МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОРГАНІЗАЦІЯ БАЗ ДАНИХ

КОНСПЕКТ ЛЕКЦІЙ

для здобувачів фахової передвищої освіти
освітньо-професійної програми «Комп'ютерна інженерія»
галузь знань 12 Інформаційні технології
спеціальності 123 Комп'ютерна інженерія
денної форми навчання

Луцьк 2022

УДК 004.65 (07)
О 64

Електронна копія друкованого видання передана для внесення в репозитарій ТФК ЛНТУ
Завідувач бібліотеки _____ В. КАЗМІРЧУК
(підпис)

Рекомендовано до видання навчально-методичною радою ТФК ЛНТУ, протокол № ___ від «___« _____ 2022 року.
Голова навчально-методичної ради ТФК ЛНТУ_____ С. БУСНЮК

Розглянуто і схвалено на засіданні випускової циклової комісії «Комп'ютерна інженерія» ТФК ЛНТУ,
протокол № ___ від «___» _____2022 року.

Голова ВЦК _____П. ВОВК
(підпис)

Укладач: _____ В. ЗАВІША, викладач ТФК ЛНТУ;
(підпис)

Рецензент: _____ Н. БАГНЮК, к.т.н., доцент кафедри комп'ютерної інженерії та кібербезпеки ЛНТУ.

О 64 Організація баз даних. Конспект лекцій для здобувачів фахової передвищої освіти освітньо-професійної програми «Комп'ютерна інженерія» галузь знань 12 Інформаційні технології спеціальності 123 Комп'ютерна інженерія денної форми навчання/ В. В. Завіша. Луцьк : ТФК ЛНТУ, 2022. 112 с.

Конспект лекцій містить змістовний матеріал з теоретичних основ організації баз даних і систем керування базами даних. Докладно на численних прикладах наведено відомості з проектування, створення та обслуговування баз даних, етапів їх проектування, організації ефективної структури зберігання даних, організації запитів до збережених даних, методів забезпечення цілісності даних, а також здобуття практичних навичок використання мови запитів SQL.

Видання призначене для здобувачів фахової передвищої освіти, що навчаються на спеціальностях галузі знань 12 Інформаційні технології.

ЗМІСТ

ВСТУП 5

1. ВВЕДЕННЯ. ЗАГАЛЬНІ ПОЛОЖЕННЯ 6

1.1 Загальні положення 6

1.2 Інформація та інформаційна система 7

1.3 Обробка інформації в ІС 11

2. АРХІТЕКТУРА БД. МОДЕЛІ ДАНИХ 13

2.1 Архітектура БД 13

2.2 Моделі даних 18

2.2 Логічна модель 25

3. РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ ТА ЇЇ ХАРАКТЕРИСТИКИ 29

3.1 Реляційна структура даних 29

3.2 Побудова бази даних 32

3.3 Реляційна база даних 33

3.4 Управління реляційними базами даними 35

3.5 Теорія нормалізації реляційної моделі даних 37

3.6 Недоліки проекту БД «Харчування» 40

4. РЕЛЯЦІЙНА АЛГЕБРА 43

4.1 Унарні операції 44

4.2 Операції з множинами (бінарні) 45

5. АПАРАТНІ ТА ПРОГРАМНІ СКЛАДОВІ ДЛЯ ПРОЄКТУВАННЯ БД 51

5.1 Апаратні складові БД 51

5.2 Програмні складові БД 56

6. МОВА ЗАПИТІВ SQL 60

6.1 Структура мови мови запитів SQL 60

6.2 Основні команди SQL 62

7. ОГЛЯД ТИПІВ СУЧАСНИХ БАЗ ДАНИХ 100

7.1 Найпростіші типи баз даних 101

7.2 Реляційні БД 103
7.3 NoSQL бази даних 104
7.4 Комбіновані типи 108
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 110

ВСТУП

Метою дисципліни «Організація баз даних» для студентів галузі знань 12 «Інформаційні технології» є формування компетентностей щодо формування системи теоретичних і практичних знань в області баз даних, вивчення концепції моделювання даних в інформаційних системах, організації реляційних, розподілених та об'єктно-орієнтованих баз даних та знань, етапів їх проектування, організації ефективної структури зберігання даних, організації запитів до збережених даних, методів забезпечення цілісності даних, а також здобуття практичних навичок використання мови запитів SQL в складі інформаційних системах.

В конспекті лекцій приділяється значна увага проектуванню та створенню баз даних, їх нормалізації. Для створення, модифікації та керування даними у реляційних базах даних вивчається універсальна мова структурованих запитів SQL.

Основними дисциплінарними результатами навчання, після завершення лекційного курсу «Організація баз даних», є:

– надбання базових теоретичних знань щодо проектування баз даних;

– вміння проводити аналіз предметної області, для якої розробляється база даних;

– розроблення інформаційної системи та бази даних засобами MS SQL Server та в архітектурі клієнт-сервер;

– отримання навичок проектування реляційну модель бази даних, а також вміння виконання трансакцій в БД;

– здобуття практичних навичок використання мови запитів SQL.

1. ВВЕДЕННЯ. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Загальні положення

Необхідність пошуку потрібної інформації у людини виникає повсякчас, незалежно від сфери її професійних інтересів: з якої платформи відправляється потяг на Хмельницький, як приготувати вареники з вишнями, яку будову має молекула води, скільки днів тривала Друга світова війна, чи справджується прикмета про чорну кішку, яка частота змінного електричного струму в побутовій електричній мережі, яке закінчення мають іменники третьої відміни в родовому відмінку однини та ін. Відповіді на частину з цих запитань людина може отримати зі своєї пам’яті, для отримання інших необхідно звернутися до інформаційної системи залізничного вокзалу, переглянути кулінарну книжку, довідник з хімії, фізики чи електротехніки, посібник з правопису тощо. Для полегшення пошуку потрібної інформації людство придумало багато засобів – універсальні енциклопедії та енциклопедії з предметних галузей, довідники й словники, довідкові бюро та інформаційні табло та ін.

Обсяги повідомлень, які накопичило людство, невпинно зростають. Так, при розкопках стародавнього міста шумерів Ур було знайдено понад 20 тисяч глиняних табличок з відомостями про звичаї давнього народу, його легенди та історичні події, що відбувалися понад 5 тисяч років тому. Знаменита Александрійська бібліотека, яка була заснована в Єгипт у III ст. до нової ери, за різними джерелами містила від 100 до 700 тисяч рукописів. Сьогоднішні бібліотеки вражають обсягами різноманітних даних. Найбільшою у світі вважається Британська бібліотека в Лондоні, яка нараховує понад 150 млн одиниць зберігання, а найбільша бібліотека нашої країни – Національна бібліотека України імені В.І. Вернадського в Києві нараховує понад 15 млн одиниць зберігання.

Учені запевняють, що зберігання великих обсягів даних виправдано тільки за умови, якщо пошук потрібних даних здійснюється швидко і подаються вони в доступній для розуміння формі. Ці умови забезпечують сучасні

технології зберігання даних. Основою цих технологій є комп’ютеризовані бази даних (БД).

База даних – це впорядкований за певними правилами набір взаємопов’язаних даних.

Перша в Україні комп’ютерна база даних була розроблена в ході робіт з проектування і експлуатації електронної обчислювальної машини «Київ» (1959 р.). ЕОМ була розроблена для обчислювального центру Академії наук УРСР Л.Н. Дашевським, К.Л. Ющенко, К.О. Шкарабарою, С.Б. Погребинським під науковим керівництвом Б.В. Гніденка та В.М. Глушкова.

1.2 Інформація та інформаційна система

Змістовна сторона поняття «інформація» дуже багатогранна й не має чітких семантичних меж. Однак завжди можна сказати, що можна з нею робити. Саме відповідь на це запитання найчастіше й цікавить як системних аналітиків і розроблювачів інформаційної системи (ІС), так і користувачів інформації (її основних споживачів).

З погляду як користувачів, так і розроблювачів ІС в інформації є одна важлива властивість – вона є одиницею даних, яка підлягає обробці. Звичайно інформація надходить споживачеві саме у вигляді даних: таблиць, графіків, малюнків, фільмів, усних повідомлень, які фіксують у собі інформацію певної структури й типу. Таким чином, дані виступають як засіб подання інформації у певній, фіксованій формі, придатній для обробки, зберігання й передачі. Хоча дуже часто терміни «інформація» й «дані» виступають як синоніми, варто пам'ятати про цю їхню істотну відмінність. Саме в даних інформація знаходить інтерпретацію у конкретній ІС.

При згадуванні про «форму» подання інформації варто сказати ще про одну, «людську» властивість інформації – її сприйняття різними категоріями людей. Дані можуть бути згруповані спільно у документ. Документ може мати або не мати певної внутрішньої структури. Дані можуть бути відображені на екрані дисплея комп'ютера. Документи можуть мати аудіо- або відеоформу.

Розробляючи ІС, ніколи не слід забувати, для кого вони (системи) створюються і хто буде їх використовувати. Форма подання інформації в ІС визначає також і категорії користувачів. ІС створюються для конкретних груп користувачів, тобто вони, як правило, проблемно- орієнтовані.

Інформація є даними, яким надається деякий зміст (інтерпретація) у конкретній ситуації у рамках деякої системи понять. Інформація подається за допомогою кодування даних і здобувається шляхом їхнього декодування та інтерпретації.

У цьому визначенні фіксується три основних перетворення інформації й даних у процесі їхньої обробки в ІС: інформація – дані, дані – дані, дані – інформація.

На рисунку 1.1 наведені дві сторони визначення поняття інформації: функціональна й представницька. Перша загалом визначає коло дій над інформацією, а друга – результат виконання цих дій.

Рисунок 1.1 – Зміст поняття «інформація»

Основною метою створення ІС є задоволення інформаційних потреб користувачів шляхом надання необхідної їм інформації на основі збережених даних. Потреба в інформації як такій не вичерпує поняття інформаційних

потреб. Звичайно в поняття інформаційних потреб включають певні вимоги до якості інформаційного обслуговування й поводження системи в цілому (продуктивність, актуальність і надійність даних, орієнтація на користувача та ін.).

Під інформаційною системою розуміється організаційна сукупність технічних засобів, технологічних процесів і кадрів, що реалізують функції збору, обробки, зберігання, пошуку, видачі й передачі інформації.

Необхідність підвищення продуктивності праці у сфері інформаційної діяльності приводить до того, що в якості зовнішніх засобів зберігання й швидкого доступу до інформації найчастіше використовуються засоби обчислювальної техніки (цифровий і аналоговий) на основі комп'ютерів. Підтримка різних рівнів прийняття рішень зі сторони ІС та ІТ наведено на таблиці 1.1.

Таблиця 1.1 - Підтримка різних рівнів прийняття рішень

Рівень прийняття рішень	Напрям прийняття рішень	Типове застосування ІС	Типові ІТ-рішення
Стратегічний	Стратегії, що підтримують довгострокові цілі організації	Аналіз ринку і збуту; планування розробки товарів; оцінка ефективності	Отримання даних (Data mining); управління знаннями
Тактичний	Стратегії, що підтримують короткострокові задачі та розподіл ресурсів	Аналіз бюджету; прогнозування фонду заплати; планування запасів; обслуговування користувачів	Сховище даних; аналіз даних; електроні таблиці
Оперативний	Підтримка повсякденної діяльності персоналу та виробництва	Виплата зарплати; виписка рахунків; закупи; бухгалтерський облік	Бази даних; обробка трансакцій; генератори застосувань

Сучасні ІС – складні комплекси апаратних і програмних засобів, технології й персоналу, які ще називають автоматизованими інформаційними системами (АІС).

Структурно ІС містять у собі апаратне (hardware), програмне (software), комунікаційне (netware), проміжне програмне (middleware), лінгвістичне й організаційно-технологічне забезпечення.

Апаратне забезпечення ІС містить у собі широкий набір засобів обчислювальної техніки, передачі даних, а також цілий ряд спеціальних технічних пристроїв (пристрій графічного відображення інформації, аудіо- і відео-пристрої, засобу мовного уведення й т.д.). Апаратне забезпечення є основою будь-якої ІС.

Комунікаційне (мережне) забезпечення містить у собі комплекс апаратних мережних комунікацій і програмних засобів підтримки комунікацій в ІС. Воно має істотне значення при створенні розподілених ІС й ІС на основі Інтернету.

Програмне забезпечення ІС забезпечує реалізацію функцій введення даних, їх розміщення на носіях, модифікації даних, доступ до даних, підтримку функціонування устаткування. Програмне забезпечення можна розділити на системне (яке завершує процес вибору апаратно-програмного рішення або платформи) і користувальне (яке застосовується для вирішення завдань задоволення потреб користувача у комп'ютерному середовищі).

Лінгвістичне забезпечення ІС призначене для вирішення завдань формалізації змісту повнотекстової і спеціальної інформації для створення пошукового образу даних (профілю). У класичному змісті звичайно воно включає процедури індексування текстів, їхню класифікацію і тематичну рубрикацію.

Найчастіше ІС, що містять складно-структуровану інформацію, містять у собі тезауруси термінів і понять. Сюди можна віднести й створення процесорів спеціалізованих формальних мов кінцевих користувачів, наприклад, мов для маніпулювання бухгалтерською інформацією і т.д. Найчастіше працям з розроблення лінгвістичного забезпечення не надається належного значення.

Подібні недогляди найчастіше призводять до неприйняття користувачами самої інформації. Це стосується в першу чергу вузько спеціалізованих ІС.

У міру зростання складності і масштабів ІС важливу роль починає відігравати організаційно-технологічне забезпечення, що з'єднує різнорідні компоненти (апаратури, програми й персонал) у єдину систему й забезпечує процедури її керування й функціонування. Недооцінка цієї складової ІС найчастіше призводить до зриву строків впровадження системи й виведення її на виробничі потужності.

1.3 Обробка інформації в ІС

Одним з головних питань розроблення програмного забезпечення ІС є питання про співвіднесення програм і даних, тому що вирішення цього питання, в остаточному підсумку, визначає вибір алгоритмів обробки інформації, апаратних засобів і технологічної платформи. Фундаментальним принципом у вирішенні питання про співвіднесення програм і даних є концепція незалежності прикладних програм від даних, і неважливо, яка обробка даних передбачається: централізована або розподілена. Суть цієї концепції полягає не стільки у відділенні програм від даних, скільки у розгляді їх як самостійних взаємодіючих об'єктів.

Однією з останніх модифікацій цього принципу є концепція незалежності прикладних програм від даних разом із процедурами їхньої обробки (об’єктно-орієнтований підхід у програмуванні), що дозволяє вирішити ряд питань обробки даних, пов'язаних з інтерпретацією семантичного змісту даних.

Формування концепції БД і створення на її основі методу баз даних для вирішення завдань обробки інформації відбулося у 1962 році. До середини 60-х років минулого століття основною концепцією побудови програмного забезпечення були концепція файлової системи і так званий позадачний метод. Наприкінці 80-х років минулого століття була запропонована концепція

об’єктно-орієнтованих баз даних й об’єктно-орієнтований підхід розроблення програм на основі обробки подій.

Проблеми, які необхідно було розв’язати:

- проблема контролю надмірності даних;
- проблема взаємозв'язку між даними та прикладними програмами, які писалися стандартними мовами, при цьому програма містила як опис даних, так і алгоритми маніпулювання даними, в результаті, будь-які зміни в організації даних приводили до необхідності зміни програми.

Вимоги, які необхідно було виконати:

- інтеграція даних, коли всі дані зберігаються централізовано, створюючи модель, що динамічно оновлюється.
- максимальна незалежність прикладних програм від даних чи забезпечення фізичної та логічної незалежності даних.

Виконання цих вимог привело до створення єдиного для всіх завдань блоку даних – бази даних та розробки однієї керуючої програми для маніпулювання даними фізично – системи управління базами даними. Ця система обробляє всі програмні звернення до БД. Саме СУБД забезпечує незалежність даних, зміна фізичної організації даних сприймається СУБД і не впливає на прикладну програму. З іншого боку, зміна логіки програми не потребує реорганізації та зміни механізму доступу до фізичних даних. Введення СУБД розділяє логічну структуру даних від фізичної структури даних.

2. АРХІТЕКТУРА БД. МОДЕЛІ ДАНИХ

2.1 Архітектура БД

Якщо говорити про використання обчислювальної техніки, то глобально можна виділити два основних напрямки.

Перший напрямок – чисельні розрахунки. Історично воно з'явилося раніше і сприяло розвитку методів чисельного рішення складних математичних задач, розвитку мов програмування, орієнтованих на рішення обчислювальних задач.

Другий напрямок – це зберігання і обробка даних. Метою будь-якої інформаційної системи є зберігання і обробка даних про будь-які об'єкти реального світу.

Враховуючи, що інформація являє собою відомості про оточуючих людини предмети, явища і процеси і є об'єктом таких операцій як сприйняття, передача, перетворення, зберігання і використання. Коли використовується термін «дані», то мова йде про інформацію, представленої в формалізованому вигляді, придатної для автоматичної обробки при можливу участь людини.

У широкому сенсі слова термін «база даних» (БД) – це сукупність відомостей про конкретні об'єкти.

При створенні БД в основному мають на меті впорядкувати дані за різними ознаками, щоб мати можливість брати з даних потрібну інформацію. Створення БД, її підтримка, управління, а також доступ користувачів до самих даних здійснюється за допомогою спеціальних програмних продуктів, які називаються системами управління базами даних (СУБД).

Отже, системою керування базами даних називається сукупність програмних засобів, необхідних для використання БД і надання розробникам та користувачам доступу до великої кількості різних видів даних.

Основна особливість СУБД – це наявність процедур для введення і збереження не тільки самих даних, але і описів їх структури.

Файли, забезпечені описом збережених у них даних і знаходяться під управлінням СУБД, стали називати базами даних (БД).

Функції СУБД:

– управління буферами оперативної пам'яті;
– управління трансакціями;
– захист від відмов і відновлення (журналізація);
– забезпечення різних рівнів доступу до даних.

Розглянемо термін «архітектура». Він може мати різні тлумачення. Наприклад, коли вказуються функціональні модулі системи й способи їхньої взаємодії, йдеться про функціональну архітектуру. Спосіб реалізації функцій системи, її компоненти та взаємозв'язки між ними фіксує архітектура реалізації системи. У цьому контексті можна також згадати архітектуру технічних засобів систем. Говорячи ж про термін «архітектура БД», ми матимемо на увазі архітектуру інформаційного забезпечення.

Архітектура БД уперше була специфікована дослідницькою групою ANSI/X3/ SPARC Study Group on Data Base Management Systems (ANSI – Американський національний інститут стандартів, X3 – його комітет обчислювальної техніки й обробки інформації, SPARC – підкомітет ANSI/X3 з планування стандартів ANSI). Метою дослідницької групи було визначення галузей і технологій баз даних, в яких було б доречно проводити стандартизацію, а також вироблення рекомендацій для роботи в кожній із таких галузей. Група дійшла висновку, що, можливо, єдиним аспектом систем баз даних, який можна стандартизувати, є інтерфейси. Нею було докладено чимало зусиль для визначення загальної архітектури системи баз даних. Запропонована цією групою архітектура стала класичною та є актуальною й донині.

Класифікація баз даних:

а) за формою подання інформації

– фактографічні – зберігають безліч відомостей та фактів, що зберігаються в інформаційній системі, що задовольняють фіксовану сукупність форматів.

– документальні – зберігають сукупність довільних текстових документів;
– мультимедійні.

б) по типу використання моделі даних:
– ієрархічні,
– мережеві,
– реляційні,
– постреляційні,
– об'єктно-орієнтовані;
– багатовимірні.

в) за топологією обробки даних:
– централізовані;
– розподілені.

Централізована БД має на увазі, що робота з БД можлива тільки локально. Якщо комп'ютер працює в мережі, то доступ до інформації може здійснюватися віддалено з інших комп'ютерів мережі. Централізовані БД найбільш поширені в даний час. При цьому можливі кілька варіантів обробки даних.

Файл-серверна архітектура передбачає наявність в мережі сервера, на якому зберігаються файли централізованої БД. Відповідно до запитів користувачів файли з файл-сервера передаються на робочі станції користувачів, де і здійснюється основна частина обробки даних. Центральний сервер виконує в основному тільки роль сховища файлів, не беручи участь в обробці самих даних. Після завершення роботи користувачі копіюють файли з обробленими даними назад на сервер, звідки їх можуть взяти і обробити інші користувачі. Недоліки такої організації даних очевидні. При одночасному зверненні великої кількості користувачів до одних і тих же даних продуктивність роботи різко падає, тому що необхідно дочекатися поки користувач, що працює з даними завершить роботу. В іншому випадку можливе затирання виправлень зроблених одним користувачем, змінами інших користувачів.

В основі концепції клієнт-сервер лежить ідея про те, що крім зберігання файлів БД, центральний сервер повинен виконувати основну частину обробки даних. Користувачі звертаються до сервера за допомогою спеціальної мови структурованих запитів (SQL, Structed Query Language), на котрому описується список завдань, які виконуються сервером. Запити приймаються сервером і породжують процеси обробки даних. У відповідь користувач отримує вже відпрацьований набір даних. Технологія клієнт-сервер дозволяє уникнути передачі по мережі величезних обсягів інформації, переклавши всю обробку на центральний сервер. Такий підхід також дозволяє уникнути конфліктів при редагуванні одних і тих же даних великою кількістю користувачів.

Трирівнева архітектура («Тонкий клієнт» – сервер додатків – сервер бази даних) функціонує в Інтернет-мережах.

Клієнтська частина ("тонкий клієнт"), що взаємодіє з користувачем, являє собою HTML-сторінку в Web-браузері або Windows-додаток, що взаємодіє з Web сервісами. Вся програмна логіка винесена на сервер додатків, який забезпечує формування запитів до бази даних, що передаються на виконання сервера баз даних. Сервер додатків може бути Web-сервером або спеціалізованою програмою (див. Рис. 2.1).

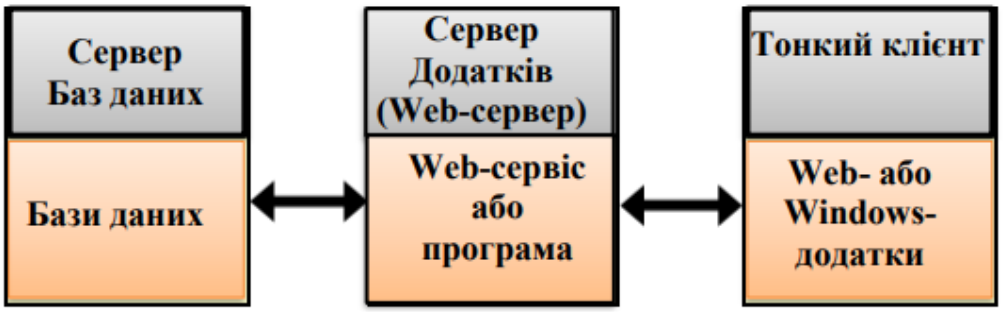


Рисунок 2.1 – Схема роботи з БД в трирівневої архітектурі

Розподілена БД розташовується на кількох комп'ютерах. Інформація на цих комп'ютерах може перетинатися і навіть дублюватися. Для управління такими БД призначена система управління розподіленими БД. Система приховує від користувачів доступ до даних, розташованим на інших

комп'ютерах. Для користувача все виглядає так, як ніби вся інформація знаходиться на одному сервері.

Основні функції СУБД:

- Безпосереднє керування даними у зовнішній пам'яті. У деяких реалізаціях СУБД активно використовуються можливості існуючих файлових систем, в інших робота виконується аж до рівня пристроїв зовнішньої пам'яті.

- Управління буферами оперативної пам’яті. Для збільшення швидкості обміну інформацією з оперативною пам’яттю У розвинених СУБД підтримується власний набір буферів оперативної пам'яті з дисципліною заміни буферів.

- Управління трансакціями. Транзакція – це послідовність операцій над БД, що розглядаються СУБД як єдине ціле.

- Серіалізація трансакцій – порядок планування їх роботи, при якому сумарний ефект суміші трансакцій еквівалентний ефекту їх деякого послідовного виконання.

Серіальний план виконання суміші трансакцій – це такий план, що призводить до серіалізації трансакцій. Якщо вдається досягти дійсно серіального виконання суміші трансакцій, то для кожного користувача, з ініціативи якого утворена трансакція, присутність інших трансакцій буде непомітною.

- Журналізація. Журнал – це особлива частина БД, що недоступна користувачам СУБД і підтримується з особливою ретельністю (іноді підтримуються дві копії журналу, що розташовуються на різних фізичних дисках), до якої надходять записи про всі зміни основної частини БД.

- Підтримка мов БД. Внутрішня мова СУБД складається з двох частин: мови визначення даних (DDL) та мови маніпулювання даними (DML). Теоретично для кожної схеми в трирівневій архітектурі можна було б виділити кілька різних мов DDL, а саме мову зовнішніх схем DDL, мову DDL концептуальної схеми і мову DDL внутрішньої схеми. Однак на практиці існує

одна спільна мова DDL, яка дозволяє задавати специфікації як мінімум для зовнішньої та концептуальної схем.

Розглянемо типову організацію сучасної СУБД. Сучасна СУБД містить такі компоненти (рис. 2.2):

- ядро, яке відповідає за керування даними у зовнішній та оперативній пам'яті та введення журналу,
- процесор мови бази даних, що забезпечує оптимізацію запитів на вилучення та зміну даних та створення, як правило, машинно-незалежного виконуваного внутрішнього коду;
- підсистему підтримки часу виконання, яка інтерпретує програми маніпуляції даними, що створюють інтерфейс користувача з СУБД;
- сервісні програми (зовнішні утиліти), що забезпечують ряд додаткових можливостей з обслуговування інформаційної системи.

