

Циклова комісія комп'ютерної інженерії

КОНСПЕКТ ЛЕКЦІЙ З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

ОРГАНІЗАЦІЯ БАЗ ДАНИХ

спеціальність

123 Комп'ютерна інженерія

освітня програма

Обслуговування комп'ютерних систем і мереж

(назва освітньої програми)

відділення

Комп'ютерних технологій та систем

(назва відділення)

Дніпро
2023 рік

Зміст

1.	Введення в бази даних. Трирівнева архітектура бази даних	2
2.	Типова організація сучасної СУБД. Функції СУБД. Архітектура багатокористувацьких СУБД	14
3.	Життєвий цикл бази даних	33
4.	Методологія проектування бази даних. Поняття моделей даних	47
5.	Побудова концептуальної моделі даних.	58
6.	Реляційна модель даних.	73
7.	Нормалізація відношень реляційної бази даних.	93
8.	Методика перетворення концептуальних структур даних в реляційні структури.	111
9.	Структура та типи даних мови SQL. Створення та видалення баз даних. Створення, видалення та модифікація таблиць. Вставка, видалення та оновлення даних.	121
10.	Оператори мови SQL	144
11.	Формування простих запитів MySQL. Робота з стовпцями та рядками таблиці БД при формування простих запитів	149
12.	Складно-складові запити. Багатотабличні запити. Природні, внутрішні й перехресні об'єднання. Прямі, ліві та праві об'єднання.	164
13.	Вбудовані функції MySQL.	178
14.	Створення підзапитів.	188
15.	Обробка транзакцій: транзакції, невдачі і відновлення, управління паралелізмом.	200
16.	Тригери: створення, використання, видалення.	214
17.	Адміністрування бази даних засобами MySQL.	221
18.	Рівні привілеїв. Створення облікових записів користувачів за допомогою GRANT й REVOKE.	231

Лекція № 1

з навчальної дисципліни Організація баз даних

Тема Введення в бази даних. Трирівнева архітектура бази даних

Мета Пояснити принцип трирівневої архітектури СУБД. Дати характеристику кожному рівню представлення бази даних.

Час – 2 години

План проведення лекції

1 Введення. Основні поняття й визначення теорії баз даних

2 Характеристика складових банку даних

3 Трирівнева архітектура бази даних

3.1 Зовнішній рівень

3.2 Концептуальний рівень

3.3 Внутрішній рівень

Література

1. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.

2. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.

3. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.

4. Ярцев В.П. Організація баз даних та знань: навчальний посібник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Введення. Основні поняття й визначення теорії баз даних

Стрижневі ідеї сучасних інформаційних технологій базуються на концепції баз даних. Відповідно до цієї концепції, основою інформаційних технологій є дані, які повинні бути організовані в бази даних з метою адекватного відображення реального миру, що змінюється, і задоволення інформаційних потреб користувачів.

Одним з найважливіших понять у теорії баз даних є поняття інформації. Під *інформацією* розуміються будь-які відомості про яку-небудь подію, процес, об'єкти. До інформації може ставитися все, що може цікавити користувача будь-якого рівня.

Дані - це інформація, представлена в певному виді, що дозволяє автоматизувати її збір, зберігання й подальшу обробку людиною або інформаційним засобом. Для комп'ютерних технологій дані - це інформація в дискретному, фіксованому виді, зручна для зберігання, обробки на ЕОМ, а також для передачі по каналах зв'язку.

База даних (БД) - іменована сукупність даних, що відбиває стан об'єктів й їхніх відносин у розглянутій предметній області, або інакше БД - це сукупність взаємозалежних даних при такій мінімальній надмірності, що допускає їхнє використання оптимальним образом для одного або декількох додатків у певній предметній області. БД складається з безлічі зв'язаних файлів.

Система керування базами даних (СУБД) - сукупність мовних і програмних засобів, призначених для створення, ведення й спільного використання БД багатьма користувачами.

