_class = SimpleStrategy
AND strategy_opΘons:replicaΘon_factor = 1;              ← Створення простору ключів

use Flights;

create ColumnFamily FlightDetails                        ← Створення сімейства стовпців
(airline varchar PRIMARY KEY, Kms int, Noflights int, Hrs float, Pass int);

copy FlightDetails (airline, Kms, Noflights, Hrs, Pass) from 'DataOnly.csv';   ← Додавання даних з файлу

select * from FlightDetails;                             ← Перегляд даних
```

Рис. 9.27. Приклад програмної роботи з даними

В наведеному прикладі також визначається стратегія реплікації. Рекомендується, щоб коефіцієнт реплікації (кількість разів копіюється кожен рядок) був меншим або дорівнював кількості вузлів, які використовуються для зберігання реплікованих даних. В наведеному прикладі передбачається, що користувач працюватиме з автономною, одновузловою версією, тому встановлюється стратегія SimpleStrategy і коефіцієнт реплікації 1 – іншими словами, на одному вузлі існуватиме лише одна копія.

9.5.6. Графові сховища

Графи є важливим типом даних, що зберігаються в бази даних. Наприклад, комп'ютерна мережа з декількома маршрутизаторами і зв'язками між ними може бути змодельована у вигляді графа, де маршрутизатори є вершинами, а зв'язки – ребрами. Вебсторінки з гіперпосиланнями між ними є ще одним прикладом графів, де вебсторінки можна моделювати як вузли, а гіперпосилання між ними – як ребра. Насправді, якщо розглядаємо ER-модель підприємства, кожна сутність може бути змодельована як вузол графа, а кожен бінарний зв'язок може бути змодельований як ребро графа.

Хоча дані графів можна зберігати в реляційних базах даних, графові бази даних, такі як широко використовувана Neo4j, надають кілька додаткових можливостей:

- вони дозволяють визначати сутності як такі, що представляють вузли або ребра, і пропонують спеціальний синтаксис для визначення таких сутностей;
- вони підтримують мови запитів, призначені для легкого вираження шляхів до вузлів, які може бути важче виразити в SQL;
- запити виконуються набагато швидше, ніж якби вони були виражені на SQL і виконувалися у звичайній базі даних;
- вони забезпечують підтримку інших функцій, таких як візуалізація графіків.

У графових базах даних обхід зв'язків відбувається дуже швидко. Зв'язок між вузлами не обчислюється під час запиту, а зберігається як відношення. Обхід збережених

зв’язків відбувається швидше, ніж їх обчислення для кожного запиту. Вузли можуть мати різні типи зв’язків між собою. Оскільки кількість і тип зв’язків, які може мати вузол, не обмежені, всі вони можуть бути представлені в одній базі даних графів.

Розглянемо можливості графових баз даних на прикладі популярної СКБД Neo4J.

Модель даних Neo4J

Основні компоненти графової бази даних це вузли і ребра.

Вузли. Кожен вузол має унікальний ідентифікатор, назву та властивості, наприклад

```
node(ID, label, node data)
```

Зв’язки (ребра). Кожне ребро має ідентифікатори вузлів, що з’єднує, назву, напрямок та властивості

```
edge(fromID, toID, label, direct, edge data)
```

Зв’язки є основними елементами в графових базах даних; більша частина цінності графових баз даних походить від зв’язків. Зв’язки мають не лише тип, початковий та кінцевий вузли, але й власні властивості (рис. 9.28). Використовуючи ці властивості зв’язків, можна додати їм семантики – наприклад, з якого часу вони стали друзями, яка відстань між вузлами або які аспекти є спільними для вузлів. Ці властивості зв’язків можна використовувати для запитів до графа.

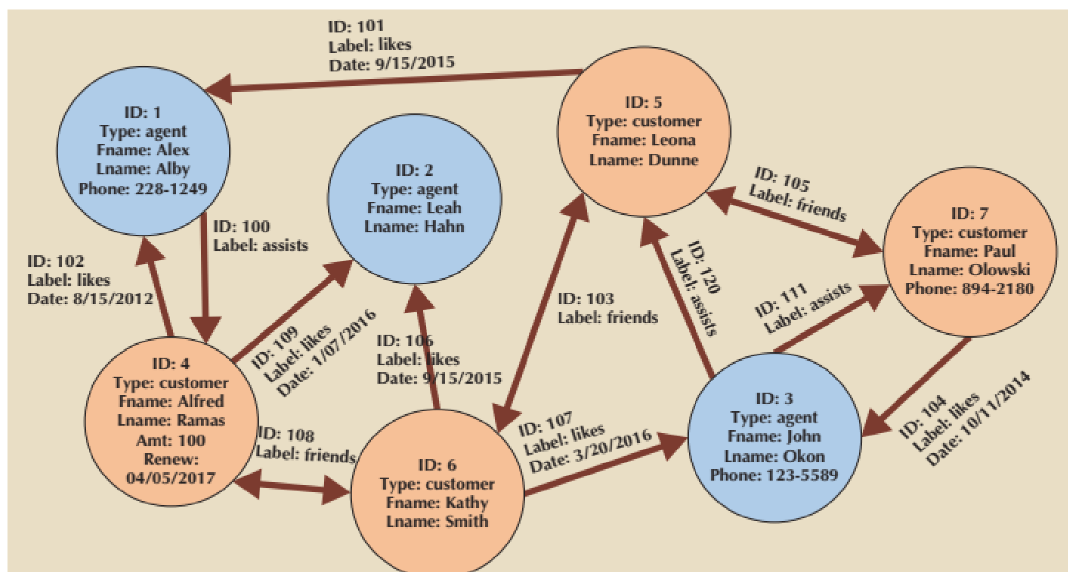


Рис. 9.28. Приклад графової моделі

Узгодженість. Оскільки бази даних графів містять інформації щодо з’єднання вузлів, більшість рішень для баз даних графів зазвичай не підтримують розподіл окремих вузлів на різних серверах. В межах одного сервера дані завжди узгоджені, особливо в Neo4J, який повністю сумісний з ACID. При роботі Neo4J в кластері, запис на головний сервер в кінцевому підсумку синхронізується з підлеглими, в той час як підлеглі завжди доступні для читання. Запис на підлеглі диски дозволено, і ті, що знаходяться на тому ж сервері негайно синхронізуються з головним; інші підлеглі диски синхронізуються не відразу – їм доведеться почекати, поки дані поширюються з головного. Графові бази даних забезпечують узгодженість за допомогою транзакцій. Вони не допускають

«висячих» зв’язків – початковий і кінцевий вузли завжди повинні існувати, і вузли можуть бути видалені лише тоді, коли до них не прикріплено жодних зв’язків.

Транзакції. СКБД Neo4J сумісна з ACID. Перш ніж змінювати будь-які вузли або додавати будь-які зв’язки до існуючих вузлів, необхідно запустити транзакцію. Якщо не зробити цього, виникне виключення NotInTransactionException. Операції читання можна виконувати без ініціювання транзакції.

Приклад команд роботи з базою даних Neo4J наведено на рис. 9.29

```
Node alex = graphDb.createNode();
alex.setProperty("name", "Alex");
Node petro = graphDb.createNode();
petro.setProperty("name", "Petro");
```

Створення графу з двох вузлів (Alex і Petro)

```
alex.createRelationshipTo(petro, FRIEND);
petro.createRelationshipTo(alex, FRIEND);
```

Створення двонаправленого зв'язку між ними:

```
Index<Node> nodeIndex = graphDb.index().forNodes("nodes");
nodeIndex.add(martin, "name", martin.getProperty("name"));
nodeIndex.add(pramod, "name", pramod.getProperty("name"));
```

Створення індексу та індексування вузлів