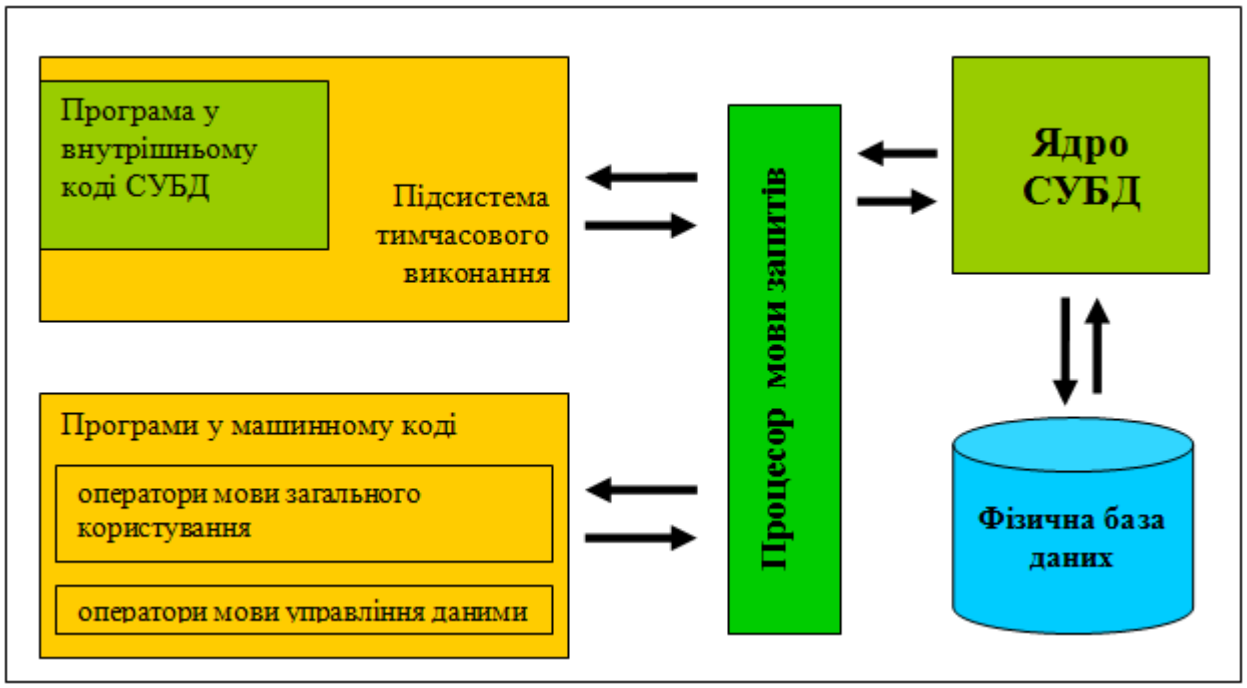


Рисунок 2.2 – Типова організація сучасної СУБД

2.2 Моделі даних

При вирішенні конкретних проблем зазвичай обмежуються тієї його частиною, яка є областю цієї діяльності. У цих випадках інтерес становлять лише деякі його об'єкти. Сукупність таких об'єктів називається предметною

областю, а об'єкти – об'єктами предметної області. Об'єктами можуть бути: люди, наприклад, перелічені в будь-якій платіжній відомості; предмети, наприклад деталі, що виробляються; побудови – уявні об'єкти, наприклад рахунки, завдання отримання рахунків.

Під терміном модель даних можна розуміти концептуальний (що відповідає деякій концепції, деяким правилам) опис предметної області. Вона включає визначення сутностей (все те, про що можна зберігати інформацію) та їх атрибутів: наприклад, сутність (Покупець) може мати атрибути Name (Ім'я) і Address (Адреса).

Предметна область має складну структуру і невпорядкована – і це природно, адже якби вона була простою і впорядкованою, то не було б потреби в її моделі. Тобто, предметна область це частина реального світу, що відображається в базі даних (див. таблиця 2.1).

Таблиця 2.1 - Опис предметної області

Предметна область	Фрагменти	Об’єкти	Процеси
Коледж	Бухгалтерія	Розрахункова відомість	Порахувати зарплату
		Прибутковий ордер	Прийняти гроші в касу
		Головна книга	
	Відділ кадрів		
	Бібліотека		

Кожен об'єкт має набір властивостей чи атрибутів, виконує операції, реалізовані методами, і реагує на певні події:

Об'єкт = {властивості, методи, події}

Властивості можна класифікувати за наступними ознаками:

а) за часом:

- статичні (що не змінюються у часі);
- динамічні (які змінюються у часі);

б) за структурою:

- неподільні (атомарні);
- складові (агрегат, вектор, група, що повторюється);

в) за кількістю:

- множинні,
- одиничні;

г) за наявності:

- обов'язкові,
- умовні.

д) за ознакою похідності:

- базові;
- розрахункові чи похідні.

До структурних елементів даних відносять:

- байт;
- поле;
- агрегат даних (група): вектор, група, що повторюється.

До ідентифікації записів входять:

- Потенційний ключ – претендент на роль первинного ключа .

- Первинний ключ (primary key – PK) – атрибут або група атрибутів, що однозначно ідентифікують екземпляр сутності:

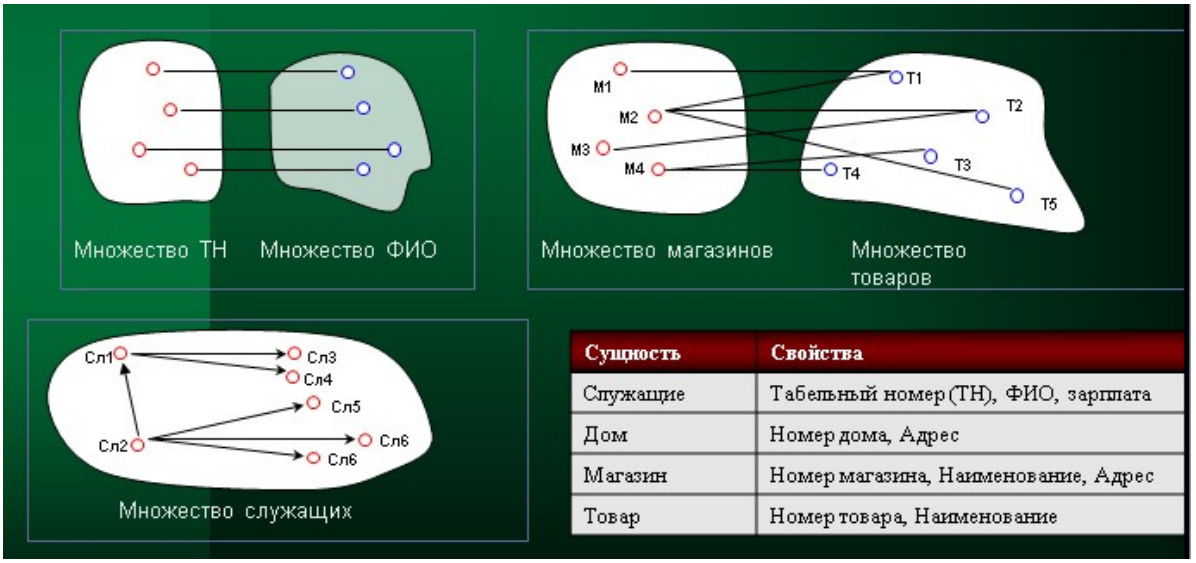
- сурогатний (штучний) ключ;
- складений (складний) ключ

- Альтернативний ключ – потенційний ключ, що не став ключовим (foreign key – FK)

- Інверсний вхід (вторинний ключ) – атрибут або група атрибутів, які не визначають екземпляр сутності унікальним чином, але часто використовуються для звернення до екземплярів сутності.

Об’єкти пов’язані між собою через зв'язки (рис.2.3):

- між атрибутами;
- між сутностями;
- рекурсивні, між елементами.



Сущность	Свойства
Служащие	Табельный номер (ТН), ФИО, зарплата
Дом	Номер дома, Адрес
Магазин	Номер магазина, Наименование, Адрес
Товар	Номер товара, Наименование

Рисунок 2.3 – Типи зв’язків.

До характеристик зв'язку відносять:

а) Клас належності (ступінь участі):

- обов'язковий
- необов'язковий

б) Розмірність зв'язку (ступінь зв'язку):

- бінарна
- потрійна
- N-арна.

в) Потужність зв'язку (кардинальність, тип відповідності):

- T(X,Y) = 1:1
- T(X,Y) = 1:M
- T (X,Y) = M:1
- T(X,Y) = M:M
- T(X,Y) = Ø.

Приклади зв'язку:

- ТОВАР-МАГАЗИН бінарний зв'язок, тип відповідності M:M.
- ТОВАР-МАГАЗИН-ПОКУПЕЦЬ потрійний зв'язок.

- СЛУЖБОВЕЦЬ-ПОСАДА бінарний зв'язок, обов'язковий зв'язок з боку СЛУЖБОВЕЦЬ і необов'язковий з боку ПОСАДА (посада може бути вакантною).

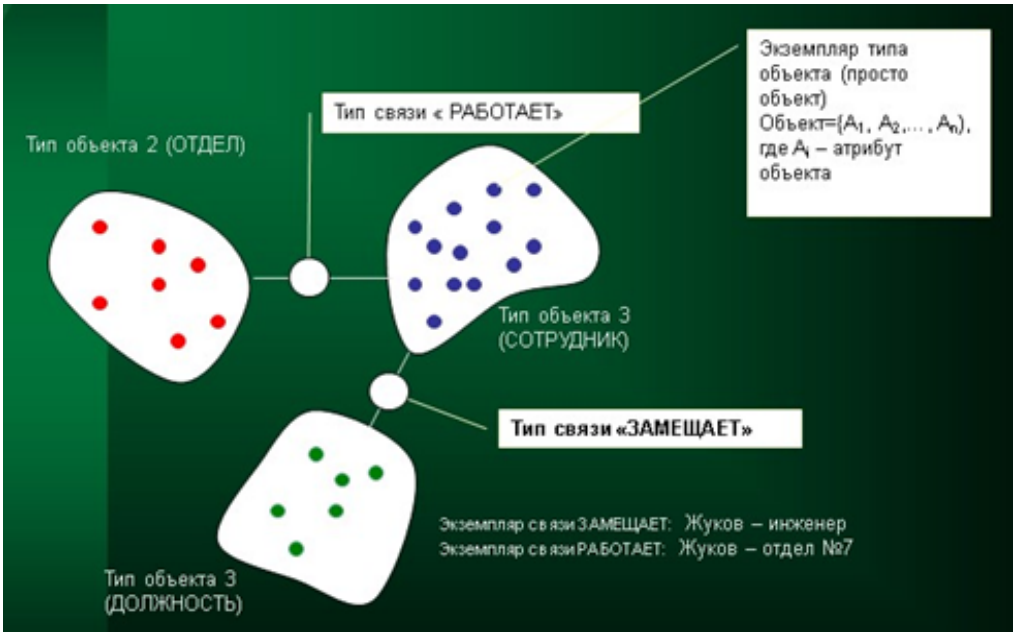


Рисунок 2.4 – Типи, екземпляри об'єктів та зв'язки

Рисунок 2.5 – Класифікація моделей даних

Згідно класифікації виділяють наступні моделі даних (рис. 2.5):

– семантичні;

– логічні;

– фізичні.

2.1.1. Семантичні мережі

Семантична мережа (СМ) – це граф, дуги якого є відносини між вершинами (значеннями). Семантичні мережі з'явилися при вирішенні завдань розбору і розуміння сенсу природної мови. Приклад семантичної мережі для пропозиції типу «Постачальник здійснив поставку виробів на замовлення клієнта до 1 червня 2021 року в кількості 1000 штук» наведено на рис. 2.6.

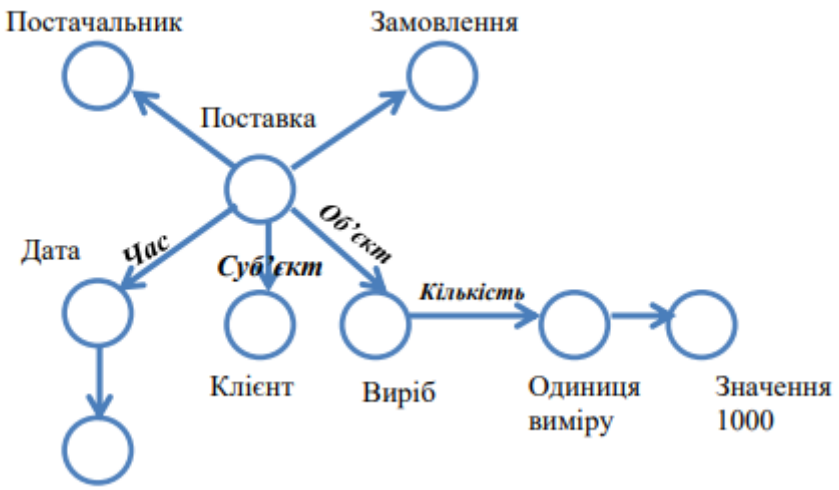


Рисунок 2.6 – Приклад семантичної мережі

На цьому прикладі видно, що між об'єктами *Постачальник* і *Поставка* визначено ставлення «*агент*», між об'єктами *Виріб* і *Поставка* визначено ставлення «*об'єкт*» і т.д.

Число відносин, використовуваних в конкретних семантичних мережах, може бути саме різне. Неповний список можливих відносин, що використовуються в семантичних мережах для розбору пропозицій, виглядає наступним чином.

Агент – це те, що (той, хто) викликає дію.

Об'єкт – це те, на що (на кого) спрямована дія. У реченні об'єкт часто виконує роль прямого доповнення, наприклад, «Роббі взяв жовту піраміду».

Інструмент – то засіб, який використовується агентом для виконання дії, наприклад, «Роббі відкрив двері за допомогою ключа».

Напівагент служить як підлеглий партнер головному агенту, наприклад, «Роббі зібрав кубики з допомогою Суззі».

Пункт відправлення і *пункт призначення* – це відправна і кінцева позиції при переміщенні агента або об'єкта.

Траєкторія – переміщення від пункту відправлення до пункту призначення: «Вони пройшли через двері по сходах на сходи».

Засіб доставки – то в чому або на чому відбувається переміщення: «Він завжди їде додому на метро».

Місцезнаходження – то місце, де відбулося (відбувається, буде відбуватися) дію, наприклад, «Він працював за столом».

Споживач – то особа, для якого виконується дія: «Роббі зібрав кубики для Суззі».

Сировина – це, як правило, матеріал, з якого щось зроблено або складається. Зазвичай сировину вводиться приводом з, наприклад, «Роббі зібрав Суззі з інтегральних схем».

Час – вказує на момент вчинення дії: «Він закінчив свою роботу пізно ввечері».

Найбільш типовий спосіб виведення в семантичних мережах (СС) – це спосіб зіставлення частин мережевої структури. Це видно на наступному простому прикладі, представленому на рис. 2.7.

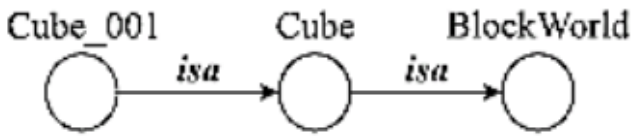


Рисунок 2.7 – Процедура співставлення СМ

Куб Cube належить світу BlockWorld.

Куб Cube_001 є різновид куба Cube.

Легко зробити висновок: Куб Cube_001 є частина світу BlockWorld.

2.2 Логічна модель

Логічні моделі умовно можна розбити на дві групи: документовані та фактичні.

2.2.1. Ієрархічна модель

Спочатку стали використовувати ієрархічні логічні моделі. Ця модель являє собою сукупність пов'язаних елементів, що утворюють ієрархічну структуру. До основних понять ієрархії відносяться рівень, вузол і зв'язок. Вузлом називається сукупність атрибутів даних, що описують деякий об'єкт. Кожен вузол пов'язаний з одним вузлом вищого рівня і з будь-якою кількістю вузлів нижнього рівня. Винятком є вузол найвищого рівня, який не пов'язаний ні з одним вузлом вищого рівня.

Кількість дерев в БД визначається кількістю коренів дерев. До кожного запису БД існує єдиний шлях від кореневого запису.

Прикладом ієрархічної моделі даних може служити адреса. На першому рівні (корені дерева) лежить наша планета – Земля. На другому – країна. На третьому – регіон (республіка, край, район), потім – населений пункт, вулиця, будинок, квартира . Ще один приклад – це система доменних імен в Інтернеті.

Але існують обмеження на використання даної моделі::

- неможливість зберігання екземплярів записів, які не мають батьківських записів;

- складність моделювання зв'язків типу M:M чи інших складніших неієрархічних зв'язків, у яких тип запису має кілька різних зв'язків із кількома іншими типами записів.

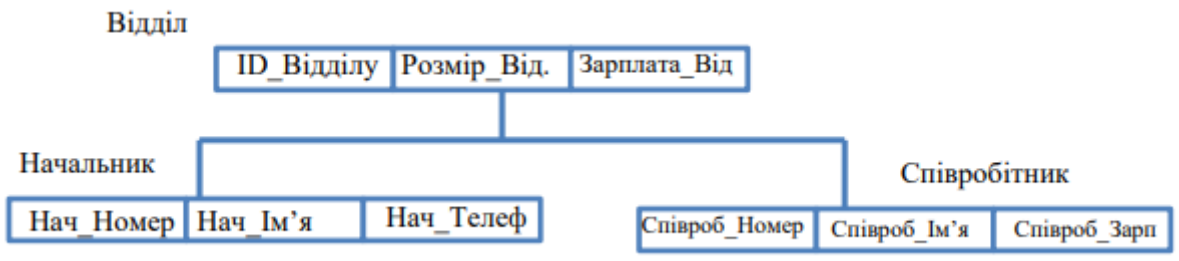


Рисунок 2.4 – Приклад ієрархічної структури

На рис 2.4 наведений приклад ієрархічної структури, де *Відділ* є предком для *Начальника* і *Співробітників*, а *Начальник* і *Співробітники* – нащадки *Відділ*. Між типами записи підтримуються зв'язки.

Типовим представником СУБД (найбільш відомим і поширеним), заснованої на ієрархічній моделі, є Information Management System (IMS) фірми IBM. Перша версія з'явилася в 1968 р.

2.2.2. Мережева модель

В основі мережевої моделі даних лежать ті ж поняття, що і в основі ієрархічної моделі – вузол, рівень і зв'язок. Однак істотною відмінністю є те, що в ієрархічних структурах запис-нащадок повинен мати в точності одного предка; в мережевій структурі даних нащадок може мати будь-яке число предків.

Мережевий підхід до організації даних є розширенням ієрархічного. У мережній моделі даних будь-який об'єкт може бути одночасно і головним, і підлеглим, і може брати участь в утворенні будь-якого числа взаємозв'язків з іншими об'єктами. Мережева БД складається з набору записів і набору зв'язків між цими записами, а якщо говорити більш точно – з набору примірників кожного типу із заданого в схемі БД набору типів запису і набору примірників кожного типу із заданого набору типів зв'язку (див. рис. 2.5).

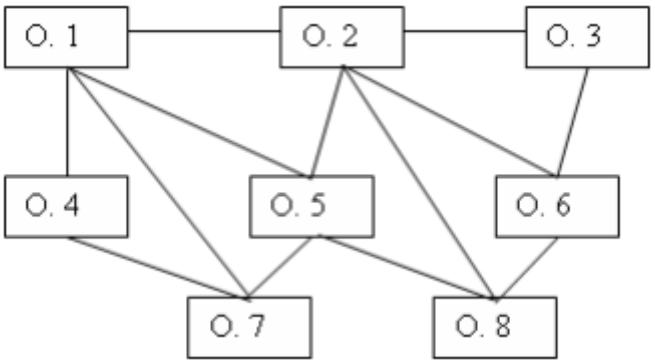


Рисунок 2.5 – Схема мережевої архітектури

Мережеві моделі створювалися для малоресурсних ЕОМ. Це досить складні структури, що складаються з «наборів» – пойменованих дворівневих дерев. «Набори» з'єднуються за допомогою «записів-зв'язок», утворюючи ланцюжки і т.д. При розробці мережевих моделей було вигадано множина «маленьких хитрощів», що дозволяють збільшити продуктивність СУБД, але істотно ускладнити останні. Прикладний програміст повинен знати масу термінів, вивчити декілька внутрішніх мов СУБД, детально представляти логічну структуру бази даних для здійснення навігації серед різних примірників, наборів, записів і т.п.

Типовим представником є Integrated Database Management System (IDMS) компанії Cullinet Software, Inc., призначена для використання на машинах основного класу фірми IBM під управлінням більшості операційних систем. Архітектура системи заснована на пропозиціях Data Base Task Group (DBTG), Комітету з мов програмування Conference on Data Systems Languages (CODASYL), організації, відповідальної за визначення мови програмування Кобол. Звіт DBTG був опублікований в 1971 р, а в 70-х роках з'явилося кілька систем, серед яких IDMS.

Складність практичного використання ієрархічних та мережевих БД змушувала шукати інші способи представлення даних. В кінці 60-х років з'явилися СУБД на основі інвертованих файлів, що відрізняються простотою організації і наявністю досить зручних мов маніпулювання даними.

До числа найбільш відомих і типових представників таких систем відносяться Datacom / DB компанії Applied Data Research, Inc. (ADR), орієнтована на використання на машинах основного класу фірми IBM, і Adabas компанії Software AG.

Організація доступу до даних на основі інвертованих списків використовується практично у всіх сучасних реляційних БД, але в цих системах користувачі не мають безпосереднього доступу до інвертованих списків (індексів).

База даних, організована за допомогою інвертованих списків, схожа на реляційну БД, але з тією відмінністю, що збережені таблиці і шляхи доступу до них видно користувачам. При цьому:

1. Рядки таблиць упорядковані системою в деякій фізичній послідовності.

2. Фізична упорядкованість рядків всіх таблиць може визначатися і для всієї БД (так робиться, наприклад, в Datacom / DB).

3. Для кожної таблиці можна визначити довільне число ключів пошуку, для яких будуються індекси. Ці індекси автоматично підтримуються системою, але явно видно користувачам.

Загальні правила визначення цілісності БД відсутні. У деяких системах підтримуються обмеження унікальності значень деяких полів, але в основному все покладається на прикладну програму.

Однак такі СУБД мають ряд обмежень на кількість файлів для зберігання даних, кількість зв'язків між ними, довжину запису і кількість її полів.

3. РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ ТА ЇЇ ХАРАКТЕРИСТИКИ

3.1 Реляційна структура даних

Концепція реляційної моделі даних пов'язані з ім'ям відомого фахівця у сфері баз даних Э. Кодда. Будучи математиком за освітою, Е.Кодд запропонував використовувати для обробки даних апарат теорії множин (об'єднання, перетин, різницю, декартовий добуток). Він показав, що будь-яке подання даних зводиться до сукупності двовимірних таблиць особливого виду, відомого в математиці як відношення – relation.

Реляційна БД – множина відносин, що містять всю необхідну інформацію про предметну область. Реляційною вважається така база даних, де всі дані представлені для користувача у вигляді прямокутних таблиць значень даних, і всі операції над базою даних зводяться до маніпуляцій з таблицями.

Найменша одиниця даних реляційної моделі – це окреме атомарне (нерозкладене) значення даних для даної моделі. Так, в одній предметної області прізвище, ім'я та по батькові можуть розглядатися як єдине значення, а в іншій – як три різних значення.

Доменом називається множина атомарних значень одного і того ж типу.

Так, в табл. 3.1 домен пунктів відправлення (призначення) – множина назв населених пунктів, а домен номерів рейсу - множина цілих позитивних чисел.

Таблиця 3.1 – Таблиця даних «Рейс»

Номер рейсу	Дні тижня	Пункт відправлення	Час вильоту	Пункт призначення	Час прибуття	Тип літака	Вартість квитка
138	2_4_7	Дніпро	21.15	Рига	0.55	МАУ-86	1110
57	3_6	Єреван	7.20	Київ	9.25	ТУ-154	920
1234	2_6	Одеса	21.00	Баку	23.50	ТУ-134	1300
242	1 по 7	Варшава	14.10	Київ	16.15	ТУ-154	1010
86	2_3_5	Львів	10.50	Вроцлав	13.10	МАУ-86	1600
137	1_3_6	Рига	7.15	Дніпро	10.45	МАУ-42	1110
241	1 по 7	Київ	09.10	Варшава	11.15	ТУ-154	1010
577	1_3_5	Рига	21.50	Таллін	22.55	ЯК-40	600
78	3_6	Львів	18.25	Київ	20.45	ТУ-134	900
578	2_4_6	Таллін	6.30	Рига	7.40	АН-24	600

Сенс доменів полягає в наступному. Якщо значення двох атрибутів беруться з одного і того ж домена, то, ймовірно, мають сенс порівняння, використовують ці два атрибути (наприклад, для організації транзитного рейсу можна дати запит «Видати рейси, в яких час вильоту з Києва до Львова більше часу прибуття з Дніпра до Києва»). Якщо ж значення двох атрибутів беруться з різних доменів, то їх порівняння, ймовірно, позбавлене сенсу: чи варто порівнювати номер рейсу з вартістю квитка?

Відносини на доменах D1, D2, ..., Dn складається з заголовка і тіла. На рис. 3.1 наведено приклад відносини для розкладу руху літаків. Ai – атрибути, Vi – значення атрибутів

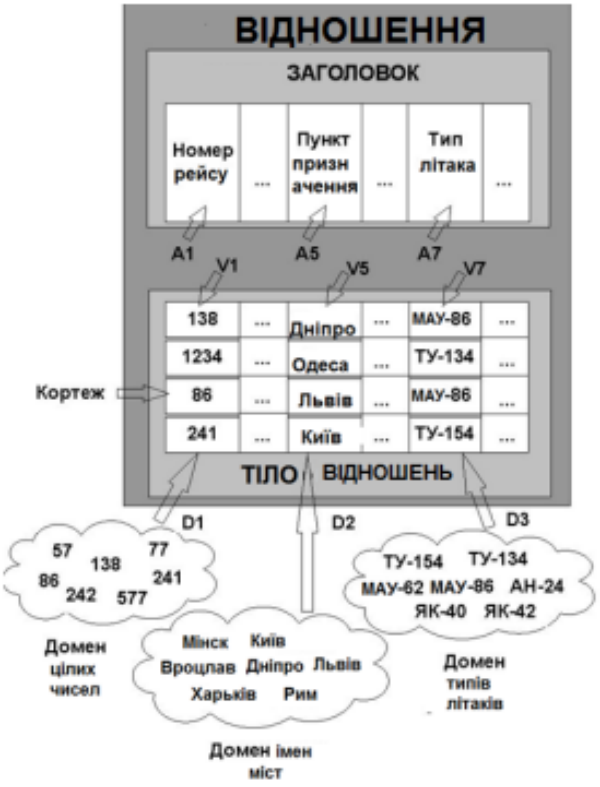


Рисунок 3.1 – Відносини (зв'язок) з математичної точки зору

Заголовок відносин складається з фіксованих атрибутів A1, A2, ..., An, де існує взаємно однозначна відповідність між цими атрибутами Ai і визначальними їх доменами Di (i = 1,2, ..., n).