Автоматизована інформаційна система (АИС) - це система, що реалізує автоматизований збір, обробку, маніпулювання даними, що функціонують на основі ЕОМ й інших технічних засобів і що включає відповідне програмне забезпечення (ПО) і персонал. Надалі в цій якості буде використатися термін інформаційна система (ИС), що має на увазі поняття автоматизована. ИС може функціонувати самостійно або служити компонентом більше складної системи.

Кожна ІС залежно від її призначення має справу з тією або іншою частиною реального миру, що прийнято називати предметною областю (Про) системи. Виявлення Про - це необхідний початковий етап розробки будь-якої ІС. Саме на цьому етапі визначаються інформаційні потреби всієї сукупності користувачів майбутньої системи, які, у свою чергу, визначають зміст її бази даних. По області застосування ІС можна розділити на системи, використовувані в утворенні, виробництві, бізнесі, науці й інших областях. У більшості випадків прибігають до розбивки всього безлічі об'єктів Про на групи об'єктів, рідних за структурою, що володіють однаковими властивостями, наприклад. У кожному фрагменті Про виділяється система об'єктів і зв'язку між об'єктами, виділяються процеси, що відбуваються в цьому фрагменті, а також кінцеве число користувачів.

Банк даних (БнД) є різновидом ІС. БнД - це система спеціальним образом організованих даних: баз даних, програмних, технічних, мовних, організаційно-методичних засобів, призначених для забезпечення централізованого нагромадження й колективного багатоцільового використання даних. Під завданнями обробки даних звичайно розуміється спеціальний клас розв'язуваних на ЕОМ завдань, пов'язаних з видом, зберіганням, сортуванням, відбором по заданій умові й угрупованням записів однорідної структури. При цьому для користувача передбачається генерація різних звітів, як правило, табличної форми.

БнД містить у собі:

- технічні засоби;
- одну або декілька БД;
- СУБД;
- словник або каталог даних;
- адміністратора;
- обчислювальну систему;
- обслуговуючий персонал.

Схематично це виглядає так, як показано на рисунку 1.

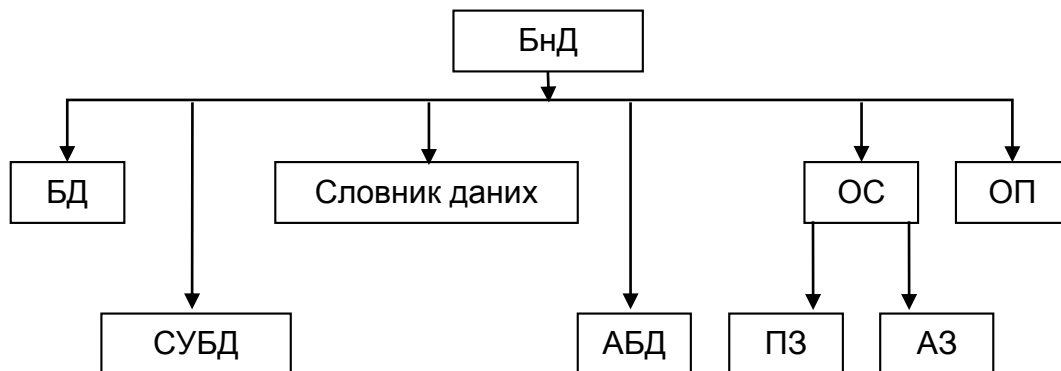


Рис. 1. Банк даних

Окремі програми або комплекс програм, що реалізують автоматизацію рішення прикладних завдань обробки даних, називаються додатками. Оскільки одні й ті самі дані можуть використовуватись для рішення багатьох завдань, то й додатків до однієї й тієї ж бази даних може бути багато. Додатки, створені засобами СУБД, відносять до додатків СУБД. Додатки, створені поза середовищем СУБД за допомогою систем програмування, що використовують засоби доступу до БД, приміром , Delphi або C++Builder, називають зовнішніми додатками. Всі додатки, що працюють із однієї й тією же базою даних, повинні функціонувати коректно, не заважати один одному й урахувати всі зміни, які вносяться іншими додатками. Така координація роботи додатків здійснюється СУБД.