```
Node alex = nodeIndex.get("name", "Alex").getSingle();
allRelationships = alex.getRelationships();
```

Доступу до вузла і читання даних

Рис. 9.29. Приклад роботи з графовою базою даних

Переваги графових баз даних

Графові бази даних дозволяють побачити явні зв’язки між об’єктами. Вершини представляють сутності, і вони пов’язані або з’єднані ребрами. У реляційних базах даних зв’язки не представлені у вигляді посилань. Замість цього два об’єкти мають спільне значення атрибута, яке називається ключем.

Швидше виконуються запити за рахунок уникнення з’єднань. У базі даних графів замість того, щоб виконувати з’єднання, виконується обхід по ребрах від вершини до вершини. Це набагато простіша і швидша операція.

Спрощене моделювання. Робота з графовими базами даних може спростити процес моделювання. В реляційних базах даних, моделювання зазвичай починається з основних сутностей у предметній області. Коли моделюється інформація про взаємодію сутностей, схема БД може почати ускладнюватися. Наприклад, у соціальних мережах багато людей можуть вподобати пост, а пост може бути вподобаний багатьма людьми. Це називається відношенням «багато до багатьох», яке в реляційній бази даних моделюється у вигляді проміжної таблиці. У графовій базі даних немає необхідності створювати таблиці для моделювання зв’язків «багато до багатьох»; натомість вони явно моделюються за допомогою ребер.

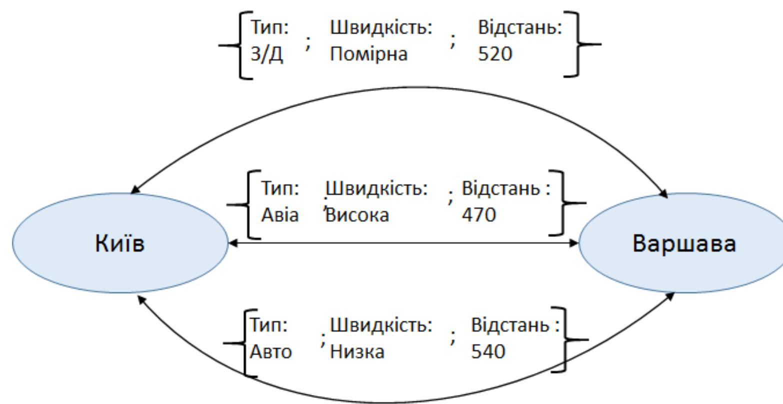


Рис. 9.30. Моделювання декількох типів зв'язків у графовій базі даних

Множинні зв'язки між сутностями. Використання декількох типів ребер дозволяє розробникам баз даних легко моделювати множинні зв'язки між сутностями. Це особливо корисно при моделюванні варіантів транспортних зв'язків. Наприклад, транспортна компанія може розглядати автомобільні, залізничні та повітряні перевезення між містами (рис. 9.30). Кожен з них має різні властивості, такі як час доставки, вартість та інші атрибути.

Контрольні запитання

1. Що таке 5V в Big Data? Коротко визначте кожну з них
2. Які чотири основні категорії баз даних NoSQL?
3. Як у документальних баз даних уникають з'єднань сутностей?
4. Опишіть та поясніть два основні компоненти MapReduce.
5. Опишіть способи моделювання ієрархій у базі даних документів
6. З яких компонентів складається граф?
7. Наведіть принаймні три приклади сутностей, які можна моделювати як вершини графа.
8. Наведіть принаймні три приклади зв'язків, які можна моделювати як ребра графа.
9. Які властивості можна асоціювати з вершиною, що представляє місто?
10. Які властивості ви можете асоціювати з ребром, що представляє шосе між двома містами?
11. Як бази даних графів уникають з'єднань?
12. Наведіть приклад бізнес-додатку, який би використовував кілька типів ребер (зв'язків) між вершинами.
13. Поясніть важливість реплікації даних.
14. Опишіть теорему CAP.
15. Опишіть особливості NoSQL СКБД, які забезпечують високу доступність, але не гарантують узгодженість.

Тестові завдання

1. Адміністратор хоче використати графову базу даних для побудови системи управління ідентифікацією та доступом. Який з наступних варіантів наведених нижче варіантів є прийнятним?
   A. Кассандра
   B. Neo4j
   C. MongoDB
   D. DynamoDB
2. Адміністратор працює над створенням бази даних для нового інтернет-магазину. Вона розробляє функцію, яка рекомендуватиме товари на основі схожих покупок та історії пошуку. Який тип бази даних обрати?
   A. Ключ-значення
   B. Документна
   C. Орієнтований на стовпці
   D. Графова
3. Якщо розробник використовує стовпчик-орієнтований дизайн, яка з наведених нижче баз даних найкраще відповідає його потребам?
   A. MySQL
   B. MongoDB
   C. Cassandra
   D. SQLite
4. Які з наведених нижче типів баз даних, найімовірніше, будуть використовуватися для відстеження зв'язків між людьми?
   A. Документальні
   B. Реляційні
   C. Графові
   D. Плоскі файли
5. Що з перерахованого нижче є стовпчиковим сховищем?
   A. Cassandra
   B. Riak
   C. MongoDB
   D. Redis
6. Які з наведених нижче баз даних є найпростішими базами даних NoSQL?
   A. Ключ-значення
   B. Стовпчикові
   C. Документні
   D. Всі перераховані
7. Сховища ________ використовуються для зберігання інформації про мережі, наприклад, соціальні зв'язки.
   A. Ключ-значення
   B. Широка колонка
   C. Документні
   D. Графові

8. Що таке теорема CAP, і які її наслідки для NoSQL баз даних?
   A. Вона стверджує, що розподілена система може гарантувати лише дві з трьох властивостей: Узгодженість, Доступність та Толерантність до розділів
   B. Доводить, що розподілена база даних завжди може забезпечити високу узгодженість і доступність
   C. Гарантує, що розподілена система може працювати з мережевими розділами без втрати даних
   D. Гарантує, що розподілена база даних завжди буде узгодженою та стійкою до розділів
9. Що таке шардинг в контексті NoSQL баз даних?
   A. Процес розбиття бази даних на декілька менших баз даних
   B. Метод реплікації даних між декількома серверами
   C. Метод рівномірного розподілу даних між кластером машин
   D. Спосіб забезпечення узгодженості даних між кількома центрами обробки даних
10. Які з наведених нижче технологій найчастіше використовуються у сфері великих даних?
   A. Hadoop і бази даних NoSQL
   B. MySQL, Oracle та SQL Server
   C. Excel, Access та Word
   D. C++, Java та Python
11. Яка з наведених нижче баз даних NoSQL класифікується як сховище ключ-значення?
   A. MongoDB
   B. Cassandra
   C. Redis
   D. CouchDB
12. У яких випадках найкраще використовувати базу даних NoSQL?
   A. Коли важливо забезпечити конфіденційність, цілісність і доступність даних
   B. Коли дані є передбачуваними
   C. Коли потрібно отримати велику кількість даних
   D. Коли швидкість отримання даних не є критичною
13. Ви розробляєте схему бази даних NoSQL для ігрової онлайн-платформи. Кожна гра має унікальний ідентифікатор, назву та список гравців. Кількість гравців може змінюватися для кожної гри. Яка з наведених нижче моделей даних буде найбільш придатною для зберігання цих даних?
   A. Сховище «ключ-значення»
   B. База даних документів
   C. База даних графів
   D. Стовпчикове сховище
14. Що з наведеного нижче є правильним?
   A. Той факт, що більшість баз даних NoSQL використовують підхід, заснований на евентуальній узгодженості, пояснюється теоремою CAP, яка стверджує, що

сильна узгодженість не може бути досягнута, коли необхідно забезпечити доступність і розбиття на частини

B. Реплікації в розподіленому середовищі NoSQL пов’язані зі створенням періодичних резервних копій бази даних на другу систему
C. Стабілізація пов’язана з часом очікування між запуском системи NoSQL і тим, коли система стає доступною для отримання запитів користувачів
D. Деякі реляційні конструкції, такі як зв’язок «багато до багатьох», важко виразити за допомогою графових баз даних

15. Яке з наступних тверджень не є правильним?
    A. Графи – це математичні структури, що складаються з вершин і ребер
    B. Графові моделі не здатні моделювати Зв’язки «багато до багатьох»
    C. Ребра в графах можуть бути одно- або двонаправленими
    D. Графові бази даних особливо добре працюють на деревоподібних структурах

10. БЕЗПЕКА БАЗ ДАНИХ

10.1. Базові концепції безпеки баз даних

Безпека баз даних – це комплекс інструментів, засобів контролю та заходів, призначених для створення та збереження конфіденційності, цілісності та доступності баз даних. Безпека баз даних має ті ж цілі та принципи, що й інформаційна безпека, але в межах або на рівні середовища баз даних. Це зосереджує увагу на виборі, розробці, впровадженні та конфігурації засобів безпеки, доступних на рівні бази даних. Управління безпекою баз даних – це захист даних від несанкціонованого доступу, запобігання витоку важливої інформації, а також уникнення втрати даних.

Безпека баз даних відповідає моделі CIA, де CIA означає конфіденційність (confidentiality), цілісність (integrity) і доступність (availability).

Конфіденційність вимагає, щоб тільки авторизовані користувачі мали доступ до інформації, щоб зберегти дані приватних осіб, інтелектуальну власність бізнесу та національну безпеку. Збереження конфіденційності передбачає використання відповідних методів шифрування, а також процедур авторизації, ідентифікації та автентифікації користувачів.

Цілісність вимагає, щоб тільки авторизовані користувачі могли змінювати дані, таким чином підтримуючи узгодженість і достовірність даних. Якщо дані є некоректними, вони більше не є корисними. Неправильні дані також можуть завдати шкоди окремим особам (наприклад, неправильні дані в кредитному звіті) та організаціям (наприклад, недійсні фінансові звіти).

Доступність вимагає, щоб інформація була доступною для авторизованих користувачів, коли це необхідно. Атаки на безпеку організації можуть спричинити недоступність бізнес-послуг, що призведе до порушення угод про рівень обслуговування, які є критично важливими для бізнес-операцій.

Політика безпеки. Політика інформаційної безпеки це сукупність правил і практичних заходів, що стосуються захисту інформаційних активів організації. .Метою політики безпеки є забезпечення конфіденційності, цілісності та доступності в контексті конкретної БД. До мети захисту даних можна підійти двома різними і взаємодоповнюючими способами.

Профілактика. Профілактика гарантує, що порушення безпеки не може статися. Основний метод полягає в тому, що система вивчає кожну дію і перевіряє її відповідність політиці безпеки, перш ніж дозволити її виконання. Цей метод називається контролем доступу.

Виявлення. Виявлення гарантує, що історія активності в системі записується в аудиторський журнал, достатньо, щоб можна було виявити порушення безпеки постфактум. Цей метод називається аудитом.

Кожна СКБД використовує певну суміш цих двох методів. У цьому розділі розглянемо переважно превентивні методи, оскільки профілактика є більш фундаментальною технікою.

10.2. Загрози безпеці баз даних

Зважаючи на широкий спектр можливих інструкцій та операцій мови SQL, зловмисник, може мати широкий вибір інструментів, які можна використовувати для пошкодження баз даних, збережених даних та додатків, що використовують ці дані, різними способами. Наприклад, одна проста інструкція, така як DROP DATABASE <*ім'я бази даних*>, може повністю скомпрометувати функціональність програми, яка покладається на бази даних для запиту даних або навіть даних автентифікації.

З цієї причини код SQL ніколи, принаймні безпосередньо, не призначений для взаємодії всередині програми. Замість цього, саме застосунок, на основі даних, введених користувачем, готує SQL-код, який потрібно відправити до бази даних, щоб витягти (або змінити) запитувані дані. Однак, існують способи, за допомогою яких потенційні зловмисники можуть зловживати синтаксисом SQL і вставляти довільні інструкції. Цей спосіб атаки загалом називається «SQL-ін'єкція» (SQL injection) і полягає у вставці коду SQL, в існуючий код, що дозволяє виконувати непередбачені завдання.

Основна проблема використання SQL полягає в тому, що код оцінюється додатком під час його роботи: якщо відсутні елементи контролю, то програма, яка вже запущена, не оцінює оператори з точки зору змісту або коректності. Зловмисник може скористатися цим, вставляючи довільні команди у введені користувачем дані, наприклад, у формах автентифікації або рядкових полях, вставляючи їх у запущений код.

10.2.1. SQL ін'єкції

Застосунки для роботи з базами даних повинні вживати заходів безпеки, щоб захистити базу даних від форми атаки, відомої як SQL-ін'єкція. Термін «ін'єкція» означає, що користувацьке введення з боку клієнта через інтерфейс програми може бути спроєктоване таким чином, щоб скористатися вразливостями, пов'язаними з динамічною побудовою SQL-запитів. Використовуючи SQL-ін'єкцію, зловмисник може вставити (або впровадити) в запит код, який може бути використаний для отримання інформації, яку зловмисник не має права бачити, зловмисного видалення або модифікації даних, або вставити дані, які дадуть зловмиснику несанкціонований доступ до бази даних.

Вперше SQL-ін'єкції були виявлені приблизно в 1998 році, і зараз вони є однією з найбільших проблем безпеки програмного забезпечення згідно з проєктом Open Web Application Security Project та списком Common Weakness Enumeration Top 25 Most Dangerous Software Errors (25 найнебезпечніших помилок у програмному забезпеченні). SQL-ін'єкції загрожують конфіденційності, цілісності, доступності, автентифікації та авторизації.

Атаки SQL ін'єкцій можуть бути організовані в різний спосіб [8].

Пряма SQL ін’єкція

Зловмиснику відома структура бази даних і він може модифікувати легітимний запит (рис. 10.1):

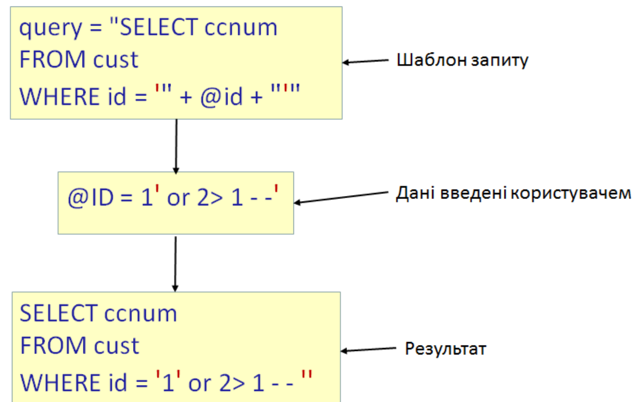


Рис. 10.1. Пряма SQL ін’єкція

Сліпа SQL ін’єкція

Якщо вебзастосунок не надає зловмиснику засобів для безпосереднього перегляду структури бази даних (зазвичай це так) можна використати сліпу SQL ін’єкцію. Сліпа логічна SQL ін’єкція (Blind Boolean-Based SQL Injection) використовує результат перевірки логічної умови (рис. 10.2) щодо структури таблиці.

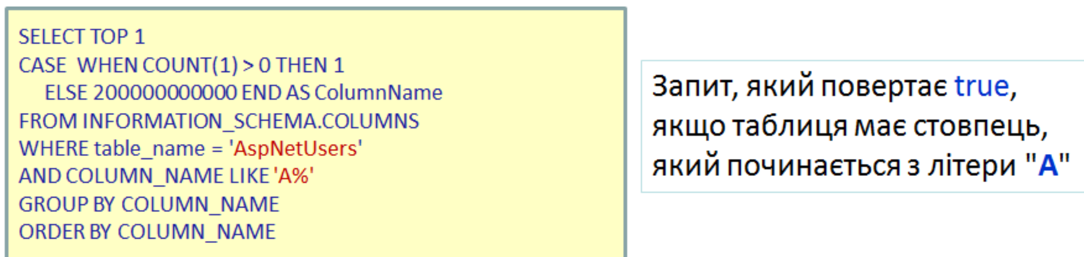


Рис. 10.2. Сліпа логічна SQL ін’єкція

Сліпа SQL ін’єкція на основі вмісту (Blind Content-Based SQL Injection) (рис. 10.3) використовується для перевірки вразливості застосунку до ін’єкцій.

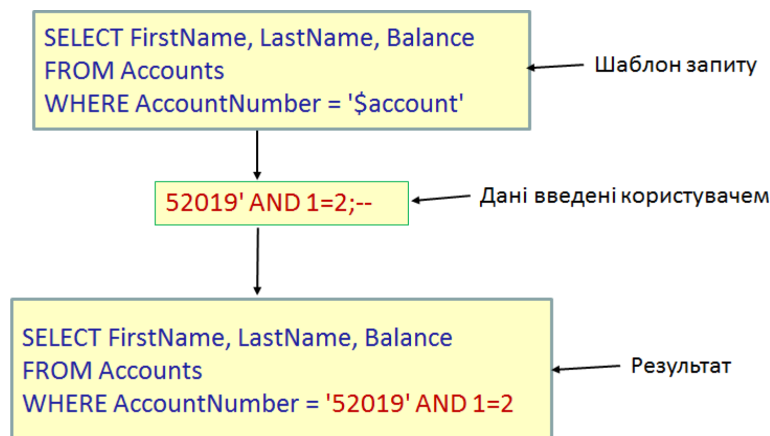


Рис. 10.3. Сліпа SQL ін’єкція на основі вмісту

Запит на рис. 10.3, звичайно, ніколи не повертає жодних результатів. Якщо зловмисник бачить порожню сторінку замість попередження, він може бути досить

впевнений, що програма вразлива до сліпої ін’єкції SQL, а потім може спробувати зробити більше шкідливих запитів.

Сліпа SQL ін’єкція на основі синхронізації (Blind Timing-Based SQL Injection). В цьому випадку зловмисник перевіряє гіпотезу щодо структури таблиці за допомогою часової затримки (рис. 10.4). У більшості випадків затримки запиту на 5 або 10 секунд достатньо, щоб запит повернув True, якщо гіпотезу підтверджено.