Тіло відносин складається з множини кортежів, де кожен кортеж складається в свою чергу з множини пар атрибутів-значення (Ai: Vi), (i = 1,2, ..., n), де Vi є значенням з єдиного домену Di, який пов'язаний з атрибутом Ai.

Ступінь відносин – це число його атрибутів.

Відносини ступеня один називають унарним, ступеня два – бінарним, ступені три – тернарного, ..., а ступеня n – n-арним. Ступінь відносин «Рейс» (рис. 3.2) дорівнює 8.

Кардинальне число або потужність відносини – це число його кортежів. Потужність відносини «Рейс» дорівнює 10. Кардинальне число відносин змінюється в часі на відміну від його ступеня.

Оскільки відносини – це множини, а множин за визначенням не містять співпадаючих елементів, то ніякі два кортежі відносин не можуть бути дублікатами один одного в будь-який довільно заданий момент часу.

Нехай R – відношення з атрибутами A1, A2, ..., An. Кажуть, що множина атрибутів K = (Ai, Aj, ..., Ak) відносини R є можливим ключем R тоді і тільки тоді, коли виконуються дві незалежні від часу умови:

1. Унікальність: в довільний заданий момент часу ніякі два різних кортежі R не мають одного і того ж значення для Ai, Aj, ..., Ak.
2. Мінімальність: жоден з атрибутів Ai, Aj, ..., Ak не може бути виключений з K без порушення унікальності.

Кожне відношення має хоча б один можливий ключ, оскільки щонайменше комбінація всіх його атрибутів задовольняє умові унікальності. Один з можливих ключів (вибраний довільним чином) приймається за його первинний ключ. Інші можливі ключі, якщо вони є, називаються альтернативними ключами.

Вищезазначені та деякі інші математичні поняття з'явилися теоретичною базою для створення реляційних СУБД, розробки відповідних мовних засобів і програмних систем, що забезпечують їх високу продуктивність, і створення основ теорії проектування баз даних.

Також на практиці широко використовуються неформальні еквіваленти

цих понять: Відносини – Таблиця, Кортеж – Рядок таблиці або Запис, Атрибут – Стовпець Таблиці або Поле.

При цьому приймається, що «запис» означає «екземпляр запису», а «поле» означає «ім'я і тип поля».

3.2 Побудова бази даних

Розглянемо приклад побудови інфологічної моделі бази даних «Харчування», де повинна зберігатися інформація про страви, їх щоденному споживанні, продуктах, з яких готуються ці страви, і постачальників цих продуктів. Інформація буде використовуватися кухарем і керівником невеликого підприємства громадського харчування, а також його відвідувачами.

За допомогою зазначених користувачів виділені наступні об'єкти і характеристики проектованої бази (рис.3.2):

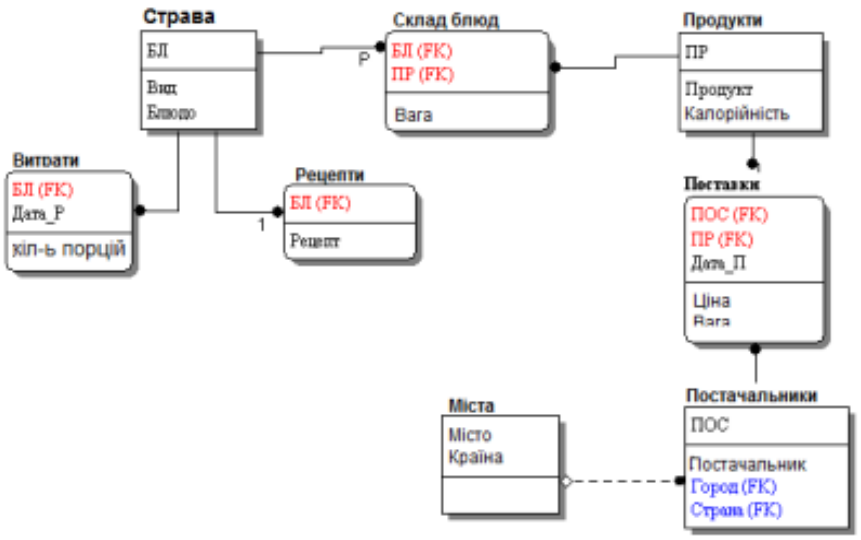


Рисунок 3.2 – Модель бази даних «Харчування»

1. Страви, для опису яких потрібні дані, що входять до їх кулінарні рецепти: номер страви (наприклад, з книги кулінарних рецептів), назва страви, вид страви (закуска, суп, гаряче і т.п.), рецепт (технологія приготування страви), вихід (вага порції), а також назва, калорійність і вага кожного продукту, що входить в страви;

2. Для кожного постачальника продуктів: найменування, місто і країна, назва поставленого товару, дата поставки і ціна на момент поставки.

3. Щоденне споживання страв (витрата): страва, кількість порцій, дата.

Аналіз дозволяє виділити:

1. незалежні сутності – Продукти, Страви, Постачальники, Міста;

2. залежні сутності – Склад страв, Поставки, Рецепти, Витрата страв.

У цих моделях Страва, Продукт і Постачальник – найменування, а БЛ, ПР і ПОС – цифрові коди страв, продуктів і постачальників продуктів.

3.3 Реляційна база даних

Реляційна база даних – це сукупність відносин, що містять всю інформацію, яка повинна зберігатися в БД.

Однак користувачі можуть сприймати таку базу даних як сукупність таблиць. Так на рис. 3.3 показані таблиці бази даних, побудовані згідно інфологічної моделі бази даних «Харчування».

1. Кожна таблиця складається з однотипних рядків і має унікальне ім'я.

2. Рядки мають фіксоване число полів (стовпців) і значень (множинні поля і повторювані групи неприпустимі). Інакше кажучи, в кожній позиції таблиці на перетині рядка і стовпця завжди є в точності одне значення або нічого.

3. Рядки таблиці обов'язково відрізняються один від одного хоча б єдиним значенням, що дозволяє однозначно ідентифікувати будь-який рядок такої таблиці.

4. Стовпцям таблиці однозначно присвоюються імена, і в кожному з них розміщуються однорідні значення даних (дати, прізвища, цілі числа або грошові суми).

5. Повний інформаційний зміст бази даних представляється у вигляді явних значень даних і такий метод подання є єдиним. Зокрема, не існує будь-яких спеціальних «зв'язків» або покажчиків, що з'єднують одну таблицю з іншого. Так, зв'язку між рядком з БЛ = 2 таблиці «Страви» на рис. 3.3 і рядком з

ПР = 7 таблиці «Продукти» (для приготування «Харчо» потрібен «Рис»), видається не за допомогою покажчиків, а завдяки існуванню в таблиці «Склад» рядки, в якій номер страви дорівнює 2, а номер продукту дорівнює 7.

Страви

БЛ	Страва	Вид
1	Лобіо	Закуска
2	Харчо	Суп
3	Шашлик	Гаряче
4	Кава	Десерт

Расход

БЛ	Порций	Дата_Р
1	158	1/9/94
2	144	1/9/94
3	207	1/9/94
4	235	1/9/94
...	...	...

Продукти

ПР	Продукт	Калор.
1	Фасоль	3070
2	Цибуля	450
3	Масло	7420
4	Зелень	180
5	М'ясо	1660
6	Помідори	240
7	Рис	3340
8	Кава	2750

Рецепти

БЛ	Рецепт
1	Ломаную очищ
...	...

Состав

БЛ	ПР	Вага(г)
1	1	200
1	2	40
1	3	30
1	4	10
2	5	80
2	2	30
2	6	40
2	7	50
2	3	15
2	4	15
3	5	180
3	6	100
3	2	40
3	4	20
4	8	8

Постачальники

ПОС	Поставщик	Город
1	"Полісся"	Київ
2	"Наталка"	Київ
3	"Хуанхэ"	Пекін
4	"Лайма"	Рига
5	"Юрмала"	Рига
6	"Даугава"	Рига

Города

Город	Страна
Київ	Україна
Пекін	Китай

Поставки

ПОС	ПР	Вага (кг)	Ціна	Дата_П
1	6	120	0.45	27/8/94
1	3	50	1.82	27/8/94
1	2	50	0.61	27/8/94
2	2	100	0.52	27/8/94
2	5	100	2.18	27/8/94
2	4	10	0.88	27/8/94
3	1	250	0.37	24/8/94
3	7	75	0.44	24/8/94
3	8	40	2.87	24/8/94
4	3	70	1.56	30/8/94
5	5	200	2.05	30/8/94

Рисунок 3.3– База даних «Харчування»

6. При виконанні операцій з таблицею її рядки і стовпці можна обробляти в будь-якому порядку безвідносно до їх інформаційного змісту. Цьому сприяє

наявність імен таблиць і їх стовпців, а також можливість виділення будь-якої їх рядки або будь-якого набору рядків із зазначеними ознаками.

3.4 Управління реляційними базами даними

Прагнення до мінімізації числа таблиць для зберігання даних може привести до виникнення різних проблем при їх оновленні та будуть надані рекомендації щодо розбиття деяких великих таблиць на кілька маленьких. Але як сформувати необхідний відповідь, якщо потрібні для нього дані зберігаються в різних таблицях?

Запропонувавши реляційну модель даних, Е. Кодд створив і інструмент для зручної роботи з відносинами – реляційну алгебру. Кожна операція цієї алгебри використовує одну або кілька таблиць (відносин) в якості її операндів і продукує в результаті нову таблицю, тобто дозволяє «розрізати» або «склеювати» таблиці (рис.3.4).

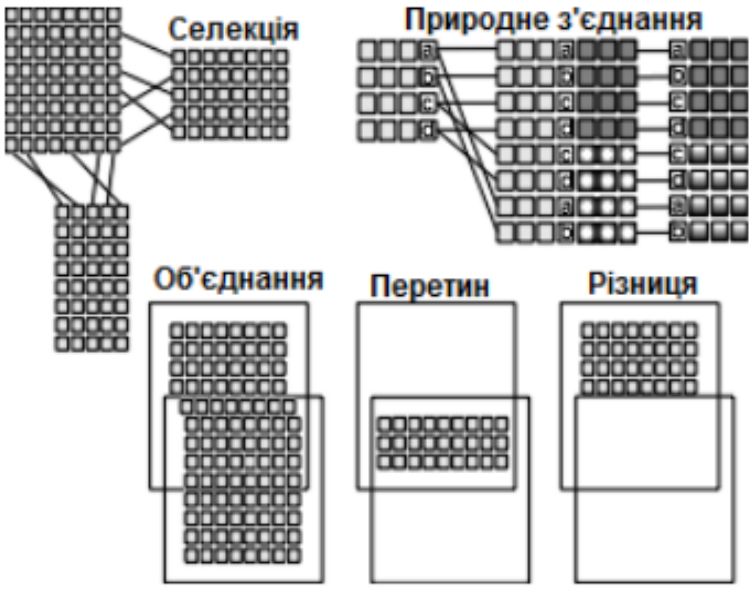


Рисунок 3.4 – Деякі операції реляційної алгебри

Створено мови управління (маніпулювання) даними, що дозволяють реалізувати всі операції реляційної алгебри і практично будь-які їх поєднання. Серед них найбільш поширений SQL (Structured Query Language –

Структурована мову запитів).

За допомогою єдиного запиту на мові SQL можна з'єднати кілька таблиць в тимчасову таблицю і вибрати з неї необхідні рядки і стовпці (селекція і проекція).

Основна мета проектування БД – це скорочення надмірності збережених даних, а отже, економія обсягу використовуваної пам'яті, зменшення витрат на багаторазові операції оновлення надлишкових копій і усунення можливості виникнення протиріч через зберігання в різних місцях відомостей про одне й те ж об'єкті. Так званий, «чистий» проект БД («Кожен факт в одному місці») можна створити, використовуючи методологію нормалізації відносин. І хоча нормалізація повинна використовуватися на завершальній перевірочної стадії проектування БД, розглянемо причини, які змусили Кодда створити основи теорії нормалізації.

Припустимо, що проектування бази даних «Харчування» починається з виявлення атрибутів і підбору даних, зразок яких показаний в табл.3.2.

Таблиця 3.2 – Дані для створення БД «Харчування»

Страва	Вид	Рецепт	Порції	Дата Р	Продукт	Калорійність	Вага (г)	Постачальник	Місто	Країна	Вага (кг)	Ціна ($)	Дата П
Лобио	Закуска	Лом.	158	1/9/94	Фасоль	3070	200	"Хуанхэ"	Пекін	Китай	250	0.37	24/8/94
					Цибуля	450	40	"Наталка"	Київ	Україна	100	0.52	27/8/94
					Масло	7420	30	"Лайма"	Рига	Латвія	70	1.55	30/8/94
					Зелень	180	10	"Даугава"	Рига	Латвія	15	0.99	30/8/94
Шашлик	Горяче	...	207	1/9/94	М'ясо	1660	180	"Юрмала"	Рига	Латвия	200	2.05	30/8/94
					Цибуля	450	40	"Полесье"	Київ	Україна	50	0.61	27/8/94
					Помідори	240	100	"Полесье"	Київ	Україна	120	0.45	27/8/94
					Зелень	180	20	"Даугава"	Рига	Латвія	15	0.99	30/8/94
Кава	Десерт	...	235	1/9/94	Кава	2750	8	"Хуанхэ"	Пекін	Китай	40	2.87	24/8/94

Цей варіант таблиці «Харчування» не є відношенням, так як більшість її рядків не атомарний. Атомарними є лише значення полів Блюдо, Вид, Рецепт

(хоча він і великий), Працюй і Дата_Р інші ж поля таблиці 3.2 – множинні. Для додання таких даних форми відносини необхідно реконструювати таблицю. Найпростіше це зробити за допомогою простого процесу вставки. Однак таке перетворення призводить до виникнення великого обсягу надлишкових даних.

3.5 Теорія нормалізації реляційної моделі даних

Встановлення функціональної залежності й одержання найкращого з погляду мінімальності подання множини функціональних залежностей дозволять побудувати найбільш оптимальний варіант БД, що забезпечує надійність зберігання й обробки даних на основі методів еквівалентних перетворень схем відношень реляційної БД. Процес вирішення такого завдання називається нормалізацією відношень інформаційної моделі ПО й полягає у перетворенні її об'єктів у логічні таблиці БД. Основні вимоги наведені нижче:

– первинні ключі відношень повинні бути мінімальними;

– число відношень БД повинне по можливості давати найменшу надмірність даних – вимога надійності даних;

– число відношень БД не повинне приводити до втрати продуктивності системи;

– дані не повинні бути суперечливими, тобто при виконанні операцій включення, видалення й відновлення даних їх потенційна суперечливість повинна бути зведена до мінімуму;

– схема відношень БД повинна бути стійкою, здатною адаптуватися до змін при її розширенні додатковими атрибутами – вимога гнучкості структури БД;

– розкид часу реакції на різні запити до БД не повинен бути великим;

– дані повинні правильно відбивати стан ПО БД у кожен конкретний момент часу – вимога актуальності даних.

Створення системи, що одночасно задовольняє всім вищезгаданим вимогам, являє собою складну оптимізаційну задачу, що часом не має однозначного вирішення.

Теорія функціональних залежностей дозволяє встановити певні вимоги до схем відношень у реляційній БД. Ці вимоги формулюються у термінах властивостей відношень і називаються нормальними формами схем відношень. Кожна нормальна форма відношень пов'язана з певним класом функціональної залежності, які представлені у відношеннях. Одним з очевидних засобів усунення потенційної суперечливості даних у відношеннях логічної моделі реляційної БД є їх розбиття на двоє або більше відношень, у кожному з яких є присутньою тільки одна функціональна залежність.

Процес усунення потенційної суперечливості й надмірності даних у відношеннях реляційної БД називається *нормалізацією вихідних схем відношень*. Нормалізація відношень полягає у виконанні декомпозиції відношень, що перебувають у попередній нормальній формі, на двоє або більше відношень, які задовольняють вимогам наступної нормальної форми.

У теорії реляційних БД звичайно виділяється така послідовність нормальних форм: перша нормальна форма (1NF); друга нормальна форма (2NF); третя нормальна форма (3NF); нормальна форма Бойса-Кодда (BCNF); четверта нормальна форма (4NF); п'ята нормальна форма, або нормальна форма проекції-з'єднання (5NF або PJ/NF).

Основні властивості нормальних форм полягають у такому: кожна наступна нормальна форма у деякому змісті краще попередньої нормальної форми; при переході до наступної нормальної форми властивості попередніх нормальних форм зберігаються.

Перша нормальна форма відношення

Відношення перебуває у 1NF, якщо всі атрибути відношення є простими (вимогу атомарності атрибутів), тобто не мають компонентів. Іншими словами, домен атрибута повинен складатися з неподільних значень і не може містити в собі безліч значень із більше елементарних доменів. Інколи у відношеннях деякі функціональні залежності атрибутів від можливого ключа не є мінімальними. Це призводить до так званих аномалій відновлення. Під *аномаліями відновлення* розуміються труднощі, з якими зустрічаються при виконанні операцій

додавання кортежів у відношення (INSERT), видалення кортежів (DELETE) і модифікації кортежів (UPDATE).

Друга нормальна форма відношення

Будемо вважати атрибут відношення ключовим, якщо він є елементом якого-небудь ключа відношення. В іншому випадку атрибут буде вважатися неключовим. Відношення перебуває у 2NF, якщо воно перебуває у 1NF, і всі неключові атрибути відношення функціонально мінімально залежать від первинного ключа. Іншими словами, 2NF вимагає, щоб відношення не містило часткових функціональних залежностей.

Третя нормальна форма відношення

Відношення перебуває у 3NF, якщо воно перебуває в 2NF, і всі неключові атрибути відношення залежать тільки від первинного ключа. Іншими словами, 3NF вимагає, щоб відношення не містило транзитивних функціонального зв’язку неключових атрибутів від ключа.

Таким чином, процедура зведення відношення до 3NF складається у виконанні двох проекцій: по правій і по лівій частині транзитивного функціонального зв’язку.

Зрозуміло, що в процесі нормалізації декомпозиція відношення на незалежні проекції є кращою. Необхідні й достатні умови незалежності проекцій відношення забезпечує *теорема Риссанена*: проекції r1 і r2 відношення r є незалежними тоді й тільки тоді, коли кожний зв'язок у відношенні r логічно виходить зі зв'язку у r1 і r2; загальні атрибути r1 і r2 утворять можливий ключ хоча б для одного з цих відношень.

Атомарним відношенням називається відношення, яке неможливо декомпозувати на незалежні проекції. Далеко не завжди для неатомарних відношень потрібна декомпозиція на атомарні проекції. При виборі способу декомпозиції необхідно прагнути до одержання незалежних проекцій, але не обов'язково атомарних.

Нормальна форма Бойса-Кодда

Змінна відношення перебуває в нормальній формі Бойса-Кодда (BCNF) у

тому і тільки в тому випадку, коли будь-який виконуваний для цього змінного відношення нетривіальний і мінімальний функціональний зв'язок має як детермінант деякий можливий ключ даного відношення.

Четверта нормальна форма відношення

У змінній відношення r з атрибутами A, B, C (у загальному випадку складовими) є багатозначна залежність B від A (AB) в тому і тільки в тому випадку, коли множина значень атрибута B, що відповідає парі значень атрибутів A й C, залежить від значення A і не залежить від значення C. Багатозначні залежності мають цікаву властивість "подвійності", що демонструє така лема Фейджина:

У відношенні r {A, B, C} виконується MVD A→→B у тому і тільки в тому випадку, коли виконується MVD A→→C.

Функціональний зв'язок є частковим випадком MVD, коли множина значень залежного атрибута обов'язково складається з одного елемента. Таким чином, якщо виконується зв'язок A→B, то виконується й MVD A→→B.

Теорема Фейджина. Нехай є змінна відношення r з атрибутами A, B, C (у загальному випадку, складовими). Відношення r декомпозується без втрат на проекції {A, B} й {A, C} тоді й тільки тоді, коли для нього виконується MVD A→→B | C.

Відношення перебуває у 4NF, якщо воно перебуває в 3NF або BCNF і всі незалежні багатозначні функціональні зв’язки рознесені в окремі відношення з тим самим ключем. Іншими словами, 4NF застосовується при наявності у відношенні більш ніж однієї MVD і вимагає, щоб відношення не містило незалежних багатозначних MVD.

3.6 Недоліки проекту БД «Харчування»

Початківець-проектувальник буде використовувати відношення «Харчування» з вставками в якості завершеною БД. Дійсно, навіщо розбивати відношення «Харчування» на кілька дрібніших відносин, якщо воно містить в

собі всі дані? А розбивати треба тому, що при використанні універсального відносини виникає кілька проблем:

1. *Надмірність*. Дані практично всіх стовпців багаторазово повторюються. Повторюються і деякі набори даних (Блюдо-Від-Рецепт, Продукт-Калорійність, Постачальник-Місто-Країна). Небажано повторення рецептів, деякі з яких набагато більше рецепта «Лобіо». І вже зовсім погано, що всі дані про страву (включаючи рецепт) повторюються кожен раз, коли це блюдо включається в меню.

2. *Потенційна суперечливість* (аномалії оновлення). Внаслідок надмірності можна оновити адресу постачальника в одному рядку, залишаючи його незмінним в інших. Якщо постачальник кави повідомив про свій переїзд в Харбін і була оновлена рядок з продуктом кави, то у постачальника «Хуанхе» з'являється дві адреси, один з яких не актуальний. Отже, при оновленнях необхідно переглядати всю таблицю для знаходження і зміни всіх відповідних рядків.

3. *Аномалії включення.* В БД не може бути записаний новий постачальник («Нярінга», Вільнюс, Литва), якщо поставлений їм продукт (Огірки) не використовується ні в одній страві. Можна, звичайно, помістити невизначені значення в стовпці Страва, Вид, Працюй і Вага (г) для цього постачальника. Але якщо з'явиться блюдо, в якому використовується цей продукт, не забудемо ми видалити рядок з невизначеними значеннями?

З аналогічних причин не можна ввести і новий продукт (наприклад, Баклажани), який пропонує існуючий постачальник (наприклад, «Полісся»). А як ввести нове блюдо, якщо в ньому використовується новий продукт («Краби»)?

4. *Аномалії видалення.* Зворотній проблема виникає при необхідності видалення всіх продуктів, що поставляються даними постачальником або всіх страв, які використовують ці продукти. При таких видаленнях будуть втрачені відомості про такий постачальника.

Багато проблем з цього прикладу зникнуть, якщо виділити в окремі таблиці, як в розглянутому вище прикладі, відомості про страви, рецепти, витрати страв, продукти та їх постачальників, а також створити сполучні таблиці «Склад» і «Поставки».

Включення. Простим додаванням рядків (Постачальники; «Нярінга», Вільнюс, Литва) і (Поставки; «Нярінга», Вільнюс, Огірки, 40) можна ввести інформацію про новий постачальника. Аналогічно можна ввести дані про новий продукт (Продукти; Баклажани, 240) і (Поставки; «Полісся», Київ, Баклажани, 50).

Видалення. Видалення відомостей про деякі поставки або страви не приводить до втрати відомостей про постачальників.

Оновлення. У таблиці 3.2 все ще багато повторюваних даних, що знаходяться в сполучних таблицях (Склад і Поставки). Отже, в даному варіанті БД збереглася потенційна суперечливість: для зміни назви постачальника з «Полісся» на «Дніпро» доведеться змінювати не тільки рядок таблиці Постачальники, але і множину рядків таблиці Поставки. При цьому не виключено, що в БД будуть одночасно зберігатися: «Полісся», «скотарство», «Дніпро», «Дніпро» та інші варіанти назв. Крім того, що повторюються текстові дані (такі як назва страви «Рулет з телячої грудинки з сосисками і гарніром з різнобарвного пюре» або продукту «Ковбаса сирокопчена») істотно збільшують обсяг збережених даних.

Для виключення посилань на довгі текстові значення останні зазвичай нумерують: номер страви у великих кулінарних книгах, товари (продукти) в каталогах і т.д. Скористаємося цим прийомом для виключення надмірного дублювання даних і появи помилок при копіюванні довгих текстових значень. Тепер при зміні назви постачальника «Полісся» на «Дніпро» виправляється єдине значення в таблиці Постачальники. І навіть якщо воно вводиться з помилкою («Дніпро»), то це не може вплинути на зв'язок між постачальниками і продуктами (в сполучній таблиці Поставки використовуються номери постачальників і продуктів, а не їх назви).

4. РЕЛЯЦІЙНА АЛГЕБРА

Реляційна алгебра – це теоретична мова операцій, що дозволяють створювати на основі одного або декількох відношень інше ставлення без зміни самих вихідних відношень. Таким чином, обидва операнда і результат є відношеннями, тому результати однієї операції можуть застосовуватися в іншій операції.

Це дозволяє створювати вкладені вирази реляційної алгебри але за будь-якої глибини вкладеності результатом є відношення. Така властивість називається замкнутістю. Вона наголошує на тому, що застосування будь-якої кількості операцій реляційної алгебри до відношень не призводить до створення інших об'єктів, крім відношень.

Реляційна алгебра є мовою послідовного використання відношень, в якій всі кортежі, можливо, навіть взяті з різних відношень, обробляються однією командою, без організації циклів.