2 Характеристика складових банку даних

База даних - це:

- Єдине, містке сховище різноманітних даних й описів їхніх структур, що після свого визначення, здійснюваного окремо й незалежно від додатків, використовується одночасно багатьма додатками.
- Збережені дані організовані в спільно використовуваний набір і логічно зв'язані між собою, точно також як у розглянутій відповідній предметній області взаємозалежні між собою об'єкти і явища.
- Оскільки структури даних визначаються засобами СУБД окремо від додатків і зберігаються в базі даних, то додавання нових структур даних або зміна існуючих не впливає на додатки, що не використовують змінені дані.

Система керування базами даних - це програмне забезпечення, за допомогою якого можна:

- визначати БД, структури її даних, а також задавати обмеження для збережених даних;
- маніпулювати даними, організувати виконання різноманітних не фіксованих заздалегідь запитів;
- надавати контрольований доступ до інформації бази даних;
- здійснювати підтримку забезпечення безпеки даних;
- забезпечувати цілісність даних;
- управляти багатокористувацьким режимом роботи, контролюючи процеси спільного доступу до даних;
- відновлювати інформацію бази даних, загублену в результаті різних апаратних або програмних збоїв.

Повною мірою ці концепції і їхні складові формувалися не відразу, а згодом .

Словник або каталог даних служить для централізованого нагромадження й опису ресурсу даних. Словник даних стежить за визначенням всіх елементів дані бази, включаючи структури даних на рівні груп і записів, відслідковує відношення, що існують між окремими групами даних. Наявність словника дозволяє зменшити надмірність даних. Він містить опис Про, відомості про структуру БД, про зв'язки між елементами БД, крім того, може містити відомості про коди даних, про розмежування доступу до даних. Словник даних можна розглядати як частину самої бази даних. У цьому випадку база даних є самоописуваною, оскільки вона містить інформацію про свою структуру.

Адміністратор БД (АБД) - людина або група осіб, які приймають рішення. Основні функції АБД:

- участь у розробці БД;
- контроль правильності функціонування БД.

Обчислювальна система (ОС) - включає програмні (ПЗ) і апаратні засоби (АЗ).

Обслуговуючий персонал (ОП) - це особи, прямими обов'язками яких є створення й підтримка коректного функціонування банку даних. Вони відповідальні за

роботу БнД і прикладного програмного забезпечення. До обслуговуючого персоналу ставляться: розроблювачі й адміністратори бази даних, АНАЛІТИКИ, програмісти.

Основні положення концепції, що визначає всі наступні етапи розробки СУБД, полягають у наступному:

- архітектура СУБД повинна забезпечувати розмежування користувальницького й системного рівнів;
- необхідно дати можливість кожному користувачеві мати своє, відмінне від інших Представлення про властивості збережених даних.

Отже початковою стадією проектування будь-якої конкретної ІС повинні бути абстрактні описи інформаційних потреб кожної групи користувачів, на основі яких генерується також абстрактний, але вже загальний для всієї організації опис структур збережених даних, а СУБД, за допомогою якої ця ІС буде створюватися й підтримуватися, зобов'язана мати для цього певні можливості.

Розглянемо архітектурні рішення побудови СУБД й її функціональних характеристик, а також типової організації сучасної СУБД, що дозволяє втілити ці рішення й характеристики в ефективний програмний продукт.

3 Трирівнева архітектура бази даних

Одним з найважливіших аспектів розвитку СУБД стала ідея відділення логічної структури БД і маніпуляцій даними, необхідних користувачам, від фізичного представлення, необхідного комп'ютерним устаткуванням. [1]

Одна й та сама БД, залежно від точки зору, може мати різні рівні опису. По числу рівнів опису даних, підтримуваних СУБД, розрізняють одне-, дво- і трирівневі системи. У цей час найчастіше підтримується трирівнева архітектура опису БД (див. рисунок 1.1), із трьома рівнями абстракції, на яких можна розглядати базу даних.