```
IF EXISTS (SELECT *
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name =
'AspNetUsers' AND COLUMN_NAME LIKE 'A%')
BEGIN
WAITFOR DELAY '00:00:05'
END--%
```

Рис. 10.4. Сліпа SQL ін’єкція на основі синхронізації

В нашому прикладі використовується функція WAITFOR() SQL Server. Аналогічні функції підтримуються і в інших СКБД, наприклад функція SLEEP() в MySQL і Oracle.

Ін’єкція другого порядку (Second order SQL injection)

Розглянуті приклади відносяться до ін’єкцій першого порядку. Зловмисник вводить зловмисний рядок і наказує негайно його виконати. Під час ін’єкційної атаки другого порядку зловмисник вводить шкідливий рядок, який є досить стійким і прихованим. Цей рядок виконується згодом. Дані, надані зловмисником, зберігаються програмою та пізніше включаються в запити SQL небезпечним способом.

В наступному прикладі (рис. 10.5) значення в таблиці USERS вважаються вірними і без перевірки копіюються в іншу таблицю:

а)

```
$firstname = someEscapeFunction($_POST[”firstName”]);
$SQL = ”INSERT INTO USERS (firstname) VALUES (’{$firstName }’);”;
someConnection->execute($SQL);
```

б)

```
Alex’); DELETE FROM USERS; //
```

← Користувач

в)

```
$userid = 42;
$SQL = ”SELECT firstname FROM USERS WHERE
(userId={$userid})”;
$RS = con->fetchAll($SQL);
$firstName = $RS[0][”firstName”];

$SQL = ”INSERT INTO Staff VALUES (’{$firstName}’);”;
```

г)