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

П'ять основних операцій реляційної алгебри :

1. Вибірка (selection).
2. Проекція (projection).
3. Декартовий добуток (cartesian product).
4. Об'єднання (union).
5. Різниця множин (set difference).

Ці операції виконують більшість дій з вилучення даних, які можуть становити для нас інтерес. На підставі п'яти основних операцій можна також винести додаткові операції, такі як:

1. З'єднання (join).
2. Перетин (intersection).
3. Розподіл (division).

Операції вибірки і проєкції є унарними, оскільки вони працюють з одним відношенням. Інші операції працюють з парами відношень, і тому їх називають бінарними операціями.

У наведених нижче визначеннях R_1 і R_2 – це два відношення, визначені на атрибутах $A = (a_1, a_{2,}.. a_n)$ і $B = (b_1, b_2,..., b_m)$ відповідно.

4.1 Унарні операції

Вибірка (або обмеження) S предикат (R)

Операція вибірки застосовується до одного відношення R і визначає результуюче відношення, яке містить тільки ті кортежі (рядки) з відношення R, які задовольняють заданій умові (предикату).

Обозначение	Определение	Синтаксис
$R[A\ \theta v]$ $R[A_1\ \theta A_2]$	$\{r : r \in R \wedge (r[A]\ \theta v)\}$ $\{r : r \in R \wedge (r[A_1]\ \theta r[A_2])\}$	терм WHERE условие

$$P[D_2 = 11] = \begin{bmatrix} 1 & 11 & x \\ 2 & 11 & y \\ 3 & 11 & z \end{bmatrix}$$

Рисунок 4.1 – Приклад використання операції вибірки

Узагальнена вибірка це унарний оператор, що записується як $\rho_\varphi(R)$, де φ є формулою числення висловлень, що складається із атомів, дозволених у звичайній вибірці та логічних операторів (кон'юнкції), (диз'юнкції) та (заперечення). Така вибірка вибирає всі кортежі зі значенням ІСТИНА.

Операція проекції P ai, aj, ..., az (R) застосовується до одного відношення, (R) і визначає нове відношення містить вертикальну підмножину відношень R, створювану за допомогою вилучення значень зазначених атрибутів.

Обозначение	Определение	Синтаксис
$R[A]$	$\{r[A] : r \in R\}$	R[список атрибутов через запятую]

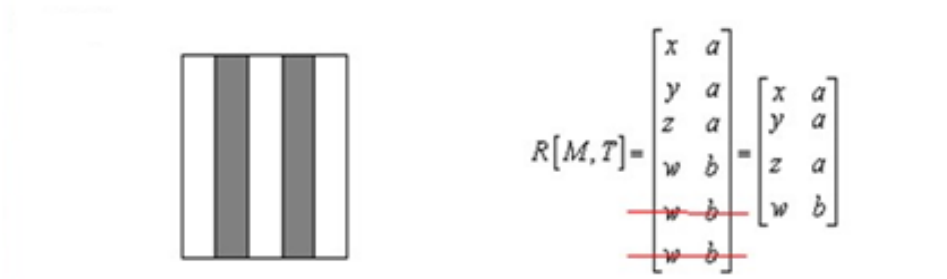


Рисунок 4.2 – Приклад використання операції проекція

Перейменування є унарним оператором, що записується як $\rho_{a/b}(R)$. Результат застосування оператора ідентичний за винятком того, що поле в усіх кортежах перейменовується на поле *a*. Цей оператор застосовується для простого перейменування атрибута відношення, або самого відношення.

В результаті застосування операції перейменування отримуємо нове відношення зі зміненими іменами атрибутів.

Синтаксис: *R* RENAME $Atr_1, Atr_2, \ldots$ AS $NewAtr_1, NewAtr_2, \ldots$

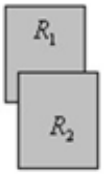
де *R* – відношення $Atr_1, Atr_2, \ldots$ – вихідні імена атрибутів $NewAtr_1, NewAtr_2, \ldots$ – нові імена атрибутів

4.2 Операції з множинами (бінарні)

Об'єднання

Об'єднання двох відношень R_1 та (було u) R_2 визначає нове відношення, яке включає всі кортежі, що містяться тільки в R_1, і тільки в R_2, водночас в R_1 і R_2 причому всі дублікати кортежів виключені. При цьому відношення R_1 і R_2 мають бути сумісними з об'єднанням.

Обозначение	Определение	Синтаксис
$R_1 \cup R_2$	$\{r : r \in R_1 \vee r \in R_2\}$	(R_1) UNION (R_2)



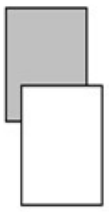
$$R[Q,T] \cup S = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \end{bmatrix} \cup \begin{bmatrix} 5 & a \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \end{bmatrix}$$

Рисунок 4.3 – Приклад використання операції об’єднання

Якщо R1 і R_2 включають, відповідно, I і J кортежів, то об'єднання цих відношень можна отримати, зібравши всі кортежі в одне відношення, яке може містити не більше (I + J) кортежів. Об'єднання можливе, тільки якщо схеми двох відношень збігаються, тобто складаються з однакової кількості атрибутів, причому кожна пара відповідних атрибутів має однаковий домен. Інакше говорячи, відношення мають бути сумісними з об'єднання. Зазначимо, що у визначенні сумісності з об'єднанням не вказано, що атрибути повинні мати однакові імена. У деяких випадках для отримання двох сумісних з об'єднанням відношень може бути використана операція проекції.

Різниця $R_1 – R_2$. (рис.4.4)

Обозначение	Определение	Синтаксис
$R_1 - R_2$	$\{r : r \in R_1 \wedge r \notin R_2\}$	(R_1) MINUS (R_2)



$$R[Q,T] - S = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \end{bmatrix} \cap \begin{bmatrix} 5 & a \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} 3 & a \\ 9 & a \\ 2 & b \\ 4 & b \end{bmatrix}$$

Рисунок 4.4 – Приклад використання операції різниця

Різниця двох відношень R_1 і R_2 складається з кортежів, які є у відношенні R_1, але відсутні у відношенні R_2. Причому відношення R_1 і R_2 мають бути сумісними з об'єднанням.

Перетин $R_1 \cap R_2$.

Операція перетину (рис.4.5) визначає відношення, яке містить кортежі, присутні як у відношенні R_1, так і у відношенні R_2. Відношення R_1 і R_2 мають бути сумісними з об'єднанням.

Обозначение	Определение	Синтаксис
$R_1 \bigcap R_2$	$\{r : r \in R_1 \wedge r \in R_2\}$	(R_1) INTERSECT (R_2)

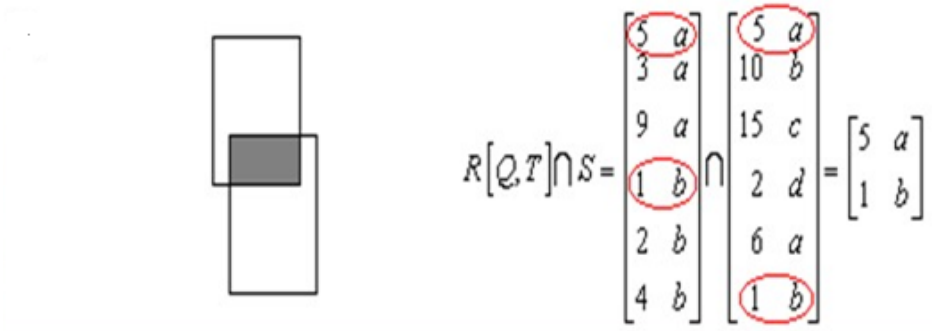


Рисунок 4.5 – Приклад використання операції перетину

Декартовий добуток R1 ⊗ R2.

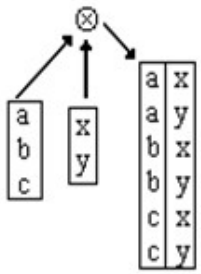
Операція декартового добутку визначає нове відношення, яке є результатом зчеплення (тобто конкатенації) кожного кортежу з відношення R1 з кожним кортежем з відношення R2.

Операція декартового добутку застосовується для множення двох відношень. Множенням двох відношень називається створення іншого відношення, що складається з усіх можливих пар кортежів обох відношень.

Отже, якщо одне відношення має I кортежів і N атрибутів, а інше – J кортежів і M атрибутів, то їх декартовий добуток буде містити (I x J) кортежів і

(N + M) атрибутів. Якщо подібні відношення містять атрибути з однаковими іменами, то в цьому випадку імена атрибутів міститимуть назви відношень у вигляді префіксів. Це необхідно для забезпечення унікальності імен атрибутів у відношенні, отриманому як результат виконання операції декартового добутку.

Обозначение	Определение	Синтаксис
$R_1 \otimes R_2$	$\{(r_1 \| r_2) : r_1 \in R_1 \wedge r_2 \in R_2\}$	(R_1) TIMES (R_2)



$$R[M,T] \otimes (R[Q,T] \cap S) = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \end{bmatrix} \otimes \begin{bmatrix} 5 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} x & a & 5 & a \\ x & a & 1 & b \\ y & a & 5 & a \\ y & a & 1 & b \\ z & a & 5 & a \\ z & a & 1 & b \\ w & b & 5 & a \\ w & b & 1 & b \end{bmatrix}$$

Рисунок 4.6 – Приклад використання операції декартового добутку

Операція DIVIDE (ділення)

Обозначение	Определение	Ситнтаксис
$R_1[A_1 \div A_2]R_2$	$\{r[A_1] : r \in R_1 \wedge R_2[A_2] \subseteq g_R(r[A_1])\}$	(R_1) DIVIDE BY (R_2)

$$R_1(A_1, A_2) = \begin{bmatrix} a & x \\ a & y \\ a & z \\ b & x \\ c & y \end{bmatrix} \quad R_2(A_3, A_4) = \begin{bmatrix} 1 & x \\ 2 & x \\ 1 & y \end{bmatrix} \quad R_1[A_2 \div A_4] = \begin{bmatrix} a & x \\ a & y \\ a & z \\ b & x \\ c & y \end{bmatrix} \div \begin{bmatrix} x \\ y \end{bmatrix} = [a]$$

Рисунок 4.7 – Приклад застосування операції ділення

Нехай задані відносини R (A, B1) і S (B2, C) такі, що B1 і B2 визначені на одному й тому самому домені (доменах). Тоді R [B1 ÷ B2] S – це максимальна

підмножина R [A] таке, що його декартовий добуток з S [B2] включається в R. Цей оператор є алгебраїчним двійником квантора загальності.

Декомпозиція складних операцій

Операції реляційної алгебри можуть у кінцевому підсумку стати надзвичайно складними. Для спрощення такі операції можна розбити на ряд менших операцій реляційної алгебри і привласнити імена результатам проміжних виразів. Для присвоювання імен результатам операції реляційної алгебри використовується операція присвоювання, позначена стрілкою вліво ($\leftarrow$).

Дія, яка при цьому виконується, аналогічна операції присвоювання у мові програмування; в даному випадку результат виразу праворуч від знака операції $\leftarrow$ присвоюється вислову, що знаходиться зліва від знака операції.

Операція з'єднання

Зазвичай користувачів цікавить лише деяка частина всіх комбінацій кортежів декартового добутку, яка задовольняє заданій умові. Тому замість декартового добутку частіше використовується одна з найважливіших операцій реляційної алгебри – операція з'єднання. Внаслідок її виконання на базі двох

Обозначение	Определение
$R_1[A_1\ \theta A_2]R_2$	$\{(r_1 \| r_2): r_1 \in R_1 \wedge r_2 \in R_2 \wedge (r_1[A_1]\ \theta r_2[A_2])\}$

Синтаксис: (R_1) JOIN (R_2)

$$P[D_3 = D_4]Q = \begin{bmatrix} 1 & 11 & x & x & 1 \\ 1 & 11 & x & x & 2 \\ 2 & 11 & y & y & 1 \\ 4 & 12 & x & x & 1 \\ 4 & 12 & x & x & 2 \end{bmatrix} = \begin{bmatrix} 1 & 11 & x & 1 \\ 1 & 11 & x & 2 \\ 2 & 11 & y & 1 \\ 4 & 12 & x & 1 \\ 4 & 12 & x & 2 \end{bmatrix}$$

Рисунок 4.8 – Приклад використання операції з'єднання

вихідних відношень створюється деяке нове відношення. Операція з'єднання є похідною від операції декартового добутку, тому що вона еквівалентна операції вибірки з декартового добутку двох операндів – відношень тих кортежів, які задовольняють умові, вказаній у предикаті з'єднання як формули вибірки.

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

Нижче перераховані різні типи операцій з'єднання, які дещо відрізняються один від одного і можуть бути тією чи іншою мірою корисні.

- Тета-з'єднання (theta join).
- З'єднання за еквівалентністю (equijoin), яке є окремим видом Тета-з'єднання.
- Природне з'єднання (natural join).
- Зовнішнє з'єднання (outer join).
- Напівз'єднання (semijoin).

Тета-з'єднання

Операція тета-з'єднання R> <FS визначає відношення, яке містить кортежі з декартового добутку відносин R і S, що задовольняють предикату F. Предикат F має вигляд Rai θ Sbj, де замість θ може бути вказаний один з операторів порівняння (>,> =, <, <=, =, <>).

Якщо предикат F містить тільки оператор рівності (=), то з'єднання називається з'єднанням за еквівалентністю – EQUI–JOIN.

Натуральне з'єднання (natural join)

Таке з'єднання нічим не відрізняється від EQUI-JOIN за винятком того, що в цьому випадку видаляються надлишкові стовпці, згенеровані в процесі виконання з'єднання. Природне з'єднання – це з'єднання, що використовується в ході нормалізації сукупності відносин.

5. АПАРАТНІ ТА ПРОГРАМНІ СКЛАДОВІ ДЛЯ ПРОЄКТУВАННЯ БД

5.1 Апаратні складові БД

Для роботи СУБД і програм необхідно деяке апаратне забезпечення. Воно може варіювати в дуже широких межах – від єдиного персонального комп'ютера чи одного мейнфрейма до мережі з багатьох комп'ютерів. Апаратне забезпечення залежить від вимог даної організації і СУБД, яка використовується. Одні СУБД призначені для роботи тільки з конкретними типами операційних систем чи устаткування, інші можуть працювати із широким колом апаратного забезпечення і різних операційних систем. Найбільш поширеною системою вважається така, що складається з мережі комп'ютерів з одним центральним комп'ютером. На центральному комп'ютері працює серверна частина СУБД (backend), що обслуговує і контролює доступ до бази даних. До нього мають доступ інші комп'ютери, які можуть бути розташовані і в інших регіонах. На них працюють клієнтські частини СУБД (frontend), що здійснюють взаємодію з користувачами. Подібна архітектура зветься клієнт/сервер (client-server), де сервером є комп'ютер із серверною астиною СУБД, а клієнтами – комп'ютери з клієнтськими частинами БД (рис.5.1).

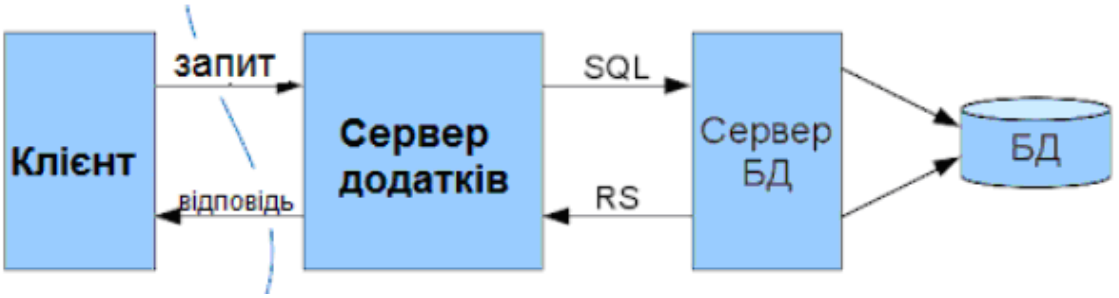


Рисунок 5.1 – Клієнт-серверна БД

Цей компонент охоплює програмне забезпечення самої СУБД і прикладних програм, разом з операційною системою, включаючи і мережне програмне забезпечення, якщо СУБД використовується в мережі. Як правило

програми створюються мовами третього покоління, такими як C, Pascal чи VB, а також мовами четвертого покоління, таких як SQL, оператори яких впроваджують у програми на мовах третього покоління. Утім, СУБД може мати з власні інструменти четвертого покоління, призначені для швидкої розробки програм з використанням убудованих непроцедурних мов запитів, генераторів звітів, форм, графічних зображень і навіть повномасштабних програм. Використання інструментів четвертого покоління може істотно підвищити продуктивність системи і сприяти створенню більш зручних для обслуговування програм.

Імовірно, найважливішим компонентом середовища БД (з погляду кінцевих користувачів) є дані. База даних містить як робочі дані, так і мета-дані, тобто «дані про дані». Структура бази даних називається схемою (schema). У нашому простому прикладі схема бази даних складається з чотирьох файлів, чи таблиць (table) (рис.5.2):

– офісні меблі;

– вітальні;

– дитячі меблі;

– кухні;

– дачні меблі.

Кожна з наведених таблиць має відповідні поля або атрибути. Крім своєї основної функції характеристики конкретних даних деякі атрибути виконують роль зв’язку з іншими таблицями.

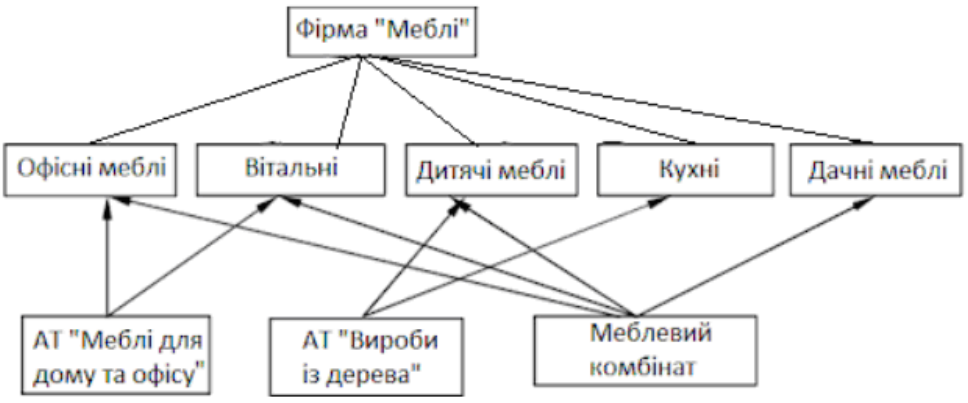


Рисунок 5.2 – Структура БД «Меблі»

Наприклад атрибут *Власник* у таблиці *Об'єкти нерухомості* моделює зв'язок між цією таблицею і таблицею *Власник*, тобто деякий власник володіє деякою нерухомістю, що здається в оренду.

У системному каталозі містяться наступні зведення:

– імена, типи і розміри елементів даних;

– імена зв'язків;

– обмеження цілісності даних;

– імена зареєстрованих користувачів, яким надані деякі права доступу до даних;

– індекси і структури збереження – наприклад, інвертовані файли чи дерева B+.

До процедур відносяться інструкції і правила, що повинні враховуватися при проектуванні і використанні бази даних. Користувачам і обслуговуючому персоналу бази даних необхідно надати документацію, що містить докладний опис процедур використання і супроводи даної системи, включаючи інструкції про правила виконання приведених нижче дій:

– реєстрація в БД;

– використання окремого інструмента БД чи програми;

– запуск та зупинка БД;

– створення резервних копій БД;

– обробка збоїв апаратного і програмного забезпечення, включаючи процедури ідентифікації компонента, що вийшов з ладу, виправлення компонента, що відмовив, (наприклад, за допомогою виклику фахівця з ремонту апаратного забезпечення), а також відновлення бази даних після усунення несправності;

– зміна структури таблиці, реорганізація бази даних, розміщеної на декількох дисках, способи поліпшення продуктивності і методи архівації даних на вторинних пристроях збереження.

Останнім, ще не розглянутим нами компонентом середовища БД, є користувачі системи. Існує чотири групи користувачів:

– адміністратори даних і баз даних;

– розроблювачі баз даних;

– прикладні програмісти;

– кінцеві користувачі.

База даних і БД є корпоративними ресурсами, якими варто управляти так само, як і будь-якими іншими ресурсами. Звичайне керування даними і базою даних передбачає управління і контроль за СУБД і розміщеними в неї даними.

Адміністратор даних, чи АД (Data Administrator -DA), відповідає за керування даними, включаючи планування бази даних, розробку і супровід стандартів, бізнес правил і ділових процедур, а також за концептуальне і логічне проектування бази даних. АД консультує і дає свої рекомендації керівництву вищої ланки, контролюючи відповідності загального напрямку розвитку бази даних установленим корпоративним цілям.

Адміністратор бази даних, чи АБД (Database Administrator – DBA), відповідає за фізичну реалізацію бази даних, включаючи фізичне проектування і втілення проекту, за забезпечення безпеки і цілісності даних, за супровід операційної системи, а також за забезпечення максимальної продуктивності програм і користувачів. У порівнянні з АД, обов'язки АБД носять більш технічний характер, для чого необхідне знання конкретної СУБД. і системного оточення. В одних організаціях між цими ролями не робиться розходжень, а в інших важливість корпоративних ресурсів відбита саме у виділенні окремих груп персоналу з зазначеним колом обов'язків.

У проектуванні великих баз даних беруть участь два різних типи *розроблювачів*: розроблювачі логічної бази даних і розроблювачі фізичної бази даних. *Розроблювач логічної бази даних* займається ідентифікацією даних (тобто сутностей і їх атрибутів), зв'язків між даними і встановлює обмеження, що накладаються на збережені дані. Розроблювач логічної бази даних повинний мати всебічне і повне розуміння структури даних організації і їх бізнес-правил. Бізнес-правила описують основні характеристики даних з погляду організації.

Для ефективної роботи розроблювач логічної бази даних повинен якомога раніше включити всіх передбачуваних користувачів бази даних у процес створення моделі даних і його робота поділяється на два етапи.

Концептуальне проектування бази даних, що зовсім не залежить від таких деталей її втілення, як конкретна цільова СУБД, програми, мови програмування чи будь-якої іншої фізичної характеристики.

Логічне проектування бази даних, що проводиться з урахуванням особливостей обраної моделі даних: реляційної, мережної, ієрархічної чи об'єктно орієнтованій.

Розроблювач фізичної бази даних одержує готову логічну модель даних, займається її фізичною реалізацією, у тому числі:

– перетворенням логічної моделі даних у набір таблиць і обмежень цілісності даних;

– вибором конкретних структур збереження і методів доступу до даних, що забезпечують необхідний рівень продуктивності, при роботі з базою даних;

– проектуванням будь-яких необхідних мір захисту даних.

Багато етапів фізичного проектування бази даних у значній мірі залежать від обраної цільової СУБД, а тому може існувати кілька різних способів реалізації необхідної схеми. Якщо концептуальне і логічне проектування бази даних відповідає на запитання «що?», то фізичне проектування відповідає на запитання «як?». Для рішення цих задач вимагаються різні навички роботи, якими найчастіше володіють різні люди.

Відразу після створення бази даних варто приступити до розробки програм, що надають користувачам необхідні їм функціональні можливості. Саме цю роботу і виконують прикладні програмісти. Звичайно прикладні програмісти працюють на основі специфікацій, створених системними аналітиками. Як правило, кожна програма містить деякі оператори, що вимагають від БД виконання визначених дій з базою даних – наприклад, такі як витяг, вставка, чи відновлення видалення даних.

Користувачі є клієнтами бази даних – вона проєктується, створюється і підтримується для того, щоб обслуговувати їх інформаційні потреби. Користувачів можна класифікувати за способами використання ними системи.

Наївні користувачі звичайно навіть і не підозрюють про наявність СУБД. Вони звертаються до бази даних за допомогою спеціальних програм які дозволяють у максимальній мірі спростити операції, які вони виконують. Такі користувачі ініціюють виконання операцій у бази даних, способом введення найпростіших команд чи вибираючи команди меню. Це значить, що таким користувачам не потрібно нічого знати про базу даних БД.

Досвідчені користувачі вже знайомі зі структурою бази даних і можливостями СУБД. Для виконання необхідних операцій вони можуть використовувати таку мову запитів високого рівня, як SQL. А деякі досвідчені користувачі можуть навіть створювати власні прикладні програми.

5.2 Програмні складові БД

Розглянемо програмні продукти проектування БД.

MySQL – вільна система управління базами даних. MySQL є власністю компанії Oracle Corporation, що отримала її разом з поглиненою Sun Microsystems, яка здійснює розробку і підтримку додатку. Розповсюджується під GNU General Public License і під власною комерційною ліцензією, на вибір.

Крім цього розробники створюють функціональність на замовлення ліцензійних користувачів, саме завдяки такому замовленню майже в найраніших версіях з'явився механізм реплікації.

Цю систему управління базами даних з відкритим кодом було створено як альтернатива комерційним системам. MySQL із самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL – одна з найпоширеніших систем управління базами даних. Вона використовується, у першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування.

MySQL є рішенням для малих і середніх додатків. Зазвичай MySQL використовується як сервер, до якого звертаються локальні або віддалені клієнти, проте до дистрибутиву входить бібліотека внутрішнього сервера, що дозволяє включати MySQL до автономних програм. Вихідні коди сервера компілюються на багатьох платформах. Найповніше можливості сервера виявляються в UNIX системах, де є підтримка багатонитевості, що підвищує продуктивність системи в цілому.

Гнучкість СУБД MySQL забезпечується підтримкою великої кількості типів таблиць: користувачі можуть вибрати як таблиці типу MyISAM, що підтримують повнотекстовий пошук, так і таблиці InnoDB, що підтримують транзакції на рівні окремих записів. Більш того, СУБД MySQL поставляється із спеціальним типом таблиць EXAMPLE, що демонструє принципи створення нових типів таблиць. Завдяки відкритій архітектурі й GPL-ліцензуванню, в СУБД

У MySQL постійно з'являються нові типи таблиць. MySQL характеризується великою швидкістю, стійкістю і простотою використання. Для некомерційного використання MySQL є безкоштовною.

Можливості сервера MySQL:

– простота у встановленні та використанні;

– підтримується необмежена кількість користувачів, що одночасно працюють із БД;

– кількість рядків у таблицях може досягати 50 млн.;

– висока швидкість виконання команд;

– наявність простої та ефективної системи безпеки.

PostgreSQL – об'єктно-реляційна система управління базами даних. Є альтернативою як комерційним СУБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СУБД з відкритим кодом (MySQL, Firebird, SQLite).

Порівняно до інших проектів з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією,

її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СУБД та впроваджувати у неї найновіші досягнення.

СУБД Oracle – це найпотужніший програмний комплекс, що дозволяє створювати додатки будь-якої складності. Ядром цього комплексу є база даних, що зберігає інформацію, кількість якої за рахунок наданих засобів масштабування практично безмежна. З високою ефективністю працювати з цією інформацією одночасно може практично будь-яка кількість користувачів (за наявності достатніх апаратних ресурсів), не проявляючи тенденції до зниження продуктивності системи при різкому збільшенні їхньої кількості.

Механізми масштабності в СУБД Oracle останньої версії дозволяють безмежно збільшувати потужність і швидкість роботи сервера Oracle і своїх додатків, просто додаючи нові й нові вузли кластеру. Це не вимагає зупинки працюючих додатків, не вимагає переписування старих додатків, розроблених для звичайної одномашинної архітектури. Крім того, вихід з ладу окремих вузлів кластера також не призводить до зупинки програми.

Вбудована мова JavaVM до СУБД Oracle, повномасштабна підтримка серверних технологій (Java Server Pages, Java-сервлети, модулі Enterprise JavaBeans, інтерфейси прикладного програмування CORBA), привели до того, що Oracle на сьогоднішній день де-факто є стандартом СУБД для Internet.

Ще однією складовою успіху СУБД Oracle є те, що вона поставляється практично для всіх існуючих на сьогодні операційних систем. Працюючи під Sun Solaris, Linux, Windows або на інший операційній системі з продуктами Oracle не буде виникати ніяких проблем у роботі. СУБД Oracle однаково добре працює на будь-якій платформі. Таким чином, компаніям, які розпочинають роботу з продуктами Oracle не доводиться міняти мережеве оточення. Існує лише невелика кількість відмінностей при роботі з СУБД, обумовлених особливостями тієї або іншої операційної системи. У цілому ж це завжди та ж сама безпечна, надійна і зручна СУБД Oracle.

Microsoft SQL Server – система управління реляційними базами даних, розроблена корпорацією Microsoft. Основна використовувана мова запитів –

Transact-SQL, створена спільно Microsoft та Sybase. Transact-SQL є реалізацією стандарту ANSI / ISO щодо структурованої мови запитів (SQL) із розширеннями. Використовується для роботи з базами даних розміром від персональних до великих баз даних масштабу підприємства, конкурує з іншими СУБД у цьому сегменті ринку.

При взаємодії з мережею Microsoft SQL Server і Sybase ASE використовують протокол рівня додатків під назвою Tabular Data Stream (TDS, протокол передачі табличних даних). Протокол TDS також був реалізований у проекті FreeTDS з метою забезпечити різні додатки можливістю взаємодії з базами даних Microsoft SQL Server і Sybase.

Для забезпечення доступу до даних Microsoft SQL Server підтримує Open Database Connectivity (ODBC) – інтерфейс взаємодії додатків з СУБД. SQL Server надає можливість підключення користувачів через веб-сервіси, що використовують протокол SOAP. Це дозволяє клієнтським програмам, не призначеним для Windows, кросплатформно з'єднуватися з SQL Server.

6. МОВА ЗАПИТІВ SQL

6.1 Структура мови мови запитів SQL

У реалізаціях конкретних реляційних БД зараз не використовується в чистому вигляді ні реляційна алгебра, ні реляційне числення. Фактичним стандартом доступу до реляційних даних стала мова SQL (Structured Query Language) - Структурована мова запитів. Мова SQL являє собою суміш операторів реляційної алгебри і виразів реляційного числення, що використовує синтаксис, близький до фраз англійської мови і розширений додатковими можливостями, відсутніми в реляційної алгебри та реляційному численні.

Сьогодні мова SQL підтримується багатьма десятками СУБД різних типів, розроблених для найрізноманітніших обчислювальних платформ, починаючи від персональних комп'ютерів і закінчуючи мейнфреймами.

Основні категорії команд мови SQL:

– DDL – мова визначення даних;

– DML – мова маніпулювання даними;

– DQL – мова запитів;

– DAL – мова управління даними;

– команди адміністрування даних;

– команди управління трансакціями.

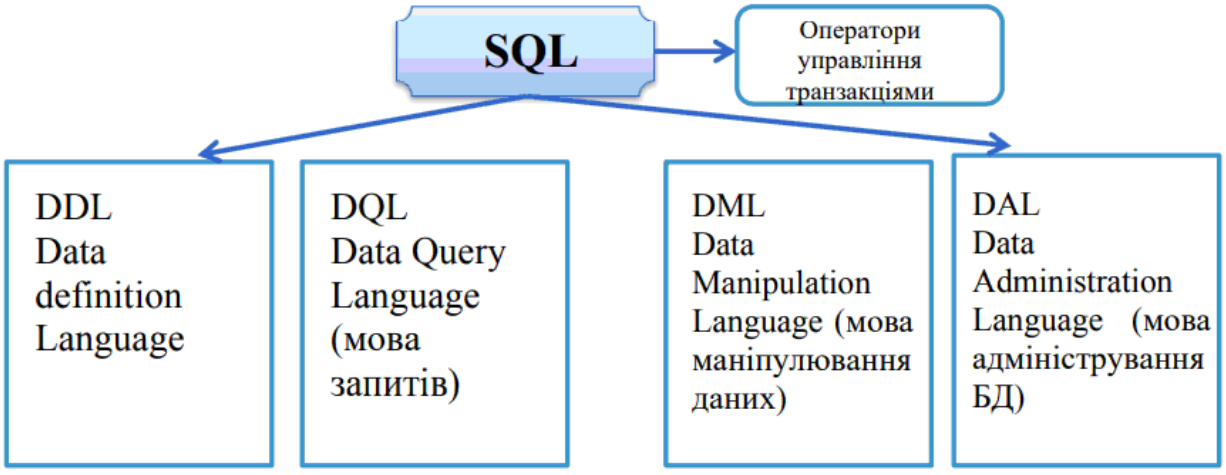


Рисунок 6.1 – Структура мови запитів SQL

6.1.1 Визначення структур бази даних (DDL)

На відміну від реляційної алгебри, де були представлені тільки операції запитів до БД, SQL є повною мовою, в ньому присутні не тільки операції запитів, але і оператори, відповідні DDL – Data Definition Language – мови опису даних. Крім того, мова містить оператори, що призначені для управління (адміністрування) БД.

Мова визначення даних (Data Definition Language, DDL) дозволяє створювати і змінювати структуру об'єктів бази даних, наприклад, створювати і видаляти таблиці. Основними командами мови DDL є наступні: CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX.

6.1.2 Маніпулювання даними (DML)

Мова маніпулювання даними (Data Manipulation Language DML) використовується для маніпулювання інформацією усередині об'єктів реляційної бази даних за допомогою трьох основних команд: INSERT, UPDATE, DELETE.

6.1.3 Вибірка даних (DQL)

Мова запитів DQL найбільш відома користувачам реляційної бази даних, не дивлячись на те, що він включає одну команду: SELECT. Ця команда разом зі своїми багаточисельними опціями і пропозиціями використовується для формування запитів до реляційної бази даних.

6.1.4 Мова управління даними (DCL)

Команди управління даними дозволяють управляти доступом до інформації, бази даних, що знаходиться усередині. Як правило, вони використовуються для створення об'єктів, пов'язаних з доступом до даних, а також служать для контролю над розподілом привілеїв між користувачами. Команди управління даними наступні: GRANT, REVOKE.

6.1.5 Команди адміністрування даних

За допомогою команд адміністрування даних користувач здійснює контроль за виконуваними діями і аналізує операції бази даних; вони також можуть виявитися корисними при аналізі ефективності системи. Не слід плутати адміністрування даних з адмініструванням бази даних, яке є загальним управлінням базою даних і має на увазі використання команд всіх рівнів.

6.2 Основні команди SQL

SQL - проста мова програмування, яка має небагато команд і яка була розроблена для роботи з БД, а саме, щоб отримувати/добавляти/змінювати дані, мати можливість опрацьовувати великі масиви інформації та швидко отримувати структуровану та згруповану інформацію.

Є багато варіантів мови SQL, але у них всіх основні команди майже однакові. Також існує і багато СУБД, але основними з них являються: Microsoft Access, Microsoft SQL Server, MySQL, Oracle SQL, IBM DB2 SQL, PostgreSQL та Sybase Adaptive Server SQL. Щоб працювати з SQL кодом, нам потрібна буде одна з вище перелічених СУБД. Для навчання ми будемо використовувати СУБД Microsoft Access.

Таблиця 6.1 – Таблиця продажу товару

ID	Month	Product	City	Quantity	Amount
2	April	Bikes	Montreal	12	$4 500,00
3	April	Bikes	Montreal	56	$21 000,00
4	April	Bikes	San Francisco	854	$320 250,00
5	April	Bikes	New York	25	$9 375,00
6	April	Skates	Montreal	56	$5 544,00
7	April	Skates	Toronto	854	$84 546,00
8	April	Skates	San Francisco	25	$2 475,00
9	April	Skates	New York	663	$65 637,00
10	April	Skis Long	Montreal	854	$209 230,00
11	April	Skis Long	Toronto	25	$6 125,00
12	April	Skis Long	San Francisco	663	$162 435,00
13	April	Skis Long	New York	21	$5 145,00
14	April	Skis Short	Montreal	21	$4 389,00
15	April	Skis Short	Toronto	4	$836,00
16	April	Skis Short	San Francisco	522	$109 098,00

SQL як і інші мови програмування має свої команди (оператори), за

допомогою яких віддаються вказівки для вибірки даних. Але потрібно враховувати, що всі оператори в SQL нечутливі до регістру, тому можна їх писати як великими буквами, так і маленькими (як правило, їх прийнято писати великими буквами, щоб розрізняти від назв полів та таблиць). Назви же таблиць та полів є навпаки чутливими до регістру та мають писатися точно як в БД.

Щоб розглянути як працюють оператори SQL, використовуємо уявну БД з інформацією про реалізовану продукцію (Таблиця 6.1).

6.2.1. Вибірка даних (SELECT)

Найпершим та найголовнішим оператором в SQL являється SELECT. З його допомогою ми можемо відбирати необхідні нам поля з даними в таблиці.

1. Вибірка окремих полів.

```
SELECT Product FROM Sumproduct
```

Бачимо, що наш SQL запит відібрав колонку Product з таблиці Sumproduct.

Product
Bikes
Bikes
Bikes
Bikes
Skates
Skates
Skates
Skates
Skis Long
Skis Long

2. Вибірка кількох полів.

Припустимо, нам необхідно вибрати назву та кількість реалізованого товару. Для цього просто перераховуємо необхідні поля через кому:

```
SELECT Product FROM Sumproduct
```

Product	Quantity
Bikes	12
Bikes	56
Bikes	854
Bikes	25
Skates	56
Skates	854
Skates	25
Skates	663
Skis Long	854
Skates	25
Skis Long	663

3. Вибірка всіх стовпців.

Якщо ж нам необхідно отримати всю таблицю зі всіма полями, тоді просто ставимо знак зірочка (*):

SELECT * FROM Sumproduct

ID	Month	Product	City	Quantity	Amount
2	April	Bikes	Montreal	12	$4 500,00
3	April	Bikes	Montreal	56	$21 000,00
4	April	Bikes	San Francisco	854	$320 250,00
5	April	Bikes	New York	25	$9 375,00
6	April	Skates	Montreal	56	$5 544,00
7	April	Skates	Toronto	854	$84 546,00
8	April	Skates	San Francisco	25	$2 475,00
9	April	Skates	New York	663	$65 637,00
10	April	Skis Long	Montreal	854	$209 230,00
11	April	Skis Long	Toronto	25	$6 125,00
12	April	Skis Long	San Francisco	663	$162 435,00
13	April	Skis Long	New York	21	$5 145,00
14	April	Skis Short	Montreal	21	$4 389,00
15	April	Skis Short	Toronto	4	$836,00
16	April	Skis Short	San Francisco	522	$109 098,00

6.2.2 Сортування даних (ORDER BY)

В майбутньому нам може знадобитися посортувати нашу вибірку - в алфавітному порядку для тексту чи по зростанню/спаданню - для цифрових значень. Для таких цілей в SQL є спеціальний оператор ORDER BY.

1. Сортування вибраних даних.

Давайте усю нашу таблицю посортуємо по сумі реалізації продукції, а саме по стовпцю *Amount*.

SELECT * FROM Sumproduct ORDER BY Amount

ID	Month	Product	City	Quantity	Amount
47	January	Skates	New York	4	$396,00
15	April	Skis Short	Toronto	4	$836,00
35	February	Skis Short	Toronto	4	$836,00
74	March	Skis Short	Montreal	4	$836,00
52	January	Skis Long	San Francisco	4	$980,00
41	February	Snow Board	New York	4	$1 232,00
18	April	Snow Board	Montreal	4	$1 232,00
79	March	Snow Board	Toronto	4	$1 232,00
62	March	Bikes	Montreal	4	$1 500,00
28	February	Skates	San Francisco	21	$2 079,00
26	February	Skates	Montreal	21	$2 079,00
46	January	Skates	Montreal	21	$2 079,00
8	April	Skates	San Francisco	25	$2 475,00

Бачимо, що запит відсортував записи по зростанню в полі *Amount*. Обов'язково потрібно дотримуватись послідовності розташування операторів, тобто оператор ORDER BY має йти в самому кінці запиту. В іншому випадку буде отримане повідомлення про помилку.

Також, особливістю оператора ORDER BY є те, що він може сортувати дані по полю, якого ми не вибирали у запиті, тобто достатньо щоб воно взагалі було в БД.

2. Сортування за кількома полями.

Тепер посортуємо наш приклад додатково за ще одним полем. Нехай це буде поле City, яке відображає місце реалізації продукції.

```
SELECT * FROM Sumproduct ORDER BY Amount, City
```

ID	Month	Product	City	Quantity	Amount
47	January	Skates	New York	4	$396,00
74	March	Skis Short	Montreal	4	$836,00
35	February	Skis Short	Toronto	4	$836,00
15	April	Skis Short	Toronto	4	$836,00
52	January	Skis Long	San Francisco	4	$980,00
18	April	Snow Board	Montreal	4	$1 232,00
41	February	Snow Board	New York	4	$1 232,00
79	March	Snow Board	Toronto	4	$1 232,00
62	March	Bikes	Montreal	4	$1 500,00
26	February	Skates	Montreal	21	$2 079,00
46	January	Skates	Montreal	21	$2 079,00
28	February	Skates	San Francisco	21	$2 079,00

Черговість сортування буде залежати від порядку розташування полів в запиті. Тобто, в нашому випадку спочатку дані будуть посортовані по колонці *Amount*, а потім по *City*.

3. Напрямок сортування.

Незважаючи на те, що по замовчуванню оператор ORDER BY сортує по зростанню, ми можемо також прописати сортування значень по спаданню. Для цього в кінці кожного поля проставляємо оператор DESC (що є скороченням від слова DESCENDING).

```
SELECT * FROM Sumproduct ORDER BY Amount DESC, City
```

ID	Month	Product	City	Quantity	Amount
4	April	Bikes	San Francisco	854	$320 250,00
22	February	Bikes	Montreal	663	$248 625,00
25	February	Bikes	New York	658	$246 750,00
70	March	Skis Long	Montreal	854	$209 230,00
10	April	Skis Long	Montreal	854	$209 230,00
21	April	Snow Board	New York	663	$204 204,00
59	January	Snow Board	Toronto	663	$204 204,00
63	March	Bikes	Montreal	522	$195 750,00
12	April	Skis Long	San Francisco	663	$162 435,00
32	February	Skis Long	San Francisco	663	$162 435,00
71	March	Skis Long	Toronto	663	$162 435,00
58	January	Snow Board	Montreal	522	$160 776,00
80	March	Snow Board	San Francisco	522	$160 776,00

В даному прикладі, значення в полі *Amount* були посортовані по спаданню, а в полі *City* - по зростанню. Оператор *DESC* застосовується лише для одного стовпця, тому при потребі його потрібно прописувати після кожного поля, яке приймає участь у сортуванні.

6.2.3 Фільтрування даних (WHERE)

В більшості випадків необхідно отримувати не всі записи, а лише ті, які відповідають певним критеріям. Тому для здійснення фільтрації вибірки в SQL є спеціальний оператор WHERE.

1. Просте фільтрування оператором WHERE.

Давайте з нашої таблиці, наприклад, відберемо записи, які стосуються лише певного товару. Для цього ми зазначимо додатковий параметр відбору, який фільтруватиме значення по колонці *Product*. Приклад запиту для відбору текстових значень:

```
SELECT * FROM Sumproduct WHERE Product = 'Bikes'
```

ID	Month	Product	City	Quantity	Amount
2	April	Bikes	Montreal	12	$4 500,00
3	April	Bikes	Montreal	56	$21 000,00
4	April	Bikes	San Francisco	854	$320 250,00
5	April	Bikes	New York	25	$9 375,00
22	February	Bikes	Montreal	663	$248 625,00
23	February	Bikes	San Francisco	21	$7 875,00
24	February	Bikes	San Francisco	54	$20 250,00
25	February	Bikes	New York	658	$246 750,00
42	January	Bikes	Montreal	75	$28 125,00
43	January	Bikes	Toronto	12	$4 500,00
44	January	Bikes	San Francisco	136	$51 000,00
45	January	Bikes	New York	21	$7 875,00
62	March	Bikes	Montreal	4	$1 500,00
63	March	Bikes	Montreal	522	$195 750,00

Як бачимо, умова відбору взята в одинарні лапки, що є обов'язковим при фільтруванні текстових значень. При фільтруванні числових значень лапки не потрібні.

Приклад запиту для відбору числових значень:

```
SELECT * FROM Sumproduct WHERE Amount > 40000 ORDER BY Amount
```

ID	Month	Product	City	Quantity	Amount
39	February	Snow Board	Toronto	136	$41 888,00
61	January	Snow Board	New York	136	$41 888,00
64	March	Bikes	San Francisco	125	$46 875,00
44	January	Bikes	San Francisco	136	$51 000,00
48	January	Skates	San Francisco	522	$51 678,00
9	April	Skates	New York	663	$65 637,00
27	February	Skates	Toronto	663	$65 637,00
65	March	Bikes	New York	212	$79 500,00
7	April	Skates	Toronto	854	$84 546,00
67	March	Skates	Toronto	854	$84 546,00
36	February	Skis Short	San Francisco	522	$109 098,00

В цьому прикладі ми відібрали записи, в яких виручка від реалізації становила більше 40 тис. $ та, додатково, всі записи посортували по зростанню по полю Amount.

В таблиці нижче, зазначено перелік умовних операторів, які підтримуються SQL:

Знак операції	Значення
=	Дорівнює
<>	Не дорівнює
<	Менше
<=	Менше або рівне
>	Більше
>=	Більше або рівне
BETWEEN	Між двома значеннями
IS NULL	Відсутній запис

2. Фільтрування по діапазону значень (BETWEEN).

Для відбору даних, які лежать в певному діапазоні, використовується оператор BETWEEN. В наступному запиті будуть відібрані усі значення, які лежать в межах від 1000$ до 2000$ включно, в полі Amount.

```
SELECT * FROM Sumproduct WHERE Amount BETWEEN 1000 AND 2000
```

ID	Month	Product	City	Quantity	Amount
18	April	Snow Board	Montreal	4	$1 232,00
41	February	Snow Board	New York	4	$1 232,00
62	March	Bikes	Montreal	4	$1 500,00
79	March	Snow Board	Toronto	4	$1 232,00

Черговість сортування буде залежати від порядку розташування полів в запиті. Тобто, в нашому випадку спочатку дані будуть посортовані по колонці Amount, а потім по City.

3. Вибірка порожніх записів (IS NULL).

В SQL існує спеціальний оператор для вибірки порожніх записів (називаєьбся NULL). Порожнім записом вважається будь-яка комірка в таблиці, в яку не введено жодного символу. Якщо в комірку введено 0 або пробіл, то вважається, що поле заповнене.

```
SELECT * FROM Sumproduct WHERE Amount IS NULL
```

ID	Month	Product	City	Quantity	Amount
5	April	Bikes	New York	25	
19	April	Snow Board	Toronto	522	

В прикладі вище, ми навмисне видалили два значення в полі Amount, щоб продемонструвати роботу оператора NULL.

4. Розширене фільтрування (AND, OR).

Мова SQL не обмежується фільтруванням по одній умові, для власних цілей ви можете використовувати досить складні конструкції для вибірки даних одночасно по багатьом критеріям. Для цього в SQL є додаткові оператори, які розширюють можливості оператора WHERE. Такими операторами являються: AND, OR, IN, NOT. Наведемо кілька прикладів роботи даних операторів.

```
SELECT * FROM Sumproduct WHERE Amount > 40000 AND City = 'Toronto'
```

ID	Month	Product	City	Quantity	Amount
7	April	Skates	Toronto	854	$84 546,00
19	April	Snow Board	Toronto	522	$160 776,00
27	February	Skates	Toronto	663	$65 637,00
39	February	Snow Board	Toronto	136	$41 888,00
59	January	Snow Board	Toronto	663	$204 204,00
67	March	Skates	Toronto	854	$84 546,00
71	March	Skis Long	Toronto	663	$162 435,00
75	March	Skis Short	Toronto	522	$109 098,00

```
SELECT * FROM Sumproduct WHERE Month= 'April' OR Month= 'March'
```

ID	Month	Product	City	Quantity	Amount
15	April	Skis Short	Toronto	4	$836,00
74	March	Skis Short	Montreal	4	$836,00
18	April	Snow Board	Montreal	4	$1 232,00
79	March	Snow Board	Toronto	4	$1 232,00
62	March	Bikes	Montreal	4	$1 500,00
8	April	Skates	San Francisco	25	$2 475,00
77	March	Skis Short	San Francisco	21	$4 389,00
14	April	Skis Short	Montreal	21	$4 389,00
2	April	Bikes	Montreal	12	$4 500,00
13	April	Skis Long	New York	21	$5 145,00
72	March	Skis Long	San Francisco	21	$5 145,00

Давайте об'єднаємо оператори AND та OR. Для цього зробимо вибірку велосипедів (Bikes) та ковзанів (Skates), які були продані в березні (March).

```
SELECT * FROM Sumproduct WHERE Product = 'Bikes' OR Product = 'Skates' AND
Month= 'March'
```

ID	Month	Product	City	Quantity	Amount
2	April	Bikes	Montreal	12	$4 500,00
3	April	Bikes	Montreal	56	$21 000,00
4	April	Bikes	San Francisco	854	$320 250,00
5	April	Bikes	New York	25	$9 375,00
22	February	Bikes	Montreal	663	$248 625,00
23	February	Bikes	San Francisco	21	$7 875,00
24	February	Bikes	San Francisco	54	$20 250,00
25	February	Bikes	New York	658	$246 750,00
42	January	Bikes	Montreal	75	$28 125,00
43	January	Bikes	Toronto	12	$4 500,00
44	January	Bikes	San Francisco	136	$51 000,00
45	January	Bikes	New York	21	$7 875,00
62	March	Bikes	Montreal	4	$1 500,00
63	March	Bikes	Montreal	522	$195 750,00
64	March	Bikes	San Francisco	125	$46 875,00
65	March	Bikes	New York	212	$79 500,00
66	March	Skates	Montreal	56	$5 544,00
67	March	Skates	Toronto	854	$84 546,00
68	March	Skates	San Francisco	212	$20 988,00

Бачимо, що в нашу вибірку попало за багато значень (крім березня (March), також січень (January), лютий (February) та квітень (April)). В чому ж причина? А в тому, що SQL має пріоритети виконання команд. Тобто оператор AND має вищий пріоритет, ніж оператор OR, тому спочатку було відібрано записи з ковзанами, що продані в березні, а потім усі записи, що стосувалися велосипедів.

Отже, щоб отримати правильну вибірку, нам потрібно змінити пріоритети виконання команд. Для цього використаємо дужки, як в математиці. Тоді, спочатку будуть опрацьовані оператори в дужках, а потім - всі решта.

```
SELECT * FROM Sumproduct WHERE (Product = 'Bikes' OR Product = 'Skates') AND
Month= 'March'
```

ID	Month	Product	City	Quantity	Amount
62	March	Bikes	Montreal	4	$1 500,00
63	March	Bikes	Montreal	522	$195 750,00
64	March	Bikes	San Francisco	125	$46 875,00
65	March	Bikes	New York	212	$79 500,00
66	March	Skates	Montreal	56	$5 544,00
67	March	Skates	Toronto	854	$84 546,00
68	March	Skates	San Francisco	212	$20 988,00
69	March	Skates	New York	56	$5 544,00

5. Розширене фільтрування (оператор IN).

```
SELECT * FROM Sumproduct WHERE ID IN (4, 12, 58, 67)
```

ID	Month	Product	City	Quantity	Amount
4	April	Bikes	San Francisco	854	$320 250,00
12	April	Skis Long	San Francisco	663	$162 435,00
58	January	Snow Board	Montreal	522	$160 776,00
67	March	Skates	Toronto	854	$84 546,00

Оператор IN виконує ту ж саму функцію, що і OR, проте має ряд переваг:

При роботі з довгими списками, речення з IN легше читати;

Використовується менша кількість операторів, що пришвидшує обробку запиту;

Найважливіша перевага IN в тому, що в його конструкції можна використовувати додаткову конструкцію SELECT, що відкриває великі можливості для створення складних підзапитів.