```
INSERT INTO Staff VALUES (’ Alex’); DELETE FROM USERS; //
```

Рис. 10.5. SQL ін’єкція другого порядку

Розробник (рис. 10.5 а) створює скрипт, що додає в таблицю додається рядок з даними, отриманими від користувача. На думку розробника, SQL injection неможлива, оскільки функція someEscapeFunction захищає від атак шляхом видалення escape-символів

(символів в рядках, які мають спеціальне значення в SQL), наприклад *NUL (ASCII 0), \n, \r, \, ', ", Control-Z.*

Але (рис. 10.5 б) користувач задає свої дані в вигляді, що не передбачено розробником. Оскільки розробник впевнений що дані коректні, тому без додаткової перевірки такий рядок зчитується і додається в іншу таблицю (рис. 10.5 в). В подальшому, коли дані зчитаються з таблиці Staff, і використовуються як параметри іншої команди, доданий код буде виконано (рис. 10.5 г).

Захист від ін’єкцій

Основна вразливість полягає в динамічній побудові SQL-запиту у вигляді рядка. Такого підходу до побудови запиту в коді програми слід уникати.

Альтернативним підходом є використання параметризованих операторів, як у підготовлених операторах API. Використання параметризованих запитів змушує значення змінних, які використовуються для побудови запиту, відповідати значенням певного типу, а не довільному рядку, який може містити зловмисні SQL-оператори.

Використання збережених процедур також може бути використано для уникнення атак SQL-ін’єкцій, якщо SQL-запити не конструюються динамічно в збереженій процедурі.

Інший спосіб запобігання – завжди перевіряти дані, що вводяться користувачем, щоб переконатися, що вони відповідають допустимим типам і шаблонам, перш ніж використовувати їх для побудови SQL-запиту.

В додатку слід ретельно фільтрувати метасимволи, тобто символи, які SQL розглядає як функції, а не як дані. В табл. 10.1 наведені такі символи.

Таблиця 10.1 Метасимволи SQL

Символ	Значення	Символ	Значення
‘	Початок або кінець рядка	‘ OR ‘1	Формування умови виконання запиту
‘ ’		‘ OR ---	
%00	Нуль-байт (поверне помилку)	“ OR “” = “	
--	коментар	“ OR 1 = 1 ---	
---		‘ OR ‘’ = ‘	
“ ”	Початок або кінець рядка	OR 1=1	
;	кінець рядка	% _	Шаблони підстановки

Підбиваючи підсумок, можна сформулювати такі заходи захисту від ін’єкцій:

1. Натомість динамічного формування інструкцій використовувати підготовлені інструкції;

2. Проводити перевірку типу даних, що вводяться, і обмежувати число символів, що вводяться.

3 Використовувати збережені процедури та функції.

4. Застосовувати функції, які автоматично усувають потенційно небезпечні символи або послідовності символів, що містять термінатори інструкцій, коментарі, одиничні лапки та символи xp_ (для SQL Server) з даних, що вводяться.

.5. Ретельно визначати права доступу, щоб інструкції не мали дозволів на запуск DDL інструкцій.

6. Вимикати видачу повідомлень про необроблені помилки.

Бази даних NoSQL також схильні до ін’єкцій – хоча вони менш поширені, але можуть бути не менш небезпечними.

10.2.2. Інші види загроз базам даних

Надмірні привілеї користувачів

Коли комусь надаються привілеї до бази даних, які перевищують вимоги його посадових обов’язків, цими привілеями можуть зловживати. Наприклад, коли працівник змінює посаду в організації, часто його права доступу до конфіденційних даних не змінюються. Надмірні привілеї зазвичай виникають через те, що механізми контролю привілеїв для робочих ролей не були чітко визначені або не підтримувалися. В результаті користувачі можуть отримати привілеї доступу, які значно перевищують їхні конкретні посадові обов’язки, або ж вони можуть просто накопичувати такі привілеї з часом.

Неактивні облікові записи також можуть становити ризик, якщо зловмисник знає про їхнє існування. Бази даних можуть мати вразливості, які дозволяють зловмисникам підвищити привілеї в маловідомому обліковому записі з низьким рівнем привілеїв і отримати доступ до прав адміністратора.

Зловживання законними привілеями

Користувачі можуть зловживати законними привілеями бази даних у несанкціонованих цілях. Наприклад, якщо доступ користувачі до даних обмежується інтерфейсом вебзастосунку, зловмисник може обійти ці обмеження, підключившись до бази даних за допомогою альтернативного клієнта, наприклад, MS-Excel. Використовуючи Excel і свої законні облікові дані для входу, користувач може отримати і зберегти всі дані без обмежень.

Слабка автентифікація

Слабка автентифікація має багато аспектів, починаючи від можливості підбору паролю користувача і закінчуючи незахищеним зберіганням облікових даних в застосунку, що використовуються.

Пароль – це, по суті, головний ключ до всієї системи та всіх її файлів. Але, за дослідженням[8], слабкі паролі є причиною понад 80% витоків даних в організаціях. До 30% витоків даних в організаціях спричинені тим, що окремі користувачі ділять паролі між собою, повторно використовують паролі або потрапляють на фішингові атаки.

Організація може мати сотні або навіть тисячі баз даних. І тримати їх усі під контролем може бути досить складно. Дуже хорошим кроком на шляху до кращого

[8] https://techreport.com/statistics/cybersecurity/password-reuse-statistics/

захисту баз даних є видалення стандартних, порожніх і слабких облікових даних для входу в систему. Хакери зазвичай відстежують облікові записи за замовчуванням і використовують їх для несанкціонованого доступу до бази даних.

Атаки соціальної інженерії, такі як фішинг, можуть бути використані для крадіжки облікових даних для входу в систему, які зловмисник може використати для доступу до мережі та бази даних.

Відсутність оновлень

Згідно зі звітом Microsoft Security Intelligence Report[9], щороку з'являється від 5000 до 6000 нових вразливостей. Це означає, що щодня з'являється щонайменше 15 нових вразливостей, які в основному націлені на слабкі місця в системі. Постачальники програмного забезпечення згодом реагують на них виправленнями. Але адміністратори баз даних часто занадто зайняті, щоб встигати за усіма випусками.

За даними дослідників, більше третини перевірених баз даних не мають оновлень безпеки або працюють на застарілих версіях програмного забезпечення. У багатьох випадках у більшості з цих систем не було патчів безпеки для баз даних більше року тому. Безумовно, це вина власників та адміністраторів баз даних, яким з певних причин важко встановити відповідні патчі.

Чим довше база даних працює з відсутніми виправленнями, тим більше вона схильна до розвитку шкідливого програмного забезпечення.

Неадекватна реєстрація і аудит

Ведення журналів та аудит є ключовими для запобігання та виявлення зловживань, а також для забезпечення належного розслідування підозр на компрометацію даних. У цьому контексті, реєстрація – це збір даних, а аудит – це перевірка даних.

Приблизно у третині баз даних аудит баз даних або відсутній, або неправильно налаштований. Неможливість всебічного моніторингу даних на всіх рівнях є серйозною вразливістю на багатьох рівнях.

Типові недоліки організації аудиту:

- Відсутність обліку дій конкретних користувачів – коли користувач звертається до бази даних через вебзастосунок, вбудовані механізми аудиту не розпізнають конкретний ідентифікатор користувача. В цьому випадку, всі дії користувача зв'язується з ім'ям облікового запису вебзастосунка.
- Спрощення або повністю відключення підсистеми аудиту з метою підвищення продуктивності ІС.
- Користувачі з правами адміністратора бази даних можуть просто вимкнути підсистему аудиту, щоб приховати шахрайську діяльність.
- Вбудовані механізми аудиту зазвичай не записує деталі , необхідні для виявлення атак, їх розслідування та відновлення системи (наприклад, IP- адреса джерела, атрибути запиту, невдалі спроби запитів і ін.).

---

[9] https://www.microsoft.com/en-us/security/security-insider/microsoft-digital-defense-report-2023

- Механізми аудиту різних СКБД є унікальними. Для організацій, що використовують різні СКБД, практично виключається можливість впровадження єдиних , масштабованих процесів аудиту всього підприємства.

Вразливості резервних копій баз даних

Втрата даних є потенційно катастрофічною для організації-власника. Насправді, 43% підприємств, які зазнають подібного, ніколи не відкриваються знову, а 51% з них зрештою закриваються через два роки. Зазвичай, коли власники думають про захист баз даних, вони думають про оригінальну базу даних, яку вони хочуть захистити, і забувають про резервні копії баз даних, до безпеки яких слід ставитися з такою ж серйозністю і ретельністю. Але, незважаючи на це, багато підприємств все ще використовують вразливі процедури резервного копіювання. Типові вразливості:

- Носії резервних копій незахищені від крадіжки;
- Зміст резервних копій не шифрується;
- Резервні копії в одному екземплярі;
- Не проводиться регулярний аудит резервних копій.

Експлуатація вразливих, неправильно налаштованих баз даних

Бази даних мають багато різних варіантів конфігурацій і параметрів , доступних DBA для точного налаштування продуктивності та розширених функцій. Деякі конфігурації можуть бути небезпечними, з багатьма вразливостями. Як правило, це конфігурації і облікові записи, увімкнені за замовчуванням або ввімкнені для зручності адміністраторів баз даних або розробників застосунків.

Кожна інсталяція бази даних постачається з пакетами доповнень усіх форм і розмірів, які здебільшого не будуть використані. Оскільки мета безпеки баз даних полягає в тому, щоб зменшити поверхню атаки, потрібно шукати пакети, які не використовуються, і відключати або видаляти їх. Це не лише зменшує ризики атак нульового дня через ці пакети, але й спрощує управління виправленнями. Коли саме ці пакети потребують виправлень, вашій організації не потрібно буде їх встановлювати.

IP-адреси сервісів постійно скануються зловмисниками на предмет пошуку вразливостей. Зловмисники знають, як використовувати ці вразливості для здійснення атак на базу даних. Типова ситуація – високе робоче навантаження та накопичення нерозв'язаних завдань для відповідних адміністраторів баз даних, складні та трудомісткі вимоги до тестування виправлень, а також проблеми з пошуком вікна для технічного обслуговування, щоб вимкнути та попрацювати над тим, що часто класифікується як критично важлива для бізнесу система. В результаті, як правило, організаціям потрібні місяці, щоб виправити бази даних, і протягом цього часу вони залишаються вразливими.

Відмова в обслуговуванні

Відмова в обслуговуванні (DoS) – це загальна категорія атак, в яких доступ до баз даних або даних заборонений для запланованих користувачів. Умови DoS можуть бути створені багатьма методами. Найпоширенішим методом, який використовується в середовищах баз даних, є перевантаження ресурсів сервера, таких як пам'ять і процесор, або за допомогою надмірної кількості запитів, або за допомогою меншого обсягу добре

продуманих запитів, які споживають непропорційно велику кількість системних ресурсів (наприклад, через те, що вони призводять до рекурсивного пошуку або операцій з таблицями). Розподілена атака на відмову в обслуговуванні (DDoS) використовує ботнет для створення величезного обсягу трафіку.

Незашифровані дані

Шифрування даних є фундаментальним і найважливішим компонентом будь-якої політики безпеки баз даних. Усі конфіденційні дані повинні бути зашифровані. Таким чином, навіть якщо якісь дані будуть викрадені, шифрування гарантує, що вони стануть непридатними для використання.

Хоча шифрування стало стандартом під час передачі даних, деякі підприємства все ще не впровадили його для інформації, що зберігається в їхніх базах даних.

Організації ніколи не повинні зберігати конфіденційні дані у вигляді відкритого тексту в таблиці бази даних. Це стосується як оригінальної бази даних, так і її копій. І всі з'єднання з базою даних повинні завжди використовувати шифрування.

10.3. Механізми безпеки баз даних

Механізм захисту інформації – це сукупність методів, засобів і процедур, спрямованих на запобігання несанкціонованому доступу, витокам, змінам, знищенню або спотворенню даних. Він включає в себе технічні, програмні та організаційні заходи, які забезпечують конфіденційність, цілісність і доступність інформації. Механізми захисту в базах даних включають автентифікацію користувачів, управління доступом, резервне копіювання, моніторинг активності, шифрування даних та інші методи, що сприяють забезпеченню безпеки інформації.

До найважливіших механізмів захисту баз даних відносяться:

- Ідентифікація і автентифікація.
- Авторизація і контроль доступу.
- Реєстрація і аудит.
- Шифрування.

Всі механізми пов'язані між собою і активно взаємодіють, як показано на рис. 10.6.

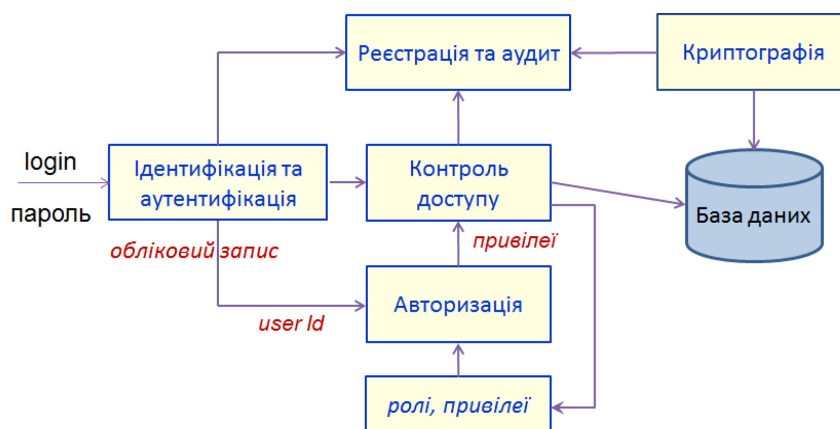


Рис.10.6. Комплекс основних механизмів захисту інформаціїї в базах даних

Розглянемо особливості реалізації механізмів засобами СКБД.

10.3.1. Ідентифікація і автентифікація

Автентифікація бази даних – це процес підтвердження того, що користувач, який намагається увійти до бази даних, має на це дозвіл. У разі успішної автентифікації для даного користувача встановлюється з'єднання, і всі дії та запити до даних, що виконуються в рамках цього з'єднання, контролюються механізмом управління доступом відповідно до привілеїв, визначених для цього користувача.

Таким чином, автентифікація користувачів захищає базу даних від несанкціонованого доступу і допомагає впроваджувати політики контролю доступу. Політики контролю доступу допомагають обмежити доступ до даних до мінімуму, необхідного для кожної ролі користувача, а також запобігти зловживанням і несанкціонованому використанню даних.

Щоб впровадити автентифікацію користувачів, потрібно визначити ролі користувачів і політики контролю доступу, вибрати метод автентифікації та налаштувати параметри автентифікації. Наприклад, при створенні бази даних в Microsoft SQL Server користувач повинен визначити, чи використовувати автентифікацію бази даних, автентифікацію операційної системи або обидва способи (так звана змішана автентифікація).

Особливості реалізації автентифікації засобами СКБД

Особливості реалізації автентифікації засобами систем керування базами даних (СКБД) включають низку специфічних аспектів, пов'язаних із забезпеченням безпеки та управління доступом безпосередньо на рівні СКБД.

Вбудовані механізми автентифікації. Більшість СКБД (таких як MySQL, PostgreSQL, Oracle, SQL Server) мають вбудовані засоби для автентифікації користувачів. Вони дають змогу створювати, керувати та перевіряти облікові записи користувачів безпосередньо в СКБД. Автентифікація заснована на перевірці імені користувача і пароля, де паролі зберігаються в зашифрованому вигляді в системних таблицях СКБД.

Підтримка різних схем автентифікації. СКБД часто підтримують кілька схем автентифікації. Наприклад, PostgreSQL підтримує автентифікацію на основі паролів, Kerberos, GSSAPI, LDAP, а також автентифікацію за довіреним хостом (trust) і через SSL-сертифікати. Oracle Database може використовувати стандартні схеми автентифікації, такі як пароль або зовнішня автентифікація через Windows (OS Authentication) і навіть автентифікація з використанням PKI-сертифікатів.

Автентифікація на рівні операційної системи. Деякі СКБД підтримують автентифікацію користувачів на рівні операційної системи. Це дає змогу користувачам з'єднатися з базою даних без необхідності введення додаткових облікових даних, якщо вони вже пройшли автентифікацію в системі. Прикладом є SQL Server, який підтримує Windows Authentication, де облікові записи користувачів бази даних можуть бути пов'язані з обліковими записами Windows.

Реалізація багатофакторної автентифікації (MFA). Деякі сучасні СКБД підтримують або дають змогу впроваджувати багатофакторну автентифікацію, додаючи додаткові рівні безпеки крім пароля. Приклади можуть включати використання додаткових факторів, таких як OTP (One-Time Passwords) або інтеграція з MFA-системами через плагіни.

Можливість інтеграції з зовнішніми службами автентифікації. Багато СКБД дають змогу інтегрувати автентифікацію із зовнішніми системами і службами, такими як LDAP, Active Directory, Kerberos. Це дає змогу використовувати єдину точку управління користувачами та їхніми правами доступу в усій організації. Прикладом може слугувати MySQL, який підтримує автентифікацію з використанням плагінів PAM (Pluggable Authentication Modules) і LDAP.

Облікові записи

Найпопулярніший спосіб автентифікації в базах даних за допомогою імені облікового запису та пароля. Облікові записи користувачів – це унікальні імена і паролі, які використовуються для визначення користувача або клієнтського застосунку, що підключився до бази даних. Права доступу до бази даних і набору даних, видані або відкликані у конкретного користувача, визначаються налаштуванням облікового запису.

Обов'язковими обліковими записами є облікові записи DBA, такі як обліковий запис sa для баз даних Microsoft SQL Server, адміністративний обліковий запис SYSTEM для баз даних Oracle, або будь-який інший обліковий запис, який використовується для адміністрування баз даних. Зазвичай буває багато інших адміністративних або внутрішніх облікових записів баз даних, які потребують спеціальних привілеїв або мають доступ до конфіденційної інформації.

Зазвичай сучасні СКБД мають графічний інтерфейс для виконання адміністративних функцій, але можна використовувати оператор SQL `CREATE USER`:

```
CREATE USER 'username' [IDENTIFIED BY 'password']
```

Оператор `CREATE USER` має бути виконаний адміністратором бази даних або іншим адміністративним облікових записів бази даних, якому надано право виконувати ці команди.

Політика безпеки щодо паролів

В випадку, коли автентифікація здійснюється за допомогою паролю, DBA повинен встановити політику безпеки що до паролів. Розробники СКБД рекомендують параметри такої політики для користувачів. Наприклад, Oracle надає такі рекомендації[10]:

- пароль користувача повинен містити щонайменше 9 символів;
- пароль адміністратора повинен містити щонайменше 15 символів;
- паролі повинні містити щонайменше 2 літери нижнього регістру;
- паролі повинні містити щонайменше 2 літери верхнього регістру;
- паролі повинні містити щонайменше 2 цифрові символи;

[10] https://docs.oracle.com/cd/E95618_01/html/sbc_scz810_adminsecurity/GUID-5E974486-498E-4369-892C-E214D375AE1C.htm#Password-Policy

- паролі повинні містити щонайменше 2 спеціальні символи;
- пароль повинен відрізнятися від попереднього щонайменше на 4 символи;
- паролі не можуть містити, повторювати або перевертати ім’я користувача;
- паролі не можуть містити три однакові символи підряд.

Рекомендації NIST SP 800-63B[11] щодо цифрової ідентифікації

Рекомендації NIST SP 800-63B від 2017 року суттєво змінює вимоги до політики безпеки паролів.

- Термін дії паролів не повинен закінчуватися. Користувачі не повинні вимагати змінювати свої паролі регулярно, наприклад, кожні 30 днів. Користувачі часто змінювали один символ під час примусу щоб змінити свій пароль.
- Користувачі не повинні обов’язково використовувати спеціальні символи. Вимагання від користувачів включати спеціальні символи важко запам’ятовувати, тому часто вони записували ці паролі.
- Користувачі повинні мати можливість копіювати та вставляти паролі. Менеджери паролів дозволяють користувачам створювати та зберігати складні паролі.
- Користувачі повинні мати можливість використовувати всі символи. Механізми зберігання паролів зазвичай мають відхилити пробіли та деякі спеціальні символи. Дозволяючи пробіли, користувачі можуть створювати довші паролі, які легше запам’ятати. Системи іноді відхиляють спеціальні символи, щоб запобігти атакам (наприклад, атака ін’єкції SQL), але належне хешування пароля маскує тики символи.
- Довжина пароля повинна бути не менше восьми символів і не більше 64 символів. Більша довжина дозволяє користувачам створювати значущі для них парольні фрази.
- Системи автентифікації повинні перевіряти паролі. Перш ніж прийняти пароль, система повинна звіряти їх зі списком часто використовуваних паролів, наприклад 12345 пароль.
- Відмовтеся від будь-яких паролів, які знаходяться в чорному списку. Чорний список повинен включати наступне:
  - Паролі, отримані з попередніх випадків порушень.
  - Словникові слова.
  - Повторювані або послідовні символи (наприклад, aaaaaa, 1234abcd).
  - Контекстні слова, такі як ім’я користувача та похідні від них.
  - Будь-які тривіальні зміни пароля з чорного списку

Вбудовані в СКБД засоби автентифікації можуть не підтримувати деякі з зазначених вимог. В такому випадку, безпечну автентифікацію мають забезпечувати застосунки, що працюють з БД.

[11] https://pages.nist.gov/800-63-3/sp800-63b.html

10.3.2. Авторизація і контроль доступу

Авторизація в базах даних – це процес, який контролює, які дії можуть виконувати користувачі після успішної автентифікації. Засобами СКБД процес авторизації здійснюється через управління правами доступу та ролями користувачів. Контроль доступу до бази даних – це процес забезпечення того, щоб доступ до даних або інших ресурсів здійснювався лише дозволеними способами, відповідно до прав доступу, наданих в процесі авторизації.

Деякі з ключових причин, чому контроль доступу є настільки важливим для безпеки баз даних:

- Запобігання витоку даних – несанкціонований доступ до конфіденційної інформації, такої як дані клієнтів, може призвести до катастрофічного витоку даних. Детальний контроль доступу мінімізує цей ризик.
- Забезпечення розподілу обов'язків – доступ можна обмежити на основі ролей, щоб запобігти перекриттю або конфлікту привілеїв.
- Захист інтелектуальної власності – контроль доступу гарантує, що запатентовані дані залишаються захищеними.
- Забезпечення підзвітності – обмежуючи та перевіряючи доступ, можна відстежити дії конкретних користувачів.

Контроль доступу базується на концепції прав доступу, або привілеїв, та механізмах надання суб'єктам доступу таких привілеїв. Суб'єктом доступу може бути користувач або роль. Об'єктом доступу можуть бути дані, або інші об'єкти БД, такі як подання, збережені процедури та інші. Привілей дозволяє отримати доступ до деякого об'єкта бази даних певним чином (наприклад, читати, модифікувати або виконувати). Користувач, який створює об'єкт бази даних, наприклад, таблицю або подання, автоматично отримує всі відповідні привілеї на цей об'єкт. Згодом СКБД відстежує, як ці привілеї надаються іншим користувачам, і гарантує, що в будь-який час тільки користувачі з необхідними привілеями можуть отримати доступ до об'єкта.

Ключові етапи й особливості авторизації та контролю доступу:

1. Автентифікація користувача. Після успішної автентифікації СКБД визначає обліковий запис користувача і переходить до етапу авторизації.

2 Визначення ролі користувача. Роль являє собою набір прав і привілеїв, що визначають доступ користувача до об'єктів бази даних (таблиць, подання, процедур тощо). Наприклад, роль «DBA» (Database Administrator) може мати повний доступ до всіх об'єктів бази даних, тоді як роль «Public» може мати доступ тільки для читання певних таблиць. У більшості СКБД користувачі можуть бути призначені на одну або кілька ролей. У деяких СКБД ролі можуть успадковувати привілеї одна від одної. Наприклад, роль «Менеджер» може включати всі привілеї ролі «Співробітник» плюс додаткові привілеї. Це дає змогу створювати ієрархії ролей, де вищі ролі автоматично отримують привілеї нижчих ролей, спрощуючи управління доступом і знижуючи вірогідність помилок під час призначення прав.

3. Визначення прав доступу. СКБД перевіряє, які права доступу пов’язані з користувачем або його ролями. Це може включати права на читання, запис, зміну або видалення даних, виконання збережених процедур, створення і видалення об’єктів бази даних та інші дії. Привілеї можуть бути призначені як на рівні бази даних загалом, так і на рівні окремих схем, таблиць або навіть рядків даних (якщо підтримується політика контролю доступу на рівні рядків, як в Oracle). У деяких СКБД можуть використовуватися політики доступу, які дають змогу задавати умови для виконання певних дій. Наприклад, доступ до даних може залежати від часу, місця розташування користувача або інших умов.

4. Контроль доступу до об’єктів бази даних. СКБД визначає, чи має користувач права на виконання конкретної операції з конкретним об’єктом бази даних. Наприклад, якщо користувач намагається виконати запит на вибірку даних з таблиці, система перевіряє, чи є у користувача привілей SELECT на цю таблицю. У разі, якщо користувач не має необхідних прав, СКБД блокує виконання операції і може повернути помилку «доступ заборонено» або аналогічне повідомлення.

Види привілеїв доступу в базах даних

Перш ніж користувач зможе отримати доступ до бази даних, йому необхідно призначити або надати привілеї. Привілеї можна призначити за допомогою оператора SQL GRANT. Інструкція GRANT охоплює широкий спектр привілеїв, які можуть просто або комплексно керувати доступом користувача до бази даних, її таблиць, стовпців і так далі. Види привілеїв, які зазвичай призначаються, показані на рис. 10.7.

SQL привілеї	Пояснення
ALL	дозволяє користувачеві мати всі наступні привілеї
SELECT	дозволяє користувачеві отримувати дані з заданої таблиці (таблиць)
UPDATE	дозволяє користувачеві змінювати існуючі дані в заданій таблиці (таблицях)
INSERT	дозволяє користувачеві додавати рядки до заданої таблиці (таблиць)
DELETE	дозволяє користувачеві видаляти рядки з заданої таблиці (таблиць)
CREATE	дозволяє створювати нові таблиці та бази даних
DROP	дозволяє видаляти цілі таблиці та бази даних
REFERENCES	Право визначати зовнішні ключі в інших таблицях, які посилаються на вказаний стовпець таблиці об'єкта

Рис. 10.7. Привілеї і їх визначення мовою SQL

Призначення привілеїв

Плануючи доступ, DBA може використовувати матрицю контролю доступу до бази даних, як показано на рис. 10.8. Заголовки стовпчиків представляють об’єкти бази даних, які можуть бути назвами таблиць, подання, елементів даних, об’єктів, модулів або інших категорій, залежно від моделі бази даних і системи управління, що використовується.

SUBJECTS	Students таблиця	StuView представлення	WrapUp процедура	Faculty таблиця	Schedule таблиця	Create Table команда
User FB01-01	READ, UPDATE	READ	EXECUTE	READ		NO
User FB01-02		READ				NO
Teacher role	READ	READ			READ, INSERT, UPDATE, DELETE	YES
- - - - -	- - - - -	- - - - -	- - - - -	- - - - -	- - - - -	- - - - -

Рис. 10.8. Матриця доступу

Стовпчик SUBJECTS містить суб'єктів доступу, тобто осіб, ролей, груп користувачів або застосунки. Записи у комірках визначають тип дозволеного доступу. Значення записів також залежать від конкретної використовуваної системи, але зазвичай вони включають операції READ, INSERT, UPDATE, DELETE, CREATE та інші. Після того, як матриця контролю доступу сформована, DBA повинен використати відповідну мову авторизації для її реалізації.

Мова SQL включає підтримку ролей починаючи з стандарту SQL-99 [7]. Ролі можна створювати і знищувати за допомогою команд CREATE ROLE і DROP ROLE. Користувачам можуть бути надані ролі (за бажанням, з можливістю передачі ролі іншим).

Стандартні команди GRANT і REVOKE можуть призначати привілеї ролям або обліковим записам (і відкликати їх).

Команда GRANT надає користувачам привілеї до базових таблиць і подання.

Синтаксис цієї команди наступний:

```
GRANT privileges_list ON object TO users [ WITH GRANT OPTION ]
```

Об'єкт – це або базова таблиця, подання або деякі інші типи об'єктів. Можна вказати декілька привілеїв, в тому числі SELECT, INSERT, UPDATE, DELETE.

Якщо користувач має привілей з опцією WITH THE GRANT OPTION, він може передати надані привілеї іншому користувачеві (з опцією надання або без неї) за допомогою команди GRANT.

Заборона привілеїв

Команда REVOKE, протилежна команді GRANT, і дозволяє відкликати привілеїв. Синтаксис команди наступний:

```
REVOKE [ GRANT OPTION FOR ] privileges_list ON object FROM users { RESTRICT |
CASCADE }
```

Наприклад