6. Розширене фільтрування (оператор NOT).

```
SELECT * FROM Sumproduct WHERE NOT City IN ('Toronto', 'Montreal')
```

ID	Month	Product	City	Quantity	Amount
47	January	Skates	New York	4	$396,00
52	January	Skis Long	San Francisco	4	$980,00
41	February	Snow Board	New York	4	$1 232,00
28	February	Skates	San Francisco	21	$2 079,00
8	April	Skates	San Francisco	25	$2 475,00
77	March	Skis Short	San Francisco	21	$4 389,00
57	January	Skis Short	New York	21	$4 389,00
13	April	Skis Long	New York	21	$5 145,00
72	March	Skis Long	San Francisco	21	$5 145,00
33	February	Skis Long	New York	21	$5 145,00
69	March	Skates	New York	56	$5 544,00
40	February	Snow Board	San Francisco	21	$6 468,00

Ключове слово NOT дозволяє забрати непотрібні значення із вибірки. Також його особливістю є те, що воно проставляється перед назвою стовпця, який бере участь у фільтруванні, а не після.

6.2.4 Символи підстановки та регулярні вирази (LIKE)

асто, для фільтрації даних, нам потрібно буде здійснити вибірку не по точному співвпадінні умови, а по наближеному значенню. Тобто коли, наприклад, ми шукаємо товар, назва якого відповідає певному шаблону (регулярному виразу) або містить певні символи чи слова. Для таких цілей в SQL існує оператор LIKE, котрий шукає наближені значення. Для конструювання такого шаблону використовуються метасимволи (спеціальні символи, для пошуку частини значення), а саме: «знак відсотка» (%) або зірочка (*), «символ підкреслення» (_) або «знак питання» (?), «квадратні дужки» ([ ]).

1. Метасимвол знак відсотка (%) або зірочка (*)

Давайте з нашої таблиці, наприклад, відберемо записи, які стосуються лише товарів, які містять у своїй назві слово Skis (лижі). Для цього складемо відповідний шаблон:

```
SELECT * FROM Sumproduct WHERE Product LIKE '*Skis*'
```

ID	Month	Product	City	Quantity	Amount
10	April	Skis Long	Montreal	854	$209 230,00
11	April	Skis Long	Toronto	25	$6 125,00
12	April	Skis Long	San Francisco	663	$162 435,00
13	April	Skis Long	New York	21	$5 145,00
14	April	Skis Short	Montreal	21	$4 389,00
15	April	Skis Short	Toronto	4	$836,00
16	April	Skis Short	San Francisco	522	$109 098,00
17	April	Skis Short	New York	136	$28 424,00
30	February	Skis Long	Montreal	522	$127 890,00
31	February	Skis Long	Toronto	125	$30 625,00
32	February	Skis Long	San Francisco	663	$162 435,00

Як бачимо, СУБД відібрала лише ті записи, де в колонці Product були товари, які містять слово Skis. Також зазначимо, що в даному прикладі використовується метасимвол «зірочка» (*), оскільки СУБД Access не підтримує «знак відсотка» (%) для оператора LIKE.

2. Метасимвол знак підкреслення (_) або знак питання (?)

Знак підкреслення або знак питання застосовується для того, щоб замінити один символ у слові. Давайте в слові Bikes замінимо всі голосні літери на «знак питання» (?) і подивимось на результат:

```
SELECT * FROM Sumproduct WHERE Product LIKE 'B?k?s'
```

		Product	City	Quantity	Amount
25	February				
42	January	Bikes	Montreal	12	$4 500,00
3	April	Bikes	Montreal	56	$21 000,00
4	April	Bikes	San Francisco	854	$320 250,00
5	April	Bikes	New York	25	$9 375,00
22	February	Bikes	Montreal	663	$248 625,00
23	February	Bikes	San Francisco	21	$7 875,00
24	February	Bikes	San Francisco	54	$20 250,00
25	February	Bikes	New York	658	$246 750,00
42	January	Bikes	Montreal	75	$28 125,00

Ми використали метасимвол «знак питання» (?), оскільки СУБД Access не підтримує «знак підкреслення» (_) для оператора LIKE.

3. Метасимвол квадратні дужки ([ ])

Метасимвол «квадратні дужки» ([ ]) використовується для одночасного вказання набору символів, по яким потрібно здійснити пошук.

```
SELECT * FROM Sumproduct WHERE City LIKE '[TN]*'
```

ID	Month	Product	City	Quantity	Amount
5	April	Bikes	New York	25	$9 375,00
7	April	Skates	Toronto	854	$84 546,00
9	April	Skates	New York	663	$65 637,00
11	April	Skis Long	Toronto	25	$6 125,00
13	April	Skis Long	New York	21	$5 145,00
15	April	Skis Short	Toronto	4	$836,00
17	April	Skis Short	New York	136	$28 424,00
19	April	Snow Board	Toronto	522	$160 776,00
21	April	Snow Board	New York	663	$204 204,00

В прикладі вище, ми відібрали записи, де в полі City назви міст починаються з букви T або N. Також, в даному випадку, ми можемо використати ще один метасимвол, який виконує зворотню дію. Добавимо у наш регулярний вираз знак оклику (!), що означатиме «не дорівнює» (для СУБД Access) або знак степені (^) (для інших СУБД).

```
SELECT * FROM Sumproduct WHERE City LIKE '[!TN]*'
```

ID	Month	Product	City	Quantity	Amount
2	April	Bikes	Montreal	12	$4 500,00
3	April	Bikes	Montreal	56	$21 000,00
4	April	Bikes	San Francisco	854	$320 250,00
6	April	Skates	Montreal	56	$5 544,00
8	April	Skates	San Francisco	25	$2 475,00
10	April	Skis Long	Montreal	854	$209 230,00
12	April	Skis Long	San Francisco	663	$162 435,00
14	April	Skis Short	Montreal	21	$4 389,00
16	April	Skis Short	San Francisco	522	$109 098,00

Тобто, останній створений нами запит читатиметься як: вибрати усі колонки з таблиці Sumproduct, та лише ті записи, де в полі City назви міст не починаються на букви T або N. Додатково зазначимо, що набір букв в метасимволі «квадратні дужки» відповідає лише за одну позицію в тексті.

Ми можемо отримати аналогічний результат, якщо скористатися вже відомим нам оператором NOT, проте зі знаком оклику (!) запис буде коротшим.

6.2.5 Розрахункові (обчислювальні) поля

Для чого потрібно використовувати розрахункові поля? Як правило, інформація в БД представлена в розрізі окремих фрагментів, оскільки так легше структуризувати дані та оперувати ними. Проте нам часто буде потрібно використовувати не окремі частини даних, а вже поєднану та оброблену інформацію. Наприклад, часто необхідно поєднувати ім'я та прізвище клієнтів, поєднувати елементи адрес, які знаходяться в різних стовпцях таблиці, обробляти текст та окремі слова, букви та символи, підсумовувати загальну вартість покупки, відображати статистику по інформації, яка знаходиться в БД. Дані, зазвичай, зберігаються окремими «кусками», що вимагає їх додаткового опрацювання на стороні клієнтської програми. Проте є можливість отримувати вже оброблену інформацію за допомогою СУБД. Саме в цьому випадку допомагають розрахункові поля. Вони автоматично створюються при виконанні запиту і мають вигляд та властивості звичайних стовпців, які є вже наявні в таблиці. Єдина відмінність полягає в тому, що фізично розрахункових полів немає, тому вони не займають додаткового місця в БД, а тимчасово існують в

«оперативній пам'яті» СУБД. Перевагою виконання операцій на стороні СУБД є швидкість опрацювання даних.

1. Виконання математичних операцій

Одним із способів використання розрахункових полів є виконання математичних операцій над вибраними даними. Давайте на прикладі розглянемо як це відбувається, використавши знову нашу таблицю Sumproduct. Припустимо, на потрібно вирахувати середню ціну придбання кожного товару. Для цього потрібно переділити колонку Amount (сума) на Quantity (кількість):

```
SELECT DISTINCT Product, Amount/Quantity FROM Sumproduct
```

Product	Expr1001
Bikes	375
Skates	99
Skis Long	245
Skis Short	209
Snow Board	308

Як бачимо, СУБД відібрала всі найменування товарів та відобразила їх середню вартість в окремому стовпці, який був створений під час виконання запиту. Також можна помітити, що ми використали додатковий оператор DISTINCT, який нам потрібен для відображення унікальних назв товарів (без нього ми б отримали дублювання записів).

2. Використання псевдонімів

В попередньому прикладі ми розраховували середню вартісь покупки кожного товару та відобразили значення в розрахунковому стовпці. Проте надалі, нам буде незручно звертатися до цього поля, оскільки його назва є неінформативною для нас (СУБД дала назву полю - Expr1001). Проте ми можемо назвати поле самостійно, наперед вказавши його назву в запиті, тобто дати псевдонім. Давайте перепишемо попередній приклад та вкажемо псевдонім для розрахункового поля:

```
SELECT DISTINCT Product, Amount/Quantity AS AvgPrice FROM Sumproduct
```

Product	AvgPrice
Bikes	375
Skates	99
Skis Long	245
Skis Short	209
Snow Board	308

Бачимо, наше розрахункове поле отримало власну назву AvgPrice. Для цього ми використали оператор AS, після якого вказали необхідну нам назву. Варто зазначити, що в SQL підтримуються лише основні математичні операції: додавання(+), віднімання(-), множення(*), ділення(/). Також для зміни черговості виконання операції можна використовувати круглі дужки.

Часто псевдоніми використовують не тільки щоби називати розрахункові поля, але і для перейменування діючих. Це може бути необхідним, якщо діюче поле має довгу назву або назва не є достатньо інформативною.

3. З'єднання полів (конкатенація)

Крім математичних операцій ми також можемо поєднувати текст та виводити його в окремому полі. Давайте розглянемо, яким чином можна здійснити склеювання (конкатенацію) тексту. Маємо такий приклад:

```
SELECT Month + ' ' + Product AS NewField, Quantity FROM Sumproduct
```

NewField	Quantity
April Bikes	12
April Bikes	56
April Bikes	854
April Bikes	25
April Skates	56
April Skates	854
April Skates	25
April Skates	663
April Skis Long	854
April Skis Long	25
April Skis Long	663
April Skis Long	21

В цьому прикладі ми з'єднали значення в двох стовпцях та вивели результат в нове поле NewField.

6.2.6 Функції обробки даних

Як і в більшості мов програмування, в SQL існують функції для обробки даних. Варто зазначити, що на відміну від SQL-операторів, функції не є стандартизованими для усіх видів СУБД, тобто для виконання одних і тих самих операції над даними, різні СУБД мають свої власні назви функцій. Це означає, що код запиту написаний в одній СУБД може не працювати в іншій, і це потрібно враховувати в подальшому. Найбільше це стосується функцій для обробки текстових значень, перетворення типів даних та маніпуляцій над датами.

В більшості СУБД підтримується стандартний набір типів функцій, а саме:

- Текстові функції, що використовуються для обробки тексту (виокремлення частини символів з тексту, визначення довжини тексту, переведення символів в верхній або нижній регістр...);
- Числові функції. Використовуються для виконання математичних операцій над числовими значеннями;
- Функції дати та часу (здійснюються маніпулювання датою та часом, розраховуються період між датами, перевіряються дати на коректність тощо);
- Статистичні функції (для обчислення максимальних/мінімальних значень, середніх значень, підрахунок кількості та суми...);
- Системні функції (надають різного роду службову інформацію про СУБД, користувача та ін.).

У специфікації SQL:2003 визнані п'ять зумовлених загальних типів змінних, усередині яких можуть бути підтипи.

Таблиця 6.2 – Тип мінних, що визначено для SQL реалізація

Тип	SQL реалізація
Строковий (символьний)	CHARACTER (чи CHAR)
	CHARACTER VARYING (чи VARCHAR)
	CHARACTER LARGE OBJECT (чи CLOB)

Числовий	точні числові типи	integer; smallint; bigint; numeric; decimal.
	приблизні числові типи	real; double precision; float;
	логічний (булевий)	boolean
	дати-часу	date; time without time zone*; time with time zone; timestamp without time zone; timestamp with time zone

1. Функції SQL для обробки тексту

Реалізація SQL в СУБД Access має наступні функції для обробки тексту:

Знак операції	Значення
LEFT()	Відбирає символи в тексті зліва
RIGHT()	Відбирає символи в тексті справа
MID()	Відбирає символи з середини тексту
UCase()	Переводить символи в верхній регістр
LCase()	Переводить символи в нижній регістр
LTrim()	Видаляє всі порожні символи зліва від тексту
RTrim()	Видаляє всі порожні символи справа від тексту
Trim()	Видаляє всі порожні символи з обох боків тексту

Переведемо назви товарів у верхній регістр за допомогою функції UCase():

```
SELECT Product, UCase(Product) AS Product_UCase FROM Sumproduct
```

Product	Product_UCase
Bikes	BIKES
Bikes	BIKES
Bikes	BIKES
Bikes	BIKES
Skates	SKATES
Skates	SKATES
Skates	SKATES
Skates	SKATES
Skis Long	SKIS LONG
Skis Long	SKIS LONG

Виділимо перші три символи у тексті за допомогою функції LEFT():

```
SELECT Product, LEFT(Product, 3) AS Product_LEFT FROM Sumproduct
```

Product	Product_LEFT
Bikes	Bik
Bikes	Bik
Bikes	Bik
Bikes	Bik
Skates	Ska
Skates	Ska
Skates	Ska
Skates	Ska
Skis Long	Ski
Skis Long	Ski

2. Функції SQL для обробки чисел

Знак операції	Значення
SQR()	Повертає корінь квадратний вказаного числа
ABS()	Повертає абсолютне значення числа
EXP()	Повертає експоненту вказаного числа
SIN()	Повертає синус вказаного кута
COS()	Повертає косинус вказаного кута
TAN()	Повертає тангенс вказаного кута

Функції обробки чисел призначені для виконання математичних операцій над числовими даними. Ці функції призначені для алгебраїчних та геометричних обчислень, тому вони використовуються значно рідше функцій обробки дати та часу. Проте числові функції є найбільш стандартизованими для усіх версій SQL. Давайте поглянемо на перелік числових функцій:

Наприклад, напишемо запит для отримання кореня квадратного для чисел в стовпці Amount за допомогою функції SQR():

```
SELECT Amount, SQR(Amount) AS Amount_SQR FROM Sumproduct
```

Amount	Amount_SQR
$4 500,00	67,08203932499
$21 000,00	144,9137674619
$320 250,00	565,9063526768
$9 375,00	96,82458365519
$5 544,00	74,45804187595
$84 546,00	290,7679487151
$2 475,00	49,74937185533
$65 637,00	256,1971896801
$209 230,00	457,4166590757
$6 125,00	78,26237921249

3. Функції SQL для обробки дати та часу

Функції маніпулювання датою та часом являються одними з найважливіших і часто використовуваних функцій SQL. В базах даних значення дат та часу зберігаються в спеціальному форматі, тому їх неможливо використовувати напряму без додаткової обробки. Кожна СУБД має свій набір функцій для обробки дат, що, нажаль, не дозволяє переносити їх на інші платформи та реалізації SQL.

Список деяких функцій для обробки дати та часу в СУБД Access:

Знак операції	Значення
DatePart()	Повертає частину дати: рік, квартал, місяць, тиждень, день, годину, хвилини, секунди
Year(), Month()	Повертає рік та місяць відповідно
Hour(), Minute(), Second()	Повертає годину, хвилини та секунди вказаної дати
WeekdayName()	Повертає назву дня тижня

Подивимось на прикладі як працює функція DatePart():

```
SELECT Date1, DatePart("m", Date1) AS Month1 FROM Sumproduct
```

Date1	Month1
11.02.2013	2

Функція DatePart() має додатковий параметр, який нам дозволяє відобразити необхідну частину дати. В прикладі ми використали значення

параметра «m», який відображає номер місяця (таким же чином ми можемо відобразити рік - «yyyy», квартал - «q», день - «d», тиждень - «w», годину - «h», хвилини - «n», секунди - «s» тощо).

4. Статистичні функції SQL

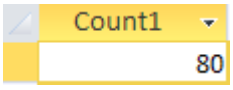
Статистичні функції допомагають нам отримати готові дані без їх вибірки. SQL-запити з цими функціями часто використовуються для аналізу та створення різноманітних звітів. Прикладом таких вибірок може бути: визначення кількості рядків в таблиці, отримання суми значень по певному полю, пошук найбільшого/найменшого або середнього значення в зазначеному стовпці таблиці. Також зазначимо, що статистичні функції підтримуються усіма СУБД без якихось особливих змін в написанні.

Список статистичних функцій в СУБД Access:

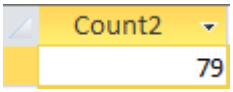
Знак операції	Значення
COUNT()	Повертає число рядків в таблиці або стовпці
SUM()	Повертає суму значень в стовпці
MIN()	Повертає найменше значення в стовпці
MAX()	Повертає найбільше значення в стовпці
AVG()	Повертає середнє значення в стовпці

Приклади використання функції COUNT():

SELECT COUNT(*) AS Count1 FROM Sumproduct - повертає кількість усіх рядків в таблиці



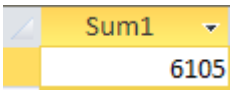
SELECT COUNT(Product) AS Count2 FROM Sumproduct - повертає кількість усіх непорожніх рядків в полі Product



Ми навмисно видалили одне значення в стовпці Product, щоб показати різницю в роботі двох запитів.

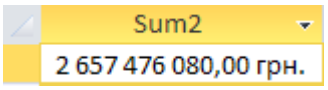
Приклади використання функції SUM():

SELECT SUM(Quantity) AS Sum1 FROM Sumproduct WHERE Month = 'April'



Даним запитом ми відобразили загальну кількість проданого товару в квітні місяці.

```
SELECT SUM(Quantity*Amount) AS Sum2 FROM Sumproduct
```



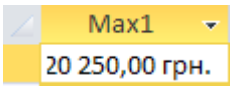
Як бачимо, в статистичних функціях ми також можемо здійснювати обчислення над кількома стовпцями з використанням стандартних математичних операторів.

Приклад використання функції MIN():

```
SELECT MIN(Amount) AS Min1 FROM Sumproduct
```

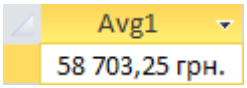
Приклад використання функції MAX():

```
SELECT MAX(Amount) AS Max1 FROM Sumproduct
```



Приклад використання функції AVG():

```
SELECT AVG(Amount) AS Avg1 FROM Sumproduct
```



6.2.7 Групування даних (GROUP BY)

Групування даних дозволяє розділити всі дані на логічні набори, завдяки чому стає можливим виконання статистичних обчислень окремо в кожній групі.

1. Створення груп (GROUP BY)

Групи створюються за допомогою речення GROUP BY оператора SELECT. Розглянемо на прикладі.

```
SELECT Product, SUM(Quantity) AS Product_num FROM Sumproduct GROUP BY
Product
```

Product	Product_num
Bikes	3450
Skates	4376
Skis Long	5251
Skis Short	2879
Snow Board	3510

Даним запитом ми витягнули інформацію щодо кількості реалізованої продукції в кожному місяці. Оператор SELECT наказує вивести два стовпця Product - назва продукту та Product_num - розрахункове поле, яке ми створили для відображення кількості реалізованої продукції (формула поля SUM(Quantity)). Речення GROUP BY вказує СУБД згрупувати дані по стовпцю Product. Варто також зазначити, що GROUP BY має іти після речення WHERE та перед ORDER BY.

2. Фільтруючі групи (HAVING)

Так само, як ми фільтрували рядки в таблиці, ми можемо здійснювати фільтрацію по згрупованим даним. Для цього в SQL існує оператор HAVING. Візьмемо попередній приклад та додамо фільтрацію по групам.

```
SELECT Product, SUM(Quantity) AS Product_num FROM Sumproduct GROUP BY
Product HAVING SUM(Quantity)>4000
```

Product	Product_num
Skates	4376
Skis Long	5251

Бачимо, що після того, як була порахована кількість реалізованого товару в розрізі кожного продукту, СУБД «відсікла» ті продукти, яких було реалізовано менше 4000 шт.

Як бачимо, оператор HAVING дуже подібний до оператора WHERE, проте між собою вони мають суттєву відмінність: WHERE фільтрує дані до того, як вони будуть згруповані, а HAVING - здійснює фільтрацію після групування. Таким чином, рядки, які були вилучені реченям WHERE не будуть включені в групу. Отож, оператори WHERE та HAVING можуть використовуватись в одному реченні. Розглянемо приклад:

```
SELECT Product, SUM(Quantity) AS Product_num FROM Sumproduct WHERE
Product<>'Skis Long' GROUP BY Product HAVING SUM(Quantity)>4000
```

Product	Product_num
Skates	4376

Ми до попереднього прикладу добавили оператор WHERE, де вказали товар Skis Long, що в свою чергу вплинуло на групування оператором HAVING. Як результат бачимо, що товар Skis Long не попав в перелік груп з кількістю реалізованої продукції більше 4000 шт.

3. Групування та сортування

Як і при звичайній вибірці даних, ми можемо посортувати групи після групування оператором HAVING. Для цього ми можемо використати вже знайомий нам оператор ORDER BY. В даній ситуації його застосування аналогічне попереднім прикладам. Наприклад:

```
SELECT Product, SUM(Quantity) AS Product_num FROM Sumproduct GROUP BY Product HAVING SUM(Quantity)>3000 ORDER BY SUM(Quantity)
```

або просто вкажемо номер поля по порядку, по якому хочемо посортувати:

```
SELECT Product, SUM(Quantity) AS Product_num FROM Sumproduct GROUP BY Product HAVING SUM(Quantity)>3000 ORDER BY 2
```

Product	Product_num
Bikes	3450
Snow Board	3510
Skates	4376
Skis Long	5251

Бачимо, що для сортування зведених результатів нам потрібно просто прописати речення з ORDER BY після оператора HAVING. Проте є один нюанс. СУБД Access не підтримує сортування груп за псевдонімами колонок, тобто в нашому прикладі, щоб посортувати значення, ми не зможемо в кінці запиту прописати ORDER BY Product_num.

6.2.8 Підзапити

Однак, дуже часто потрібно вибирати дані, що відповідають багатьом умовам, Для цього в SQL існують підзапити або вкладені запити, коли один оператор SELECT вкладається у інший.

1. Фільтрування за допомогою підзапитів

Таблиці баз даних, які використовуються в СУБД Access являються реляційними таблицями, тобто всі таблиці можна пов'язати між собою по спільним полям. Припустимо в нас зберігаються дані в двох різних таблицях і нам потрібно вибрати дані в одній із них, в залежності від того, які дані в іншій. Для цього створимо ще одну таблицю в нашій базі даних. Це буде, наприклад, таблиця Sellers з інформацією про постачальників:

ID	Address	City	Seller_name	Country
1	500 Park Street	Montreal	Michelle Green	Canada
2	1000 5th Avenue	San Francisco	Kim Howard	USA
3	42 Galaxy Road	New York	John Smith	USA
4	123 Main Street	Toronto	Denise L. Stephens	Canada

Тепер ми маємо дві таблиці - Sumproduct та Sellers, які мають однакове поле City. Припустимо, нам потрібно порахувати скільки товарів було продано лише в Канаді. Зробити це нам допоможуть підзапити. Отже, спочатку напишемо запит для вибірки міст, які знаходяться в Канаді:

```
SELECT City FROM Sellers WHERE Country = 'Canada'
```

City
Montreal
Toronto

Тепер передамо ці дані в наступний запит, який вибиратиме дані з таблиці Sumproduct:

```
SELECT SUM(Quantity) AS Qty_Canada FROM Sumproduct WHERE City IN ('Montreal','Toronto')
```

Qty_Canada
10222

Також ми можемо об'єднати ці два запити в один. Таким чином, один запит, який виводить дані буде головним, а другий запит, який передає вхідні дані, буде допоміжним (підзапитом). Для вкладення підзапиту використаємо конструкцію WHERE ... IN (...), про яку говорилось раніше:

```
SELECT SUM(Quantity) AS Qty_Canada FROM Sumproduct WHERE City IN (SELECT City FROM Sellers WHERE Country = 'Canada' )
```

Qty_Canada
10222

Бачимо, що ми отримали аналогічні дані, як і за допомогою двох окремих запитів. Таким же чином, ми можемо збільшувати глибину вкладеності запитів, вкладаючи підзапити скільки завгодно разів.

2. Використання підзапитів в якості розрахункових полів

Ми також можемо використовувати підзапити в ролі розрахункових полів. Відобразимо, наприклад, кількість реалізованої продукції по кожному продавцю за допомогою наступного запиту:

```
SELECT Seller_name, (SELECT SUM(Quantity)
FROM Sumproduct  WHERE Sellers.City = Sumproduct.City) AS Qty FROM Sellers
```

Seller_name	Qty
Michelle Green	5129
Kim Howard	5347
John Smith	3897
Denise L. Stephens	5093

Перший оператор SELECT відображає два стовпця - Seller_name та Qty. Поле Qty являється розрахунковим, воно формується в результаті виконання підзапиту, який взятий в круглі дужки. Цей підзапит виконується по одному разу для кожного запису в полі Seller_name та взагальному буде виконаний чотири рази, оскільки вибрано імена чотирьох продавців.

Також, в підзапиті, речення WHERE виконує функцію поєднання, оскільки за допомогою WHERE ми з'єднали дві таблиці по полю City, використавши повні назви стовпців (Таблиця.Поле).

6.2.9 Поєднання таблиць (INNER JOIN)

Найбільш потужною особливістю мови SQL є можливіть поєднувати різні таблиці в оперативній пам'яті СУБД під час виконання запитів. Об'єднання дуже часто використовуються для аналізу даних. Як правило, дані знаходяться в різних таблицях, що дозволяє їх більш ефективно зберігати (оскільки інформація не дублюється), спрощує обробку даних та дозволяє масштабувати базу даних (можливо додавати нові таблиці з додатковою інформацією). Таблиці

баз даних, які використовуються в СУБД Access являються реляційними таблицями, тобто всі таблиці можна пов'язати між собою по спільним полям.