```
REVOKE UPDATE, INSERT ON Employees FROM Mgr125;
REVOKE SELECT ON Salary FROM User1, User2, User3;
```

За допомогою цієї команди можна відкликати як привілей, так і лише опцію надання привілею іншим користувачам (за допомогою опції GRANT OPTION FOR). Користувач, який надав привілей іншому користувачеві, може відкликати наданий привілей.

Коли користувач виконує команду REVOKE з ключовим словом CASCADE, це призводить до відкликання названих привілеїв всім користувачам, які в даний час мають ці привілеї, отримані виключно за допомогою команди GRANT. Якщо ці користувачі отримали привілеї з опцією WITH GRANT OPTION і передали їх іншим користувачам, то ці користувачі, у свою чергу, втрачають свої привілеї внаслідок виконання команди REVOKE, якщо тільки вони не отримали ці привілеї за допомогою додаткової команди GRANT.

Надання привілеїв з опцією WITH GRANT OPTION створює ланцюжки успадкування, що графічно можна представити в вигляді графа повноважень (рис. 10.9). Вплив серії команд GRANT можна описати в термінах графа повноважень, в якому вершинами є користувачі (обліковий запис), а ребра вказують, як передаються привілеї. Існує ребро від користувача User1 до користувача User2, якщо User1 виконав команду GRANT, яка надає привілеї User2; ребро позначене дескриптором команди GRANT.

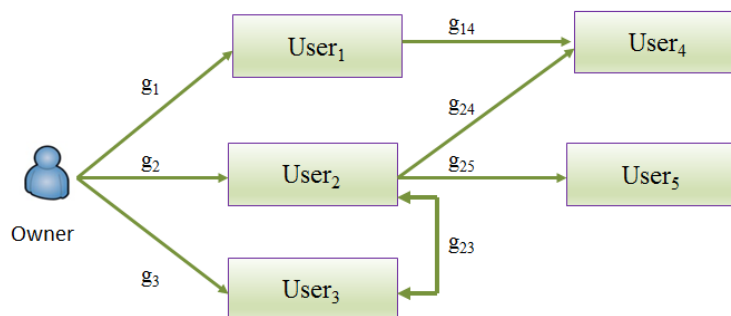


Рис. 10.9. Приклад ланцюжка успадкування

В прикладі на рис. 10.9 власник об'єкту надав привілеї з опцією *WITH GRANT OPTION* користувачам User1, User2, User3. Ті, в свою чергу, також надавали привілеї на цей об'єкт іншим користувачам. Однак, наявність багатьох таких «авторизаторів» може бути надзвичайно небезпечною. Оскільки авторизатори можуть створювати інших авторизаторів, ситуація може дуже швидко вийти з-під контролю, що ускладнить для адміністратора бази даних або власника об'єкта відкликання привілеїв.

Наприклад, в ситуації на рис. 10.9:

- якщо видалити привілеї з User1(g1) → User4 все ще має привілеї через g24
- якщо видалити привілеї з User2(g2) → User2 все ще має привілеї через g23
- якщо видалити привілеї з User3(g3) →User3 все ще має привілеї через g23
- якщо видалити привілеї з User2(g2) і User3(g3) →
- →g23 все ще існує, але більше не є частиною шляху, що починається з власника ресурсу
- →User2 і User3 більше не мають привілеїв → User5 більше не має привілеїв

Ці приклади підкреслюють складну взаємодію між командами GRANT і REVOKE.

Дозволи, надані ролі або групі, успадковуються їхніми членами. Хоча користувачеві може бути надано доступ через членство в одній ролі, членство в іншої ролі може мати заборону на дію з об'єктом. У такому разі виникає конфлікт доступу.

При вирішенні конфліктів доступу керуються таким принципом: дозвіл на надання доступу має найнижчий пріоритет, а на заборону доступу – найвищий. Це означає, що доступ до даних може бути отримано тільки явним його наданням за відсутності заборони доступу на будь-якому іншому рівні ієрархії системи безпеки.

10.3.3. Аудит

У контексті систем баз даних аудит – це процес моніторингу та запису вибраних подій і дій у базі даних. Аудит в першу чергу використовується для забезпечення підзвітності, перевірки політик безпеки, а також для фіксації та аналізу спостережуваної поведінки застосунків, користувачів та об'єктів бази даних. Аудит передбачає систематичний запис подій бази даних, таких як спроби входу, запити, модифікації даних та події, пов'язані з безпекою. Ці записи, які часто називають журналами аудиту, слугують кільком важливим цілям:

Моніторинг безпеки: Журнали аудиту уможливлюють моніторинг активності користувачів і системних подій у реальному часі, що дозволяє командам безпеки оперативно виявляти підозрілу або несанкціоновану поведінку.

Реагування на інциденти: У разі порушення безпеки або компрометації даних, журнали аудиту служать джерелом інформації для аналізу та реагування на інциденти.

NIST встановив шість компонентів аудиту безпеки, які можуть бути безпосередньо адаптовані до систем баз даних[12]. Це

(1) вибір подій аудиту безпеки,

(2) генерація даних аудиту безпеки,

(3) зберігання подій аудиту безпеки,

(4) огляд і аналіз подій аудиту безпеки,

(5) процедури автоматичне реагування.

Події безпеки, які необхідно реєструвати для всіх систем, включають, але не обмежуються ними:

1. Успішні та неуспішні події автентифікації, такі як вхід/вихід з системи, зміна пароля. Запис журналу аудиту має містити ідентифікатор користувача, дата і час події, тип події; вказівку на успіх або невдачу події. Також необхідно зберігати ідентифікація джерела події, наприклад, місцезнаходження, IP-адреса, ідентифікатор терміналу або інші засоби ідентифікації.
2. Неуспішні події доступу до ресурсів. Як мінімум, треба реєструвати ідентифікатор користувача, тип події, власно подію, дату і час події; ресурс і ідентифікацію джерела події.
3. Успішні та неуспішні привілейовані операції:

– використання системних привілейованих облікових записів;
– запуск і зупинка системи;

[12] NIST 800-92 Guide to Computer Security Log Management

– приєднання та від'єднання апаратного забезпечення;

– попередження та повідомлення про помилки управління системою та мережею; і

– події безпеки – управління обліковими записами/групами та зміни політик.

4. Успішний та неуспішний доступ до файлів журналів аудиту. Необхідно реєструвати ідентифікатор користувача, дату і час, тип події; вказівку на успіх або невдачу події; ідентифікацію джерела події.

Для систем, визначених на основі оцінки ризиків як критичні, або систем, які ще не були класифіковані, на додаток до вищезазначеного, необхідно реєструвати також події успішного доступу до ресурсів.

Більшість сучасних комерційних і відкритих СКБД надають механізми аудиту, які підтримують означені вимоги, і які здебільшого різняться за ступенем деталізації інформації про події, що може бути записана в журналі. Наприклад, тригери бази даних надають зручний засіб для запису інформації про оператори SQL – вставки, оновлення та видалення даних, включаючи запис старих і нових значень оновлених кортежів. Дії, пов'язані зі створенням, модифікацією та видаленням об'єктів бази даних, що виникають в результаті виконання операторів мови визначення даних SQL (DDL), також можуть бути предметом аудиту.

На додаток до тригерів бази даних і вбудованого аудиту, що підтримуються багатьма СКБД, ще однією корисною технікою є збережені процедури. Замість окремих операторів SQL вставки, оновлення та видалення, що виконуються з програми, викликаються збережені процедури. Збережена процедура виконує оператори модифікації даних, а також записує у допоміжних таблицях деяку додаткову інформацію, таку як роль користувача, інші користувачі, що виконують сеанси роботи з базою даних, або кількість записів, які були змінені за допомогою цих операторів.

Кожен метод реалізації аудиту має свої переваги та недоліки.

Аудит з використанням API. Можна налаштувати аудит за допомогою API, який фіксує необхідні значення, а потім зберігає дані аудиту. Перевага цього методу полягає в тому, що можна викликати API в ключових точках прикладної програми, фіксуючи дії користувачів. однак, якщо розробник не включив виклик API, дія не буде записана. Привілейовані користувачі можуть змінювати дані такого аудиту. Оскільки API – це код, його потрібно підтримувати.

Аудит з використанням збережених процедур. Реалізація аудиту за допомогою збережених процедур потребує істотної зміни в організації доступу до даних. Всі операції над даними необхідно реалізувати виключно через збережені процедури.

Аудит з використанням тригерів. Тригери також можна використовувати для налаштування детальної реєстрації для таблиці, записуючи всі зміни, час їх внесення та особу користувача, який їх здійснив. Перевага аудиту за допомогою тригерів полягає в можливості відокремити події аудиту від програми. Кожного разу, коли відбувається подія тригера, вона буде записана.

Недоліком аудиту за допомогою тригерного коду є те, що дані аудиту будуть зберігатися в різних таблицях. Тому, щоб отримати повну картину аудиту, потрібно буде

аналізувати кілька таблиць. Оскільки тригер – це код SQL, його потрібно підтримувати, і в ньому можуть виникати помилки.

Наприклад, в Oracle, якщо треба відстежувати зміни оцінок у таблиці «Enroll», необхідно спочатку створити таблицю для зберігання записів аудиту. Схема для цієї таблиці може бути такою:

```
EnrollAudit(dateandTimeOfUpdate, userId, oldStuId, oldClassNo,
oldGrade, newGrade)
```

Тригер повинен вставляти запис в таблицю EnrollAudit, коли користувач намагається оновити оцінку в таблиці Enroll.

```
CREATE OR REPLACE TRIGGER EnrollAuditTrail
BEFORE UPDATE OF grade ON Enroll
FOR EACH ROW
BEGIN
INSERT INTO EnrollAudit
VALUES(SYSDATE, USER, :OLD.stuId, :OLD.classNumber, :OLD.grade,
:NEW.grade);
END;
```

Функція `SYSDATE` повертає поточну дату і час, а функція `USER` – ідентифікатор користувача, що виконує оновлення таблиці Enroll.

Вбудований аудит. Системи управління базами даних надають механізми для конфігурування та ввімкнення аудиту. Ось приклад з використанням SQL Server. Цей SQL-скрипт створює аудит на рівні сервера з назвою «MyServerAudit» і вказує місце для зберігання журналів аудиту.

```
-- Enable Auditing
USE master;
GO
CREATE SERVER AUDIT MyServerAudit
TO FILE (
FILEPATH = 'C:\SQLServerAuditLogs\')
WITH (
QUEUE_DELAY = 1000,
ON_FAILURE = CONTINUE
);
```

У наступному прикладі вмикаємо аудит на рівні бази даних для конкретних дій, виконаних користувачем «dbo» над таблицею «MyTable».