1. Створення об'єднання таблиць

Об'єднання таблиць дуже проста процедура. Потрібно вказати всі таблиці, які будуть включені в об'єднання та «пояснити» СУБД, як вони будуть пов'язані між собою. Поєднання робиться за допомогою слова WHERE, наприклад:

```
SELECT DISTINCT Seller_name, Product FROM Sellers, Sumproduct
WHERE Sellers.City = Sumproduct.City
```

Seller_name	Product
Denise L. Stephens	Bikes
Denise L. Stephens	Skates
Denise L. Stephens	Skis Long
Denise L. Stephens	Skis Short
Denise L. Stephens	Snow Board
John Smith	Bikes
John Smith	Skates
John Smith	Skis Long
John Smith	Skis Short
John Smith	Snow Board
Kim Howard	Bikes
Kim Howard	Skates
Kim Howard	Skis Long
Kim Howard	Skis Short
Kim Howard	Snow Board
Michelle Green	Bikes
Michelle Green	Skates
Michelle Green	Skis Long
Michelle Green	Skis Short
Michelle Green	Snow Board

Поєднавши дві таблиці, ми змогли побачити які товари реалізує кожен продавець. Розглянемо код запиту детальніше, оскільки він трохи відрізняється від звичайного запиту. Оператор SELECT починається з вказанням стовпців, які ми хочемо вивести, проте ці поля знаходяться в різних таблицях, речення FROM містить дві таблиці, які ми хочемо поєднати в операторі SELECT, таблиці поєднуються за допомогою слова WHERE, яке вказує стовпці для об'єднання. Обов'язково потрібно вказувати повну назву поля (Таблиця.Поле), оскільки поле City є в обидвох таблицях.

2. Внутрішнє об'єднання

В попередньому прикладі для об'єднання таблиць ми використали слово WHERE, яке здійснює перевірку на основі еквівалентності двох таблиць. Об'єднання такого типу називається також «внутрішнім об'єднанням». Існує також і інший спосіб об'єднання таблиць, який явно вказує на тип об'єднання. Розглянемо наступний приклад:

```
SELECT DISTINCT Seller_name, Product FROM Sellers INNER JOIN Sumproduct ON
Sellers.City = Sumproduct.City
```

Seller_name	Product
Denise L. Stephens	Bikes
Denise L. Stephens	Skates
Denise L. Stephens	Skis Long
Denise L. Stephens	Skis Short
Denise L. Stephens	Snow Board
John Smith	Bikes
John Smith	Skates
John Smith	Skis Long
John Smith	Skis Short
John Smith	Snow Board
Kim Howard	Bikes
Kim Howard	Skates
Kim Howard	Skis Long
Kim Howard	Skis Short
Kim Howard	Snow Board
Michelle Green	Bikes
Michelle Green	Skates
Michelle Green	Skis Long
Michelle Green	Skis Short
Michelle Green	Snow Board

В цьому запиті замість WHERE ми використали конструкцію INNER JOIN ... ON ..., яка дала аналогічний результат. Незважаючи на те, що об'єднання з реченням WHERE є коротшим, все таки краще використовувати INNER JOIN, оскільки вона є більш гнучкою, про що буде детальніше розказано в наступних розділах.

6.2.10 Розширене поєднання таблиць (OUTER JOIN)

Раніше булт розглянуті самі прості способи об'єднання таблиць - за допомогою речень WHERE та INNER JOIN. Ці об'єднання називаються внутрішніми об'єднаннями або об'єднаннями по еквівалентності. Проте SQL має у своєму арсеналі набагато більше можливостей поєднати таблиці, а саме

існують також й інші види об'єднань: зовнішні об'єднання, природні об'єднання та самооб'єднання. Але для початку розглянемо, яким чином ми можемо присвоювати таблицям псевдоніми, оскільки, в подальшому, ми будемо змушені використовувати повні назви полів (Таблиця.Поле), якими без скорочень буде дуже важко оперувати через їх велику довжину.

1. Використання псевдонімів таблиць

В попередньому розділі ми дізналися, як можна використовувати псевдоніми для посилання на певні поля таблиці або на розрахункові поля. SQL так само надає нам можливість використовувати псевдоніми замість імен таблиць. Це надає нам такі переваги, як більш короткий синтаксис SQL та дозволяє багато раз використовувати одну й ту ж таблицю в операторі SELECT.

```
SELECT Seller_name, SUM(Amount) AS Sum1
FROM Sellers AS S, Sumproduct AS SP
WHERE S.City = SP.City
GROUP BY Seller_name
```

Seller_name	Sum1
Denise L. Stephens	966 818,00 грн.
John Smith	965 791,00 грн.
Kim Howard	1 346 110,00 грн.
Michelle Green	1 417 541,00 грн.

Ми відобразили загальну суму реалізованого товару по кожному продавцю. В нашому SQL запиті ми використали такі псевдоніми: для розрахункового поля SUM(Amount) псевдонім Sum1, для таблиці Sellers псевдонім S та для Sumproduct псевдонім SP. Зауважимо, що псевдоніми таблиць можуть бути застосовані також і в інших реченнях, як ORDER BY, GROUP BY та інших.

2. Самооб'єднання

Розглянемо приклад. Припустимо, нам потрібно дізнатися адресу продавців, які торгують в тій самій країні, що і John Smith. Для цього створимо такий запит:

```
SELECT City, Country, Seller_name
FROM Sellers
```

WHERE Country = (SELECT Country FROM Sellers WHERE Seller_name = 'John Smith')

Address	City	Country	Seller_name
1000 5th Avenue	San Francisco	USA	Kim Howard
42 Galaxy Road	New York	USA	John Smith

Також, цю задачу ми можемо вирішити і через самооб'єднання, прописавши наступний код:

```
SELECT S1.Address, S1.City, S1.Country, S1.Seller_name
FROM Sellers AS S1, Sellers AS S2
WHERE S1.Country = S2.Country AND S2.Seller_name = 'John Smith'
```

Address	City	Country	Seller_name
42 Galaxy Road	New York	USA	John Smith
1000 5th Avenue	San Francisco	USA	Kim Howard

Для вирішення цієї задачі використовувалися псевдоніми. Перший раз для таблиці Sellers присвоїли псевдонім S1, другий раз - псевдонім S2. Після цього ці псевдоніми можна застосовувати в якості імен таблиць. В операторі WHERE ми до назви кожного поля додаємо префікс S1, для того, щоб СУБД розуміла поля якої таблиці потрібно виводити (оскільки ми з однієї таблиці зробили дві віртуальні). Речення WHERE спочатку об'єднує таблиці, а потім фільтрує дані другої таблиці по полю Seller_name, щоби повернути лише необхідні значення.

Самооб'єднання часто використовують для заміни підзапитів, які вибирають дані з тієї ж таблиці, що і зовнішній оператор SELECT. Хоча кінцевий результат виходить тим самим, багато СУБД опрацьовують об'єднання набагато швидше ніж підзапити. Варто поекспериментувати, щоби визначити, який запит працює швидше.

3. Природнє об'єднання

Природнє об'єднання - це об'єднання, в якому ви вибираєте тільки ті стовпці, які не повторюються. Зазвичай це робиться за допомогою запису (SELECT *) для однієї таблиці і вказанням переліку полів - для решти таблиць. Наприклад:

```
SELECT SP.*, S.Country
FROM Sumproduct AS SP, Sellers AS S
WHERE SP.City = S.City
```

В цьому прикладі метасимвол (*) використовується тільки для першої таблиці. Всі решта стовпці вказані явно, тому дублікати стовпців не вибираються.

4. Зовнішнє об'єднання (OUTER JOIN)

Зазвичай, при об'єднанні зв'язують рядки однієї таблиці з відповідними рядками іншої, проте в деяких випадках може знадобитися включати в результат рядки, які не мають пов'язаних рядків в іншій таблиці (тобто вибираються геть усі рядки з однієї таблиці та додаються лише пов'язані рядки з іншої). Об'єднання такого типу називається зовнішнім. Для цього використовуються ключові слова OUTER JOIN ... ON ... з приставкою LEFT або RIGHT. Розглянемо приклад, попередньо добавивши в таблицю Sellers нового продавця - Semuel Piter, який ще немає продаж:

```
SELECT Seller_name, SUM(Quantity) AS Qty
FROM Sellers LEFT OUTER JOIN Sumproduct ON Sellers.City=Sumproduct.City
GROUP BY Seller_name
```

Seller_name	Qty
Denise L. Stephens	5093
John Smith	3897
Kim Howard	5347
Michelle Green	5129
Semuel Piter	

Даним запитом ми витягнули перелік усіх продавців в базі та підрахували для них загальну кількість проданого товару за усі місяці. Бачимо що по новому продавцю Semuel Piter відсутні продажі. Якби ми використали внутрішнє об'єднання, то нового продавця ми б не побачили, оскільки він немає записів в таблиці Sumproduct. Ми також можемо змінювати напрямок поєднання не лише прописуючи LEFT або RIGHT, але й просто змінюючи порядок таблиць (тобто такі два записи будуть давати однаковий результат: Sellers LEFT OUTER JOIN Sumproduct та Sumproduct RIGHT OUTER JOIN Sellers).

Також деякі СУБД дозволяють здійснювати зовнішнє об'єднання за спрщеним записом, використовуючи знаки *= та =*, що відповідає LEFT OUTER JOIN та RIGHT OUTER JOIN відповідно. Таким чином попередній запит можна було б переписати так:

```
SELECT Seller_name, SUM(Quantity) AS Qty
FROM Sellers, Sumproduct
WHERE Sellers.City *= Sumproduct.City
```

Нажаль Access не підтримує скорочений запис для зовнішнього об'єднання.

5. Повне зовнішнє об'єднання (FULL OUTER JOIN)

Також існує й інший тип зовнішнього об'єднання - повне зовнішнє об'єднання, яке відображає усі рядки з обидвох таблиць та пов'язує лише ті, які можуть бути пов'язані. Синтаксис повного зовнішнього об'єднання наступний:

```
SELECT Seller_name, Product
FROM Sellers FULL OUTER JOIN Sumproduct ON Sellers.City=Sumproduct.City
```

Знову ж, повне зовнішнє об'єднання не підтримують такі СУБД: Access, MySQL, SQL Server та Sybase. Як обійти цю несправедливість, ми розглянемо в наступному розділі.

6.2.11 Комбіновані запити (UNION)

В більшості SQL-запитів використовується один оператор, за допомогою якого повертаються дані із однієї або кількох таблиць. SQL також дозволяє виконувати одночасно кілька окремих запитів та відображати результат у вигляді єдиного набору даних. Такі комбіновані запити зазвичай називають поєднаннями або складними запитами.

1. Використання оператора UNION

Запити в мові SQL комбінуються за допомогою оператора UNION. Для цього необхідно вказати кожен запит SELECT та розмістити між ними ключове слово UNION. Обмежень щодо кількості використанньо оператора UNION в одному загальному запиті немає. В попередньому розділі ми зазначали, що Access не має можливості створювати повне зовнішнє об'єднання, тепер ми подивимось, як можна цього досягнути через оператор UNION.

```
SELECT *
FROM Sumproduct LEFT JOIN Sellers ON Sumproduct.City = Sellers.City
UNION
SELECT *
FROM Sumproduct RIGHT JOIN Sellers ON Sumproduct.City = Sellers.City
```

Sumproduct.ID	Month	Product	Sumproduct	Quantity	Amount	Sellers.ID	Address	Sellers.City	Seller_name	Country
						5	4th Avenue	Ottawa	Semuel Piter	Canada
2	April	Bikes	Montreal	12	4 500,00 грн.	1	500 Park Street	Montreal	Michelle Gree	Canada
3	April	Bikes	Montreal	56	21 000,00 грн.	1	500 Park Street	Montreal	Michelle Gree	Canada
4	April	Bikes	San Francisco	854	320 250,00 грн.	2	1000 5th Avenue	San Francisco	Kim Howard	USA
5	April	Bikes	New York	25	9 375,00 грн.	3	42 Galaxy Road	New York	John Smith	USA
6	April	Skates	Montreal	56	5 544,00 грн.	1	500 Park Street	Montreal	Michelle Gree	Canada
7	April	Skates	Toronto	854	84 546,00 грн.	4	123 Main Street	Toronto	Denise L. Stepl	Canada
8	April	Skates	San Francisco	25	2 475,00 грн.	2	1000 5th Avenue	San Francisco	Kim Howard	USA
9	April	Skates	New York	663	65 637,00 грн.	3	42 Galaxy Road	New York	John Smith	USA
10	April	Skis Long	Montreal	854	209 230,00 грн.	1	500 Park Street	Montreal	Michelle Gree	Canada

Бачимо, що запит відобразив як всі колонки з першої таблиці - так і з другої, незалежно від того, чи всі записи мають відповідники у іншій таблиці.

Також варто зазначити, що в багатьох випадках замість UNION ми можемо використовувати речення WHERE з багатьма умовами, та отримувати аналогічний результат. Проте через UNION записи виглядають більш лаконічними та зрозумілими. Також необхідно дотримуватись певних правил при написанні комбінованих запитів:

- запит UNION повинен включати два і більше операторів SELECT, відділених між собою ключовим словом UNION (тобто якщо в запиті використовується чотири оператора SELECT, то повинно бути три ключових слова UNION);
- кожен запит в операторі UNION повинен мати одні й ті ж стовпці, вирази чи статистичні функції, які, до того ж, мають бути перераховані в однаковому порядку;
- типи даних стовпців мають бути сумісними. Вони не обов'язково мають бути одного типу, проте мають мати подібний тип, щоби СУБД могла їх однозначно перетворити (наприклад, це можуть бути різні числові типи даних або різні типи дати).

2. Включення або виключення повторюваних рядків

Запит з UNION автоматично видаляє усі повторювані рядки з набору результатів запиту (іншими словами, веде себе як речення WHERE з кількома

умовами в одному операторі SELECT). Така поведінка оператора UNION по замовчуванню, але при бажанні ми можемо змінити це. Для цього нам варто використовувати оператор UNION ALL замість UNION.

3. Сортування результатів комбінованих запитів

Результати виконання оператора SELECT сортуються за допомогою речення ORDER BY. При комбінуванні запитів за допомогою UNION тільки одне речення ORDER BY може бути використане, і воно має бути проставлене в останньому операторі SELECT. Дійсно, на практиці немає особливого змісту частину результатів сортувати в одному порядку, а іншу частину - в іншому. Тому кілька речень ORDER BY застосовувати не дозволяється.

6.2.12 Додавання даних (INSERT INTO)

В попередніх розділах ми розглядали роботу з отримання даних із заздалегідь створених таблиць. Тепер вже час розібрати, яким же чином ми можемо створювати/видаляти таблиці, добавляти нові записи та видаляти старі. Для цих цілей в SQL існують такі оператори, як: CREATE - створює таблицю, ALTER - змінює структуру таблиці, DROP - видаляє таблицю або поле, INSERT - добавляє дані в таблицю. Почнемо знайомство з даною групою операторів з оператора INSERT.

1. Додавання цілих рядків

Як видно з назви, оператор ввв використовується для вставлення (додавання) рядків в таблицю бази даних. Додавання можна здійснити кількома шляхами:

- добавити один повний рядок;
- добавити частину нового рядка;
- добавити результати запиту.

Отже, щоби добавити новий рядок в таблицю, нам потрібно вказати назву таблиці, перелічити назви колонок та вказати значення для кожної колонки за допомогою конструкції INSERT INTO назва_таблиці (поле1, поле2 ...) VALUES (значення1, значення2 ...). Розглянемо на прикладі.

INSERT INTO Sellers (ID, Address, City, Seller_name, Country) VALUES ('6', '1st Street', 'Los Angeles', 'Harry Monroe', 'USA')

ID	Address	City	Seller_name	Country
1	500 Park Street	Montreal	Michelle Green	Canada
2	1000 5th Avenue	San Francisco	Kim Howard	USA
3	42 Galaxy Road	New York	John Smith	USA
4	123 Main Street	Toronto	Denise L. Stephens	Canada
5	4th Avenue	Ottawa	Semuel Piter	Canada
6	1st Street	Los Angeles	Harry Monroe	USA

Також можна змінювати порядок вказання назв колонок, проте одночасно потрібно змінювати і порядок значень в параметрі VALUES.

2. Додавання частини рядків

В попередньому прикладі при використанні оператора INSERT ми явно зазначали імена стовпців таблиці. Використовуючи даний синтаксис, ми можемо пропустити деякі стовпці. Це значить, що ви вводите значення для одних стовпців та не пропонуєте їх для інших. Наприклад:

INSERT INTO Sellers (ID, City, Seller_name) VALUES ('6', 'Los Angeles', 'Harry Monroe')

ID	Address	City	Seller_name	Country
1	500 Park Street	Montreal	Michelle Green	Canada
2	1000 5th Avenue	San Francisco	Kim Howard	USA
3	42 Galaxy Road	New York	John Smith	USA
4	123 Main Street	Toronto	Denise L. Stephens	Canada
5	4th Avenue	Ottawa	Semuel Piter	Canada
6		Los Angeles	Harry Monroe	

В даному прикладі ми не вказали значення для двох стовпців Address та Country. Ви можете виключати деякі стовпці із оператора ааа, якщо це дозволяє робити визначення таблиці. В цьому випадку має дотримуватися одна з умов: цей стовпець визначений як такий, що допускає значення null (відсутність будь-якого значення) або у визначення таблиці вказане значення по замовчуванню. Це означає, що, якщо не вказане ніяке значення, буде використане значення по замовчуванню. Якщо ви пропускаєте стовпець таблиці, яка не допускає появи у своїх рядках значень NULL і не має значення, визначеного для використання по замовчуванню, СУБД видасть повідомлення про помилку, і цей рядок не буде добавлений.

3. Додавання відібраних даних

В попередній прикладах ми вставляли дані в таблиці, прописуючи їх вручну в запиті. Проте оператор INSERT INTO дозволяє автоматизувати цей процес, якщо ми хочемо вставляти дані з іншої таблиці. Для цього в SQL існує така кострукція як INSERT INTO ... SELECT ... . Дана конструкція дозволяє одночасно вибирати дані з однієї таблиці, та вставити їх в іншу. Припустимо ми маємо ще одну таблицю Sellers_EU з переліком продавців нашого товару в Європі і нам потрібно їх добавити в загальну таблицю Sellers. Структура цих таблиць однакова (та ж сама кількість колонок та ті ж самі їх назви), проте інші дані. Для цього ми можемо прописати наступний запит:

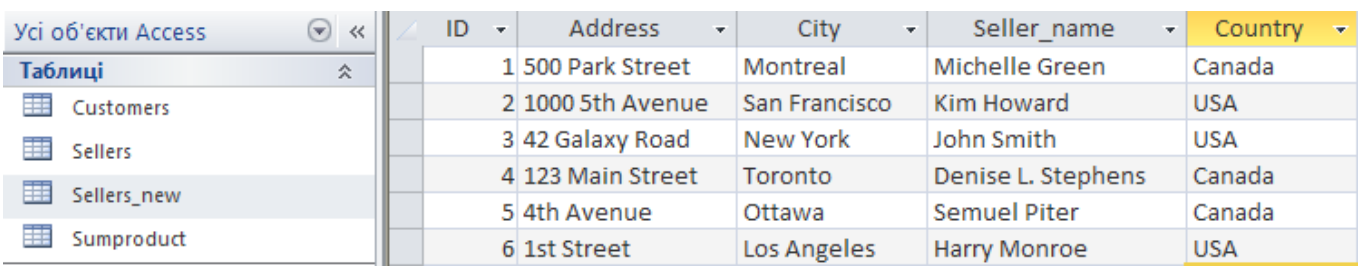
```
INSERT INTO Sellers (ID, Address, City, Seller_name, Country) SELECT ID, Address,
City, Seller_name, Country FROM Sellers_EU
```

Потрібно звернути увагу, щоби значення внутрішніх ключів не повторювалися (поле ID), в інакшому випадку станеться помилка. Оператор SELECT також може включати речення WHERE для фільтрування даних. Також слід відмітити, що СУБД не звертає уваги на назви колонок, які містяться в операторі SELECT, для неї важливо лише порядок їх розташування. Тому дані в першому вказаному стовпці, що були вибрані через SELECT, будуть в будь-якому разі заповнені в перший стовпець таблиці Sellers, що вказана після оператора INSERT INTO, незалежно від назви поля.

4. Копіювання даних з однієї таблиці в іншу

Часто при роботі з базами даних виникає необхідність у створені копій будь-яких таблиць, з метою резервування або модифікації. Щоби зробити повну копію таблиці в SQL передбачений окремий оператор SELECT INTO. Наприклад нам потрібно створити копію таблиці Sellers, потрібно буде прописати запит наступним чином:

```
SELECT * INTO Sellers_new FROM Sellers
```



ID	Address	City	Seller_name	Country
1	500 Park Street	Montreal	Michelle Green	Canada
2	1000 5th Avenue	San Francisco	Kim Howard	USA
3	42 Galaxy Road	New York	John Smith	USA
4	123 Main Street	Toronto	Denise L. Stephens	Canada
5	4th Avenue	Ottawa	Semuel Piter	Canada
6	1st Street	Los Angeles	Harry Monroe	USA

На відміну від попередньої конструкції INSERT INTO ... SELECT ..., коли дані добавляютья в існуючу таблицю, конструкція SELECT ... INTO ... FROM ... копіює дані в нову таблицю. Також можна сказати, що перша конструкція імпортує дані, а друга - експортує. При використанні конструкції SELECT ... INTO ... FROM ... слід враховувати наступне:

- можна використовувати будь-які речення в операторі SELECT, такі як GROUP BY та HAVING;

- для додавання даних з декількох таблиць можна використовувати об'єднання;

- дані можливо добавити лише в одну таблицю, незалежно від того, зі скількох таблиць вони були взяті.

6.2.13 Створення таблиць (CREATE TABLE)

Мова SQL використовується не лише для обробки інформації, але й призначена для виконання усіх операцій з базами даних і таблицями, включаючи також створення таблиць та робота з ними. Існує два способи створення таблиць: 1) більшість СУБД володіють візуальним інтерфейсом для интерактивого створення таблиць та управління ними; 2) таблицями також можна маніпулювати, використовуючи оператори SQL. Варто зазначити, що, коли ви використовуєте інтерактивний інструментарій СУБД, насправді уся робота виконується операторами SQL, тобто інтерфе йс сам створює ці команди незамітно для користувача (це подібно на запис макроса в Excel, коли макрорекодер записує ваші дії та перетворює їх в команди VBA).

1. Створення таблиць

Для створення таблиць програмним способом використовують оператор CREATE TABLE. Для цього потрібно вказати наступні дані:

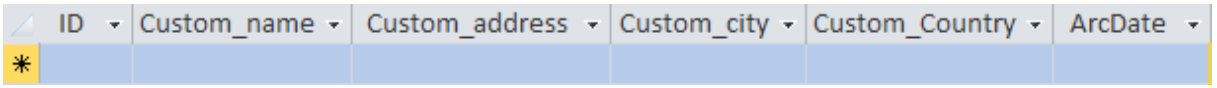
- ім'я таблиці, яке вказується після ключового слова CREATE TABLE;

- імена та визначення стовпців таблиці, що відділені комами;

- в деяких СУБД також вимагається, щоби було вказано місце розташування таблиці.

Давайте створимо нову таблицю та назвемо її Customers:

```
CREATE TABLE Customers (
ID CHAR(10) NOT NULL Primary key,
Custom_name CHAR(25) NOT NULL,
Custom_address CHAR(25) NULL,
Custom_city CHAR(25) NULL,
Custom_Country CHAR(25) NULL,
ArcDate CHAR(25) NOT NULL, DEFAULT NOWO)
```



Так ми спочатку вказуємо назву нової таблиці, потім в дужках перераховуємо стовпці, які будемо створювати, причому їх назви не можуть повторюватися в межах однієї таблиці. Після назв стовпців вказується тип даних для кожного поля (CHAR(10)), потім зазначаємо чи може поле містити порожні значення (NULL або NOT NULL), а також потрібно вказати поле, яке буде первинним ключем (Primary key).

Мова SQL також дозволяє визначати для кожного поля значення по замовчуванню, тобто, якщо користувач не вкаже значення для певного поля - воно буде автоматично проставлене СУБД. Значення по замовчуванню визначається ключовим словом DEFAULT при визначенні стовпців оператором CREATE TABLE.

2. Оновлення таблиць

Для того, щоб змінити таблицю в SQL використовується оператор ALTER TABLE. При використанні даного оператора слід ввести наступну інформацію:

- ім'я таблиці, яку ми хочемо змінити;

- перелік змін, які ми хочемо зробити.

Для прикладу давайте додамо нову колонку в таблицю Sellers, в якій будемо зазначати телефон реалізатора:

```
ALTER TABLE Sellers ADD Phone CHAR (20)
```

ID	Address	City	Seller_name	Country	Phone
1	500 Park Street	Montreal	Michelle Green	Canada	
2	1000 5th Avenue	San Francisco	Kim Howard	USA	
3	42 Galaxy Road	New York	John Smith	USA	
4	123 Main Street	Toronto	Denise L. Stephens	Canada	
5	4th Avenue	Ottawa	Semuel Piter	Canada	
6	1st Street	Los Angeles	Harry Monroe	USA	

Крім додавання стовпців, ми також можемо їх видаляти. Давайте тепер видалимо поле Phone. Для цього пропишемо наступний запит:

```
ALTER TABLE Sellers DROP COLUMN Phone
```

ID	Address	City	Seller_name	Country
1	500 Park Street	Montreal	Michelle Green	Canada
2	1000 5th Avenue	San Francisco	Kim Howard	USA
3	42 Galaxy Road	New York	John Smith	USA
4	123 Main Street	Toronto	Denise L. Stephens	Canada
5	4th Avenue	Ottawa	Semuel Piter	Canada
6	1st Street	Los Angeles	Harry Monroe	USA

3. Видалення таблиць

Видалення таблиць здійснюється за допомогою оператора DROP TABLE. Щоби видалити таблицю Sellers_new, ми можемо прописати наступний запит:

```
DROP TABLE Sellers_new
```

В багатьох СУБД застосовуються правила, що запобігають видаленню таблиць, які є вже пов'язані з іншими таблицями. Якщо ці правила діють і ви видаляєте таку таблицю, то СУБД блокує операцію видалення до тих пір, поки не буде видалений зв'язок. Такі заходи запобігають випадковому видаленню потрібних таблиць.