```
-- Enable Auditing at the Database Level
USE MyDatabase;
GO
CREATE DATABASE AUDIT SPECIFICATION MyDBAuditSpec
FOR SERVER AUDIT MyServerAudit
ADD (SELECT, INSERT, UPDATE, DELETE ON dbo.MyTable BY dbo);
```

Ефективне управління журналами аудиту передбачає не лише можливість аудиту, але й регулярний перегляд та архівування журналів, щоб забезпечити їх доступ до них у разі потреби для аналізу чи дотримання нормативних вимог.

Розуміння принципів аудиту та ведення журналів має важливе значення для підтримки безпеки систем баз даних. Впроваджуючи надійні методи аудиту та регулярно переглядаючи журнали, можна покращити здатність виявляти та реагувати на загрози безпеці, захищати конфіденційні дані та відповідати нормативним вимогам.

10.3.4. Криптографічний захист даних

Щоб унеможливити доступ до файлів безпосередньо через операційну систему або викрадення файлів, дані можуть зберігатися в базі даних у зашифрованому вигляді. Коли авторизовані користувачі отримують доступ до інформації належним чином, СКБД витягує дані і розшифровує їх автоматично. Шифрування також слід використовувати, коли дані передаються на інші сайти, щоб зловмисники могли отримувати тільки зашифровані дані.

Шифрування не вирішує проблеми управління доступом. Однак воно підвищує захист конфіденційності даних навіть під час обходу зловмисником системи управління доступом. Наприклад, якщо сервер бази даних неправильно налаштований, а хакер отримує конфіденційні дані, то вкрадені дані можуть бути не придатними, якщо їх зашифровано.

Для криптографічного захисту потрібна система шифрування, яка складається з наступних компонентів:

- Алгоритм шифрування, який приймає на вході звичайний текст (відкритий текст), виконує над ним певні операції і видає на виході зашифрований текст (шифротекст)
- Ключ шифрування, який є частиною вхідних даних для алгоритму шифрування і вибирається з дуже великого набору можливих ключів
- Алгоритм розшифрування, який оперує зашифрованим текстом на вході і створює відкритий текст на виході
- Ключ дешифрування, який є частиною вхідних даних для алгоритму дешифрування і вибирається з дуже великого набору можливих ключів.

Шифрування з симетричним ключем – це форма шифрування, при якій ключ розшифрування збігається з ключем шифрування, а алгоритм розшифрування є оберненим до алгоритму шифрування. Однією з широко використовуваних схем шифрування з симетричним ключем був стандарт шифрування даних (DES), розроблений компанією IBM прийнятий в 1977 році. У схемі DES сам алгоритм є відкритим, а ключ – закритим.

DES не є дуже безпечною схемою, оскільки вона може бути зламана за розумний проміжок часу через короткі ключі. В результаті декількох відомих випадків злому ключів DES, в 1999 році NIST рекомендував більш безпечну версію, яка називається Triple DES або 3DES. Потрійний DES зараз широко використовується в комерційних системах. Система потрійного DES використовує три ключі і, по суті, виконує шифрування DES тричі, по одному разу з кожним ключем.

У 2001 році було розроблено вдосконалену схему шифрування під назвою Advanced Encryption Standard (AES). Він був прийнята в якості стандарту і широко використовується в комерційних системах. Він використовує симетричну схему, яка є більш складною, ніж схема DES, і підтримує три можливих розміри ключа: 128 біт, 192 біт або 256 біт, в залежності від необхідного рівня безпеки. Через більші розміри ключів зламати схему складніше.

Альтернативним підходом до шифрування є **шифрування з відкритим ключем**, яке також відоме як асиметричне шифрування. Шифрування з відкритим ключем використовує два окремі ключі, один з яких є відкритим, а інший - закритим. Відкриті ключі вільно поширюються, так що будь-хто, хто бажає надіслати повідомлення користувачеві, може легко знайти його відкритий ключ. Потім відкритий ключ використовується як вхідні дані для алгоритму шифрування, який створює зашифрований текст для цього користувача. Цей метод є настільки ж безпечним, наскільки безпечним є приватний ключ, тому користувачі повинні отримувати свої приватні ключі у певний безпечний спосіб і повинні захищати їх від крадіжки. Одним з відомих методів шифрування з відкритим ключем є RSA, названий на честь його розробників, Рівеста, Шаміра та Адлемана.

Ось кілька рекомендацій щодо шифрування:

- Ключі шифрування повинні мати довжину не менше 128 бітів. Чим довший ключ, тим надійнішим він вважається.
- До втрати ключа шифрування слід ставитися так само серйозно, як і до втрати даних, які він використовував для шифрування.
- Конфіденційні дані слід шифрувати, якщо вони підлягають постійному зберіганню. Які дані вважати конфіденційними – це рішення, яке повинні приймати власники даних, а не DBA. Загалом, однак, будь-які персональні дані, які можуть бути використані для крадіжки особистих даних, слід вважати конфіденційними.
- Усі дані, які не вважаються загальнодоступними, повинні бути зашифровані, якщо вони передаються в електронному вигляді через мережеві з’єднання, які не зашифровані в інший спосіб. Наприклад, якщо з бази даних експортується файл замовлення на купівлю торговому партнеру через FTP, цей файл має бути зашифрований. Немає жодних гарантій, що зловмисники не стежать за загальнодоступними мережами.
- Електронна пошта не вважається безпечною, тому будь-яка конфіденційна інформація, що надсилається електронною поштою, повинна бути зашифрованим вкладенням, а не в основному тексті електронного повідомлення.

Як приклад, розглянемо систему криптозахисту Microsoft SQL Server. Для збільшення надійності криптозахисту та зменшення навантаження на систему застосовується спеціальна ієрархія ключів.

1. Кожна база даних шифрується за допомогою спеціального ключа – Database Encryption Key.
2. Database Encryption Key шифрується сертифікатом, який створено в базі даних Master.
3. Сертифікат бази даних Master шифрується її головним ключем.
4. Головний ключ БД Master шифрується головним ключем служби Service Master Key.
5. Головний ключ служби SMK шифрується службою захисту даних операційної системи Data Protection API (DP API).

На кожному рівні дані нижчого рівня шифруються на основі комбінації сертифікатів, асиметричних ключів і симетричних ключів. Асиметричні ключі та симетричні ключі можна зберігати за межами SQL Server у модулі розширеного управління ключами (EKM).

На рис. 10.10 показано, що на кожному рівні ієрархії засобів шифрування шифруються дані нижчого рівня. Доступ до початку ієрархії, як правило, захищається паролем.

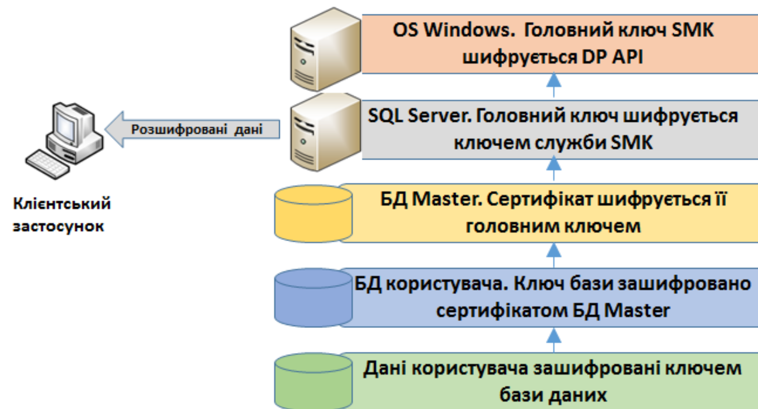


Рис. 10.10. Шифрування даних в MS SQL Server

Для шифрування змісту колонок таблиці БД в MS SQL Server необхідно виконати наступні дії:

1. Впевнитися, що для екземпляра SQL Server створено головний ключ. Він створюється при інсталяції екземпляра сервера.
2. Створити ключ бази даних.
3. Створити сертифікат. Сертифікат підписується SQL Server.
4. Змінити схему даних. Для зберігання зашифрованої інформації даних тип колонки має бути varbinary, тому в таблицю додається колонка такого типу з іменем наприклад, Credit_card_number_encrypted:
5. Шифрування колонки таблиці. Для шифрування використовується команда EncryptByKey. Перед шифруванням необхідно відкрити симетричний ключ, а по закінченню закрити.
6. Видалити незашифровану колонку:
7. Надання повноважень для доступу до зашифрованих даних. Для доступу до зашифрованих даних користувач повинний мати доступ до сертифіката і симетричного ключа.

Контрольні запитання

1. Які існують рівні конфіденційності даних?
2. Що таке вразливості і загрози безпеці? Наведіть приклади вразливостей безпеки в різних компонентах інформаційної системи.
3. Поясніть вказані визначення з точки зору забезпечення безпеки бази даних:
   а) авторизація

б) автентифікація

в) резервне копіювання та відновлення;

г) шифрування;

д) технологія RAID.

4. Описати проблеми безпеки, пов’язані з використанням СКБД в Інтернеті.
5. Що таке граф наслідування повноважень? Зокрема, обговоріть, що відбувається, коли користувачі передають привілеї, які вони отримали від когось іншого.
6. Поясніть, чому простого механізму паролів недостатньо для автентифікації користувачів, які отримують доступ до бази даних віддалено, наприклад, через Інтернет.
7. Що таке SQL-ін’єкція?
8. Як можна використовувати подання як складову безпеки даних?
9. Яке призначення SQL-команд GRANT та REVOKE? Перерахуйте деякі привілеї, які можуть бути надані користувачеві або відкликані у нього.
10. Поясніть, як контроль цілісності може бути використаний для захисту баз даних.
11. Скільки ключів знадобиться 50 користувачам бази даних, які вирішили використовувати симетричне шифрування? Якщо така ж кількість користувачів перейде на асиметричне шифрування, скільки ключів буде потрібно?
12. Сервер БД зберігає паролі у вигляді хешів. Якщо ви втратили свій пароль, як можна використати хеш для отримання відкритого пароля?
13. Чому адміністратори баз даних зазвичай блокують невикористовувані облікові записи?
14. Які існують рекомендації щодо створення та аналізу журналів аудиту?
15. У чому недолік вибору дуже стійкого алгоритму шифрування для захисту даних у базі даних?

Тестові завдання

1. Адміністратор вивчає журнали вебсервера і виявляє, що користувач відправив вхідні дані до вебзастосунку, які містять рядок `WAITFOR`. Який тип атаки, ймовірно, намагався здійснити користувач?
   A. Timing-based SQL injection
   B. HTML injection
   C. Cross-site scripting
   D. Content-based SQL injection
2. Ролі (кілька варіантів):
   A. Може містити будь-яку кількість привілеїв об’єкта
   B. Може містити лише один привілей об’єкта
   C. Може існувати раніше, ніж користувачі
   D. Може бути призначений тільки одному користувачеві
   E. Може бути спільним для багатьох користувачів
3. Адміністратор розробляє стратегію доступу до даних для своєї організації, яка складається з 150 осіб у 5 групах, з 3 ролями в кожній групі. Скільки ролей він повинен створити, щоб належним чином захистити дані?

   A. 3
   B. 5
   C. 15
   D. 150
4. Користувачу потрібно поділитися даними медичних знімків з партнером, який розробляє модель машинного навчання для виявлення хвороби. Що потрібно зробити, щоб забезпечити конфіденційність? Виберіть найкращу відповідь.
   A. Деідентифікувати дані
   B. Відсортувати дані у випадковому порядку
   C. Зашифрувати дані
   D. Агрегувати дані
5. Користувач має два набори знеособлених даних. Перший містить дані про медичне лікування, а другий – біографічну інформацію. Поштовий індекс, стать і дата народження присутні в обох наборах даних. Чи піддається ризику особиста конфіденційність?
   A. Ні, дані знеособлюються
   B. Ні, набори даних є окремими
   C. Так, повторна ідентифікація можлива за поштовим індексом, статтю та датою народження
   D. Так, повторна ідентифікація можлива, оскільки в одному з наборів даних є біографічна інформація
6. Яка мета атаки SQL ін'єкції?
   A. Покращення естетики сайту
   B. Впровадження шкідливих скриптів на вебсторінки
   C. Отримання несанкціонованого доступу до бази даних шляхом маніпулювання SQL-запитами
   D. Підвищення продуктивності сервера
7. Який з наведених нижче підходів, коли це можливо, є найефективніший спосіб протистояти ін'єкційним атакам?