7. **ОГЛЯД ТИПІВ СУЧАСНИХ БАЗ ДАНИХ**

Понад тридцять років для вирішення задач організаційно-економічного управління на рівні підприємств, установ і організацій використовувались реляційні моделі бази даних.

Проте на сьогодення реляційні моделі та СКБД, що їх підтримують, не завжди задовольняють всі проблеми цифрової економіки. Реляційні бази даних є вузьким місцем WEB-проектів. Основними проблемами використання реляційних баз даних у WEB-технологіях є наступні: потреба опрацювання надвеликих обсягів даних (Big Data), які часто представлені у неструктурованому чи слабоструктурованому форматі, для яких властиві часті зміни і велика кількість користувачів; забезпечення горизонтального масштабування та робота з кластерами; необхідність підтримки агрегатів. Реляційні моделі не підтримують агрегатні дані, тому іноді їх називають безагрегатними (aggregate-ignore).

Вказані недоліки реляційних баз даних та пошуки їх усунення привели до нового напряму організації баз даних, який дістав загальну назву NoSQL. NoSQL — це нова технологія, яка охоплює ряд підходів до створення баз даних, відмінних від традиційних реляційних моделей. Слід зауважити, що термін NoSQL росшифровується як Not Only SQL — «не лише SQL», тобто це напрям розвитку баз даних не проти SQL, а за те, щоб використовувати нереляційні бази даних в тих задач, де не можливе табличне представлення даних.

Промовистими даними про розвиток NoSQL технології є рейтинг СКБД, який опублікований інформаційним виданням DB-Engines. Згідно цього рейтингу станом на лютий 2022 року в десятку лідерів поряд з такими відомими і поширеними реляційними СКБД, як Oracle, MySQL і Microsoft SQL Server, до першої десятки потрапили також системи NoSQL: СКБД MongoDB (5 місце в рейтингу) та СКБД Redis і Elasticsearch, які займають 9 і 10 місця відповідно. Всього в першій півсотні рейтингу налічується 20 систем NoSQL, що свідчить про їх стрімкий розвиток і популярність.

Розглянемо основні типи баз даних, які використовуються для організації даних в системі управління базами даних (СУБД). Вибір типу залежить від того, які операції зможе виконувати додаток та як будуть представлені дані.

7.1 Найпростіші типи баз даних

Почнемо з найпростіших типів БД, які все ще можуть зустрічатися в спеціалізованих середовищах, але в основному замінені надійними і продуктивними альтернативами.

Прості структури даних

Перший і найпростіший спосіб зберігання даних – текстові файли. Метод застосовується і сьогодні для роботи з невеликими обсягами інформації. Для поділу полів використовується спеціальний символ: кома або крапка з комою в csv-файлах датасета, двокрапка або пробіл в * nix-подібних системах:

/ Etc / passwd в * nix системі

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing
List                                Manager:/var/list:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
syslog:x:102:106::/home/syslog:/usr/sbin/nologin      bob:x:1000:1000:Bob
Smith,,,:/home/bob:/bin/bash
```

Рисунок 7.1 – Приклад текстового файлу

Властивості даного типу**:**

- обмежений тип і рівень складності інформації, що зберігається;
- важко встановити зв'язки між компонентами даних;
- відсутність функцій паралелізму;
- практичні тільки для систем з невеликими вимогами до читання і запису;

- використовуються для зберігання конфігураційних даних;
- немає необхідності в сторонньому програмному забезпеченні.

Приклади**:** /etc/passwdi /etc/fstab в * nix-системах, csv-файли.

Ієрархічні бази даних

На відміну від текстових таблиць, в наступному типі БД з'являються зв'язки між об'єктами. В ієрархічних базах даних кожен запис має одного «батька». Це створює деревоподібну структуру, в якій записи класифікуються за їхніми стосункам з ланцюжком батьківських записів.

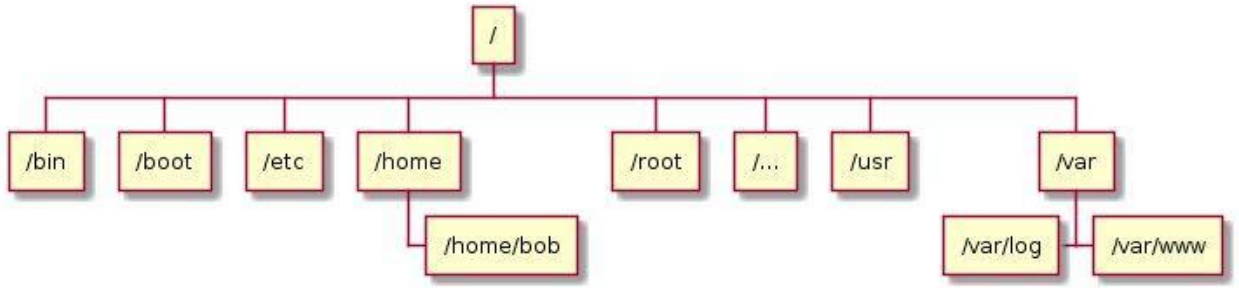


Рисунок 7.2 – Приклад побудови ієрархічних зв'язків

Властивості ієрархічних баз даних:

- інформація організована у вигляді дерева з відносинами «предок-нащадок»;
- кожен запис може мати не більше одного з батьків;
- зв'язки між записами виконані у вигляді фізичних покажчиків;
- неможливо реалізувати відносини «багатьох до багатьох».

Приклади: файлові системи, DNS, LDAP.

Мережеві бази даних

Мережеві бази даних розширюють функціональність ієрархічних: записи можуть мати більше одного батька. А значить, можна моделювати складні відносини.

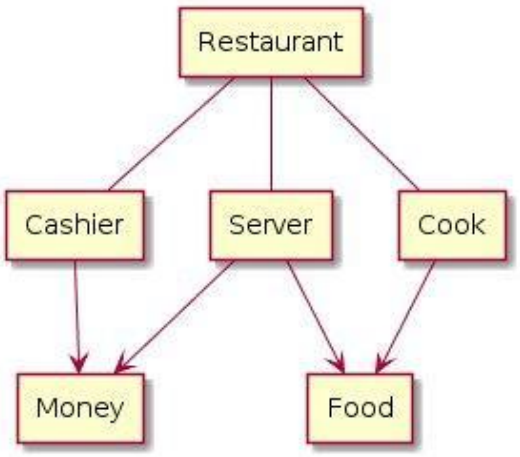


Рисунок 7.3 – Приклад зв'язків в мережевій базі даних

Властивості мережевих баз даних**:**

- мережеві бази даних подаються не деревом, а загальним графом;
- обмежені тими ж шаблонами доступу, що ієрархічні БД.

Приклади бази**:** IDMS.

7.2 Реляційні БД

SQL бази даних

Реляційні бази даних – найстаріший тип досі широко використовуваних БД загального призначення. Дані та зв'язки між даними організовані за допомогою таблиць. Кожен стовпець у таблиці має ім'я і тип. Кожен рядок представляє окремий запис або елемент даних в таблиці, який містить значення для кожного з стовпців.

Властивості**:**

- поле в таблиці, зване зовнішнім ключем, може містити посилання на стовпці в інших таблицях, що дозволяє їх з'єднувати;
- високоорганізована структура і гнучкість робить реляційні БД потужними і такими, що адаптуються до різних типів даних;
- для доступу до даних використовується мова структурованих запитів (SQL);
- надійний вибір для багатьох додатків.

Приклади баз: MySQL, MariaDB, PostgreSQL, SQLite.

7.3 NoSQL бази даних

NoSQL – група типів БД, що пропонують підходи, відмінні від стандартного реляційного шаблону. Говорячи NoSQL, мають на увазі або «не-SQL», або «не тільки SQL», щоб уточнити, що іноді допускається SQL-подібний запит.

Бази даних «ключ-значення»

У базах даних «ключ-значення» для зберігання інформації ви предоставляте ключ і об'єкт даних, який потрібно зберегти. Наприклад, JSON-об'єкт, зображення або текст. Щоб зробити запит, відправляєте ключ і отримуєте blob-об'єкт .

Рисунок 7.4 – Приклад бази даних «ключ-значення»

Властивості**:**

- сховища забезпечують швидкий доступ з незначними витратами;
- часто зберігають дані конфігурацій і інформацію про стан даних, представлених словниками або хешем;
- немає жорсткої схеми відносини між даними, тому в таких БД часто зберігають одночасно різні типи даних;
- розробник відповідає за визначення схеми іменування ключів і за те, щоб значення мало відповідний тип / формат.

Приклади: Redis, memcached, etcd.

Документні бази даних

Документні бази даних (також документоорієнтовані БД або сховища документів), спільно використовують базову семантику доступу і пошуку сховищ ключів і значень. Такі БД також використовують ключ для унікальної

ідентифікації даних. Різниця між сховищами «ключ-значення» і документними БД полягає в тому, що замість зберігання blob-об'єктів, документоорієнтовані бази зберігають дані в структурованих форматах – JSON, BSON або XML.

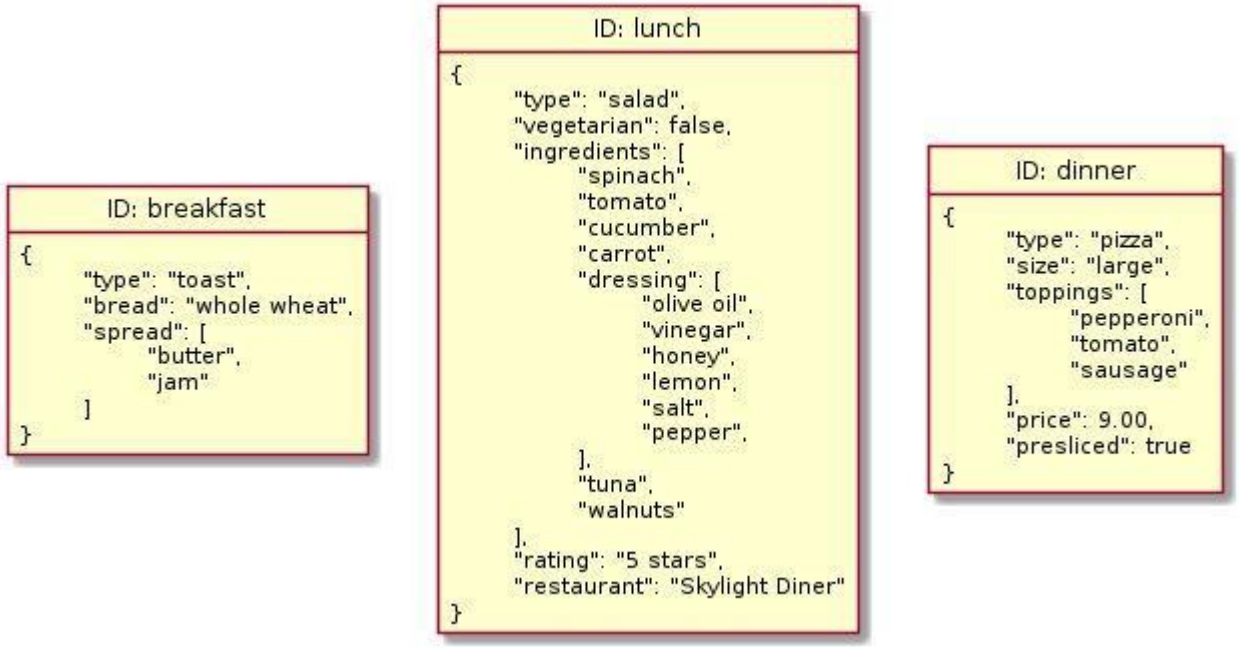


Рисунок 7.5 – Приклад документної бази даних

Властивості**:**

- база даних не виділяє окремий формат або схему;
- кожен документ може мати свою внутрішню структуру;
- документні БД є хорошим вибором для швидкої розробки;
- в будь-який момент можна змінювати властивості даних, не змінюючи структуру або самі дані.

Приклади**:** MongoDB, RethinkDB.

Графові бази даних

Замість зіставлення зв'язків з таблицями і зовнішніми ключами, графові бази даних встановлюють зв'язки, використовуючи вузли, ребра і властивості.

Графові бази представляють дані у вигляді окремих вузлів, які можуть мати будь-яку кількість пов'язаних з ними властивостей.

Властивості**:**

- виглядають аналогічно мережевим;

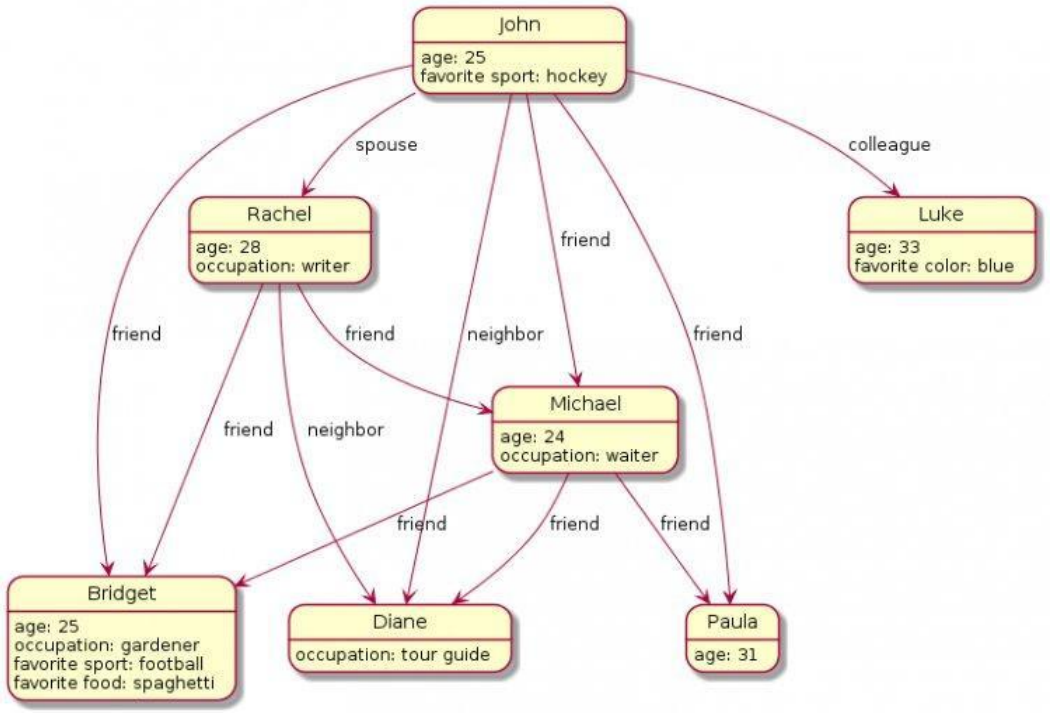


Рисунок 7.6 – Приклад графової бази

- фокусуються на зв'язках між елементами;
- явно відображають зв'язки між типами даних;
- не вимагають покрокового обходу для переміщення між елементами;
- немає обмежень в типах зв'язків.

Приклади: Neo4j, JanusGraph, Dgraph.

Стовбчикові бази даних

Стовбчикові бази даних (також нереляційні колоночні сховища або бази даних з широкими стовпцями) належать до сімейства NoSQL БД, але зовні схожі на реляційні БД. Як і реляційні, стовпчикові БД зберігають дані, використовуючи рядки і стовпці, але з іншим зв'язком між елементами.

У реляційних БД всі рядки повинні відповідати фіксованій схемою. Схема визначає, які стовпчики будуть в таблиці, типи даних та інші критерії. У стовпчикових базах замість таблиць є структури – «стовпчик сімейства». Сімейства містять рядки, кожна з яких визначає власний формат. Рядок складається з унікального ідентифікатора, що використовується для пошуку, за яким слідують набори імен та значень стовпців.

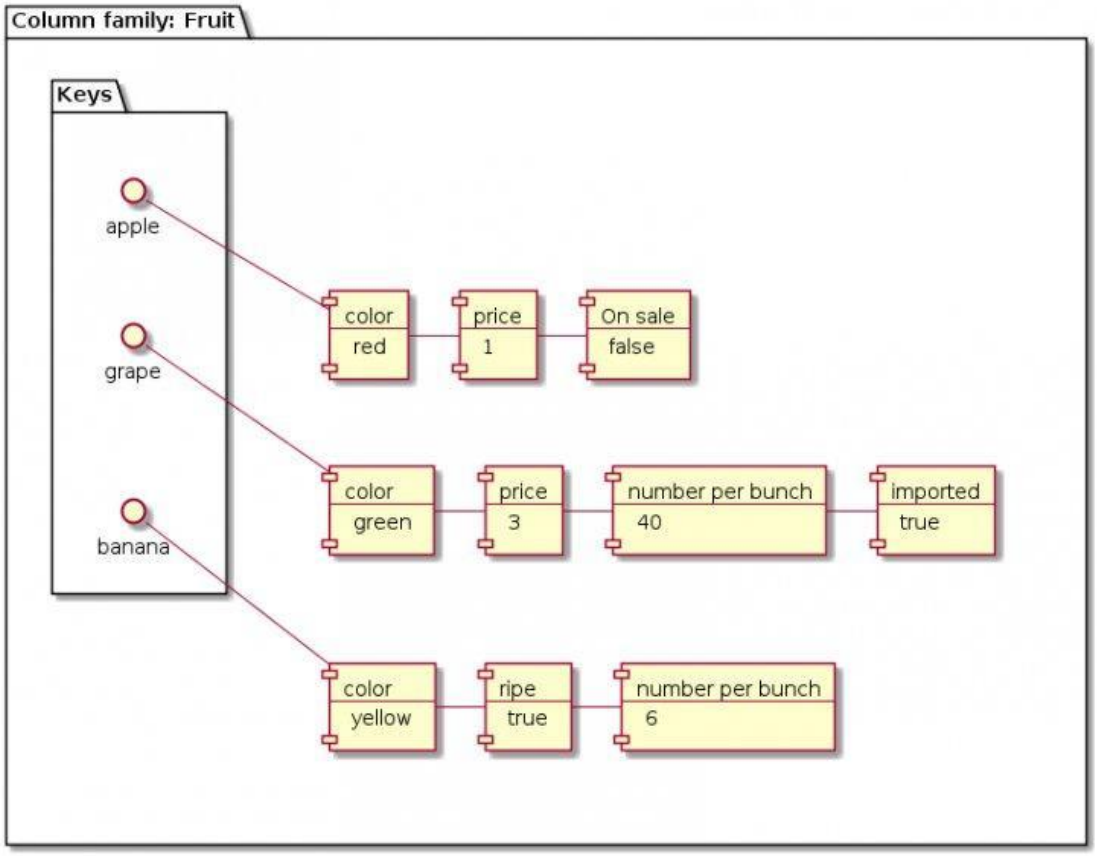


Рисунок 7.7 – Приклад стовпчикової бази

Властивості стовпчикової бази**:**

- БД зручні при роботі з додатками, що вимагають високої продуктивності;
- дані та метадані доступні по одному ідентифікатору;
- гарантоване розміщення всіх даних з рядка в одному кластері, що спрощує сегментацію і масштабування даних.

Приклади**:** Cassandra, HBase.

Бази даних часових рядів

Бази даних часових рядів створені для збору і управління елементами, що змінюються з плином часу. Більшість таких БД організовані в структури, які записують значення для одного елемента. Наприклад, можна створити таблицю для відстеження температури процесора. Усередині кожне значення буде

складатися з тимчасової мітки і показника температури. У таблиці може бути декілька метрик.

Time	CPU Temp	System Load	Memory Usage %
2019-10-31T03:48:05+00:00	37	0.85	92
2019-10-31T03:48:10+00:00	42	0.87	90
2019-10-31T03:48:15+00:00	33	0.74	87
2019-10-31T03:48:20+00:00	34	0.72	77
2019-10-31T03:48:25+00:00	40	0.88	81
2019-10-31T03:48:30+00:00	42	0.89	82
2019-10-31T03:48:35+00:00	41	0.88	82

Рисунок 7.8 – Приклад бази даних часових рядів

Властивості бази даних часових рядів**:**

- орієнтовані на запис;
- призначені для обробки постійного потоку вхідних даних;
- продуктивність залежить від кількості відслідковуваних елементів, інтервалу опитування між записом нових значень і фактичного корисного навантаження даних.

Приклади**:** OpenTSDB, Prometheus, InfluxDB, TimescaleDB.

7.4 Комбіновані типи

NewSQL і багатомодельні БД є різними типами баз даних, але вирішують одну групу проблем, викликаних полярними підходами SQL або NoSQL-стратегії. Чому б не об'єднати переваги обох груп? Розглянемо комбіновані типи.

NewSQL бази даних

NewSQL бази даних успадковують реляційну структуру і семантику, але побудовані з використанням більш сучасних, масштабованих конструкцій. Мета – забезпечити більшу масштабованість, ніж реляційні БД, і більш високі гарантії узгодженості, ніж в NoSQL. Компроміс між узгодженістю і доступністю є фундаментальною проблемою розподілених баз даних, описаних теоремою CAP.

Властивості**:**

- можливість горизонтального масштабування;

- висока доступність;
- велика продуктивність і реплікація;
- невеликий функціонал і гнучкість;
- чимале споживання ресурсів і необхідність спеціалізованих знань для роботи з базою даних.

Приклади**:** MemSQL, VoltDB, Spanner, Calvin, CockroachDB, FaunaDB, yugabyteDB.

Багатомодельні бази даних

Багатомодельні бази даних – бази, які поєднують функціональні можливості кількох видів БД. Переваги такого підходу очевидні – одна і та ж система може використовувати різні уявлення для різних типів даних.

Спільне розміщення даних з декількох типів БД в одній системі дозволяє виконувати нові операції, які в іншому випадку були б ускладнені або неможливі. Наприклад, багатомодельні бази можуть дозволити користувачам отримати доступ до даних, що зберігаються в різних типах БД, і управляти ними в рамках одного запиту, а також підтримують узгодженість даних при виконанні операцій, що змінюють інформацію відразу в декількох системах.

Властивості**:**

- допомагають зменшити навантаження на СУБД;
- дозволяють розширюватися до нових моделей у міру зміни потреб без внесення змін до базової інфраструктури;
- забезпечують безперервний доступ і простий розподіл даних;
- мають лінійну масштабованість і прості для розробки.

Приклади**:** ArangoDB, OrientDB, Couchbase.

На сьогодення не існує універсальної, інтегрованої системи, що могла б задовольнити всі вимоги цифрової економіки. Тому для задач, які потребують підтримку трансакцій, за умови їх централізованого опрацювання, підходять класичні реляційні СКБД, для надвеликих розподілених даних – системи NewSQL. Для зберігання і опрацювання розподілених, неструктурованих WEB-орієнтованих даних найбільш оптимальними є бази даних NoSQL.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Гайдаржи В., Ізварін І. Бази даних в інформаційних системах: Навчальний посібник. – Тернопіль: Навчальна книга.– 2018.– 418 с.
2. Мулеса О.Ю. Інформаційні системи та реляційні бази даних. Навч.посібник. – Електронне видання, 2018. – 118 с.
3. Корнієнко С.К. Проектування інформаційного забезпечення автоматизованих систем. Навч. Посібник. – Запоріжжя: ЗНТУ, 2015. – 210 с.
4. Карпуша В.Д., Панченко Б.Є. Моделювання та проектування реляційних баз даних : навч. посіб. – Суми : СДУ, 2010.
5. Сенів М. М., Яковина В. С. Безпека програм та даних : навч. посіб. Львів : Видавництво Львівської політехніки, 2015.
6. Завадський І.О. Основи баз даних: [Навч. посіб.] / І.О. Завадський. – К. : Видавець І.О. Завадський, 2011. – 192 с.
7. Балик Н., Мандзюк В.Бази даних MySQL: Навчальний посібник. – Тернопіль: Навчальна книга – Богдан, 2010.– 160 с.
8. Гайдаржи В.І., Дацюк О.А. Основи проектування та використання баз даних : Навч. посібник Київ : Політехніка, 2004. – 166 с.
9. Уроки SQL [Електронний ресурс] URL: http://moonexcel.com.ua/index.php
10. Рейтинг DB-Engines. [Електронний ресурс] URL: https://dbengines.com/en/ranking.
11. 11 типів сучасних баз даних: короткий опис, схеми і приклади БД [Електронний ресурс] URL: https://senior.ua/articles/11-tipv-suchasnih-baz-danih-korotkiy-opis-shemi--prikladi-bd.

О 64 Організація баз даних. Конспект лекцій для здобувачів фахової передвищої освіти освітньо-професійної програми «Комп'ютерна інженерія» галузь знань 12 Інформаційні технології спеціальності 123 Комп'ютерна інженерія денної форми навчання/ В. В. Завіша. Луцьк : ТФК ЛНТУ, 2022. 112 с.

Конспект лекцій містить змістовний матеріал з теоретичних основ організації баз даних і систем керування базами даних. Докладно на численних прикладах наведено відомості з проектування, створення та обслуговування баз даних, етапів їх проектування, організації ефективної структури зберігання даних, організації запитів до збережених даних, методів забезпечення цілісності даних, а також здобуття практичних навичок використання мови запитів SQL.

Видання призначене для здобувачів фахової передвищої освіти ТФК ЛНТУ, що навчаються на спеціальностях галузі знань 12 Інформаційні технології.

Комп'ютерний набір В. ЗАВІША

Редактор В. ЗАВІША

Підп. до друку «__»_____ 2022 р.
Формат 60х84/16. Папір офс. Гарнітура Таймс.
Ум. друк. арк. 7.0. Тираж ___ прим.

Відокремлений структурний підрозділ
«Технічний фаховий коледж
Луцького національного технічного університету»
43023 м. Луцьк, вул. Конякіна, 5