   A. Перевірка вводу в браузері
   B. Список дозволених вхідних даних
   C. Заборона вхідних даних
   D. Перевірка підпису
8. Яка основна мета криптографії в безпеці баз даних?
   A. Запобігти несанкціонованому доступу до бази даних
   B. Виявлення вразливостей в базі даних
   C. Підвищити ефективність роботи бази даних
   D. Спростити адміністрування бази даних
9. Яке основне призначення журналів аудиту в безпеці баз даних?
   A. Для виявлення та запобігання SQL ін'єкційним атакам
   B. Для моніторингу активності бази даних та виявлення потенційних порушень безпеки
   C. Шифрування конфіденційних даних в базі даних

D. Резервне копіювання та відновлення бази даних у разі аварії

10. Як може статися витік даних через неправильні конфігурації у сховищах даних?
    A. Неправильні конфігурації не впливають на витік даних
    B. Неправильні конфігурації можуть призвести до ненавмисного доступу до конфіденційних даних неавторизованих користувачів
    C. Неправильні конфігурації впливають лише на продуктивність сховищ даних
    D. Витік даних спричинений виключно зловмисними діями
11. Який основний метод використовується для запобігання витоку резервних копій даних в системі безпеки баз даних?
    A. Шифрування
    B. Контроль доступу
    C. Стеганографія
    D. Захист від атак на відмову в обслуговуванні
12. Як адміністратор може забезпечити дотримання принципу «найменших привілеїв» для запобігання витоку даних?
    A. Принцип найменших привілеїв не впливає на запобігання розкриттю даних
    B. Забезпечуючи користувачам максимальний рівень доступу до даних
    C. Обмеження доступу користувачів до мінімального рівня, необхідного для виконання їхніх завдань, зменшує ризик витоку даних
    D. Принцип найменших привілеїв застосовується лише до фізичної безпеки
13. Який захід безпеки може допомогти захиститися від атак SQL-ін’єкцій?
    A. Перевірка вхідних даних
    B. Використання сесійних файлів cookie
    C. Міжсайтовий скриптинг (XSS)
    D. Шифрування збережених даних
14. У чому особливість атак типу Man-in-the-Middle (MitM) в контексті перехоплення даних?
    A. Вони вимагають валідації на стороні сервера
    B. Вони передбачають перехоплення комунікації неавторизованою третьою стороною
    C. Покращують взаємодію з користувачем
    D. Вони покладаються виключно на контроль на стороні клієнта
15. Яка з наведених нижче команд використовується для авторизації?
    A. `CREATE`
    B. `ALTER`
    C. `GRANT`
    D. `REVOKE`

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Coronel C., Morris S. Database Systems: Design, Implementation, and Management, 14th Edition, 2023, Boston, Cengage Learning, 816 p. ISBN 9780357673034
2. Silberschatz A., Korth H.F., Sudarshan S., Database system concepts, Seventh edition, 2020, New York, McGraw-Hill, 1344 p., ISBN 9780078022159
3. Lemahieu W., vanden Broucke S., Baesen B., Principles of database management : the practical guide to storing, managing and analyzing big and small data, 2018, Cambridge, Cambridge University Press, 808 p. ISBN: 9781107186125
4. Hoffer J. A., Ramesh V., Topi H., Modern Database Management, 13th edition, 2019, London, Pearson Education, 586 p. ISBN 9780134773650
5. Friedrichsen L., Ruffolo L., Monk E., Starks J., Pratt P., Last M., Concepts of Database Management, 10th Edition , 2021, Boston, Cengage Learning 418 p., ISBN: 9780357422083
6. Hernandez M.J., Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design 4th Edition, 2021, Boston, Pearson Education, 635 p., ISBN 9780136788041 (e-book)
7. Ricardo C., Urban S., Databases illuminated. Third edition, 2017 p. Burlington, Jones & Bartlett Learning, 743 p., ISBN 9781284056945,
8. Domdouzis K., Lake P., Crowther P., Concise Guide to Databases A Practical Introduction Second Edition, 2021, Cham, Springer, 400 p., ISBN 9783030422240 (eBook)
9. Shellman M., Afyouni H., Pratt P., Last M., A Guide to SQL, Tenth Edition, 2021, Boston, Cengage Learning, 338 p.. ISBN: 9780357361689
10. Hughes S., Neer D., Singh R., Mala S., Andrews L., Zhang C., SQL Query Design Patterns and Best Practices., 2023, Birmingham, Packt Publishing, 270 p., ISBN 9781837633289
11. Shan J., Goldwasse M. Malik U., 2022, SQL for Data Analytics. Birmingham, 540 p., Packt Publishing, ISBN: 978-1-80181-287-0
12. Patni J., A comprehensive study of SQL: practice and implementation. 2023, Boca Raton, CRC Press, 236 p. ISBN 9781003324690 (e-book)
13. Connolly T., Begg C., Database systems A Pratical Approach to Design, Implementation, and Management, Sixth Edition, 2015., Harlow, Pearson Education Limited, 1442 p., ISBN 9780132943260,
14. Sciore E., Database Design and Implementation, Second Edition, 2020., Springer Nature Switzerland 468 p., ISBN 978-3-030-33836-7
15. Molinaro A., de Graaf R., SQL Cookbook, 2021, Sebastopol, O'Reilly Media, 572 p., ISBN 978-1-492-07744-2
16. Kline K.,Obe R., Hsu L., SQL in a Nutshell, 2022, Sebastopol, O'Reilly Media, 1123 p., ISBN 9781492088868

ПОКАЖЧИКИ

BASE, 252
CASE-вираз з пошуком, 165
Cassandra, 260
CIA, 268
Common Table Expression *CTE*, 143
ETL, 237
Hadoop, 238
HDFS, 238
`INNER JOIN`, 134
JSON, 256
`LEFT OUTER JOIN`, 134
MapReduce, 240
MongoDB, 257
N - рівнева клієнт-серверна архітектура, 14
Neo4j, 262
NIST SP 800-63B, 279
NoSQL, 250
Null значення, 73
Riak, 254
`RIGHT OUTER JOIN`, 136
shared disk, 244
shared-nothing, 245
SQL ін'єкції, 269
`WITH GRANT OPTION`, 283
Автентифікація на рівні операційної системи, 277
Авторизація, 280
Аксіоми Армстронга, 97
Альтернативний ключ, 69
Аналітичні функції, 171
аномалії видалення, 93
аномалії включення, 93
аномалії модифікації, 93
Аномалії операцій з даними, 92
Арифметичні оператори, 116
Архітектура ANSI / SPARC, 24
Атомарний атрибут, 44
Атрибут, 40, 65
аудит, 284
Аудит з використанням API, 285
Аудит з використанням збережених процедур, 285
Аудит з використанням тригерів, 285
Багатозначна залежність, 108
Багатозначна функціональна залежність, 97
Багатозначні атрибути, 45
Багатотабличні запити, 133
База даних, 20
Бекенд, 16

Вбудований аудит, 286
Вбудовані механізми автентифікації, 277
Великі дані, 235
Вертикальне масштабування, 244
Взаємно-незалежні атрибути, 94
Вибірка, 81
Видалення таблиць, 123
Види привілеїв доступу в базах даних, 281
Визначення кандидатів в сутності, 42
Визначення ключових атрибутів, 48
Відношення, 64
Віконні функції, 170
Вкладені запити, 138
Властивості відношень, 67
Внесення змін в структуру таблиці, 123
Горизонтальна масштабованість, 236
Горизонтальне масштабування, 247
Графові сховища, 261
Дані, 10
Даталогічні моделі., 29
Декомпозиція без втрат, 100
Денормалізація, 109
Діаграми потоків даних, 35
Ділення відношень, 86
Ділення таблиць, 163
Документ, 257
Документальні моделі, 30
Домен, 62
Доступність, 268
Друга нормальна форма, 103
Е.Ф. Кодд, 92
Еквівалентні схеми, 66
Еквi-з'єднання, 84
журнал аудиту, 284
Замикання функціональних залежностей, 97
Захист від ін'єкцій, 272
Збережені процедури, 150
Зв'язок, 40
Зв'язок багато до багатьох, 55
Зв'язок один до багатьох, 54
Зв'язок один до одного, 52
Зовнішнє з'єднання, 85
Зовнішні ключі, 52, 71, 72
Зовнішня сутність, 37
Ідентифікація і автентифікація, 277
Ін'єкція другого порядку, 271
Інфологічні моделі, 29
Кардинальність, 65
Кардинальність зв'язку, 51
кінцева узгодженість, 253
Кластер HDFS, 239
ключ відношення, 68

Ключ-кандидат, 69
ключове слово CUBE, 169
Ключові і зарезервовані слова, 116
Команда DELETE, 157
Команда GRANT, 282
Команда INSERT, 155
Команда REVOKE, 282
Команда UPDATE, 156
Контроль доступу, 280
Конфіденційність, 268
Концептуальна модель, 20, 39
концептуальний рівень, 26
концепція «5V», 236
Коректна схема БД, 92
Корельовані підзапити в операторі WHERE, 142
Корельовані підзапити в списку виводу SELECT, 142
Кортеж, 65
ланцюжки успадкування, 283
Ліве зовнішнє з’єднання, 85
Логічна незалежність, 26
Логічні оператори, 117
Масштабованість, 244
Матеріалізовані подання, 149
Мінімальне покриття, 98
Можливі ключі відношення, 69
Надлишкова функціональна залежність, 96
Недоліки тригерів, 153
Нерекурсивні CTE, 144
Нормальна форма Бойса–Кодда, 107
Об’єднання відношень, 78
Об’єднання таблиць, 162
Облікові записи, 278
Обов’язковий атрибут, 44
Обчислюваний (похідний) атрибут, 45
Однорангова реплікація, 249
Оператор ALL, 129
Оператор ANY, 130
Оператор EXIST, 130
Оператор GROUP BY, 132
Оператор HAVING, 132
Оператори SQL, 116
Оператори обробки множин, 118
Оператори порівняння, 117
Оптимізація вкладених запитів, 143
Основні компоненти системи баз даних, 19
Основні функції подання, 148
паролі, 278
Первинний ключ, 69
Переваги використання баз даних, 18
Перетин відношень, 78
Перетин таблиць, 163
перша нормальна форма, 102

Підзапити в реченні FROM, 140
Підзапити в реченні HAVING, 141
Підзапити в реченні WHERE, 140
Підзапити в списку виводу SELECT, 140
Повна функціональна залежність, 95
Повне з'єднання, 137
Повне зовнішнє з'єднання, 86
Повний декартів добуток, 63
Подання, 147
подання що оновлюються, 149
Події безпеки, 284
Порівнянні атрибути, 65
Посилальна цілісність, 71
Потік даних у MapReduce, 241
Потоки даних, 36
Праве зовнішнє з'єднання, 85
Правила для атрибутів сутності, 44
Предметна, 20
Приклад процесу map- reduce, 242
Природне з'єднання, 83
Пріоритет операторів, 118
Пріоритет операцій реляційної алгебри, 87
Проекція, 81
Проміжні підсумки, 168
простий CASE, 164
Прості вкладені запити, 139
Процес, 36
Пряма SQL ін'єкція, 269
Ранжування даних, 166
Рекурсивний зв'язок, 56
Рекурсивні CTE, 144
Реляційна алгебра, 76
Реплікація, 248
Реплікація головний – підлеглий, 248
Різниця відношень, 79
Різниця таблиць, 163
розподіл даних за шардами, 245
Розподіл даних за шардами, 247
Розрахунок ковзного середнього, 172
Розширений декартів добуток, 80
Самоз'єднання, 137
Сильні і слабкі сутності, 42
Синтаксис мови SQL, 115
Система керування базами даних, 19
СКБД, 19
Складений атрибут, 44
Сліпа SQL ін'єкція, 270
спеціальні операції, 77
створення таблиці, 121
Ступінь, 65
Ступінь зв'язку, 51
Суперключ, 69

Сутність, 40
Схема, 40
Схема бази даних, 66
Схема відношення, 66
Сховища «ключ – значення», 253
Сховища документів, 256
Сховища на основі стовпців, 259
Сховище даних, 37
Теорема CAP, 251
теоретико-множинні операції, 77
Тета-з’єднання, 84
Типи даних в SQL, 119
Типи інформаційних систем, 16
Типи обмежень DDL, 121
Транзитивна функціональна залежність, 96
Третя нормальна форма, 105
Тривіальна функціональна залежність, 96
Тригери, 152
Трирівнева архітектура клієнт-сервер, 13
Умовне з’єднання, 82
Унарні оператори, 118
Уточнення списку атрибутів, 47
Фактографічні моделі, 29
Фізична незалежність, 27
Фізичний рівень, 25
Фізичні моделі, 30
Формат команди SELECT, 124
Форми мови SQL, 115
Фронтенд, 16
Функції MAX і MIN, 131
Функції SUM і AVG, 131
Функції зсуву, 171
функції зсуву LAG() і LEAD, 173
Функції ранжування, 171
Функціональна взаємозалежність, 95
функціональна модель, 35
функціональні залежності, 94
Функціональні категорії команд SQL, 115
Функція COUNT, 131
Функція DENSE_RANK(), 167
функція PIVOT(), 170
Функція RANK(), 167
функція ROLLUP(, 168
Функція ROW_NUMBER(), 167
Цілісність, 268
Четверта нормальна форма, 108
Шардінг, 245
Шифрування, 287
шифрування з відкритим ключем, 288
Шифрування з симетричним ключем, 287
Ядро СКБД, 22