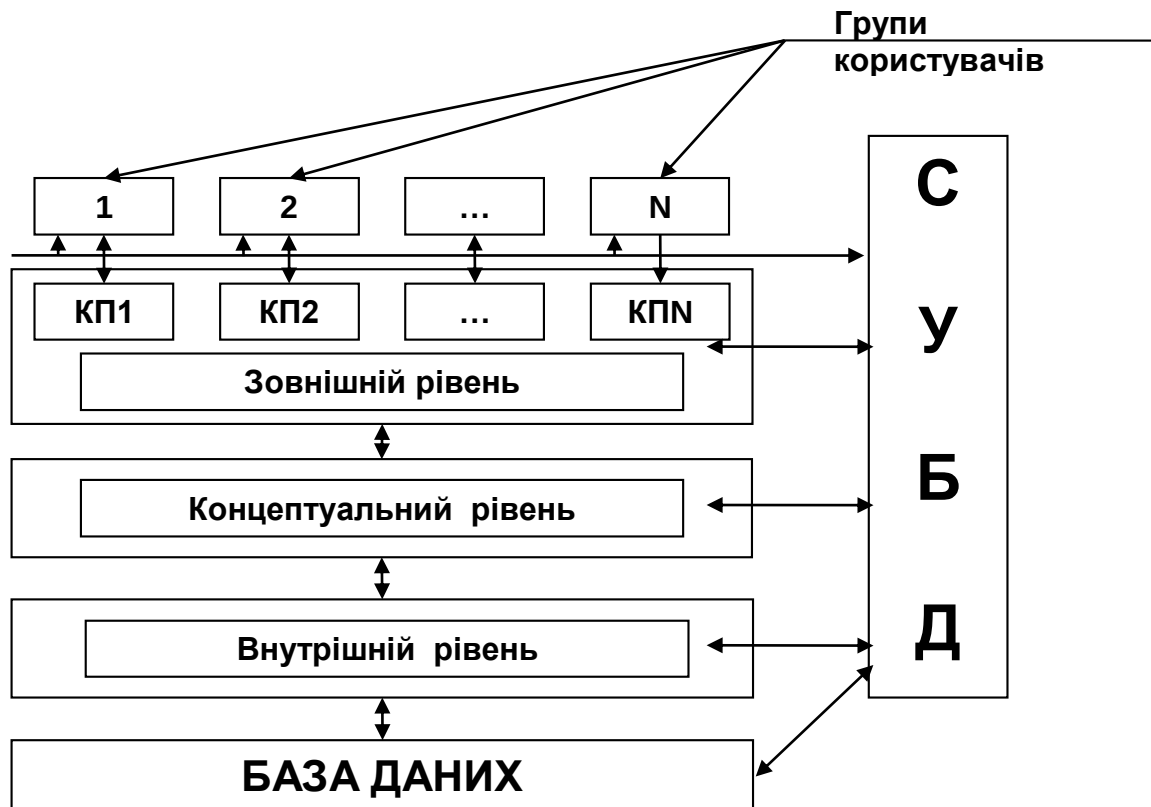


Рисунок 3.1. Трирівнева архітектура СУБД

Така архітектура включає:

зовнішній рівень, на якому користувачі сприймають дані, де окремі групи користувачів мають своє представлення (ПП) на базу даних;

внутрішній рівень, на якому СУБД й операційна система сприймають дані;

концептуальний рівень Представлення даних, призначений для відображення зовнішнього рівня на внутрішній рівень, а також для забезпечення необхідної їхньої незалежності друг від друга; він пов'язаний з узагальненим представленням користувачів.

Перші пропозиції даної архітектури надійшли в 1971 році від робочої групи CODASYL (Conference on Data Systems and Languages – Конференція по мовах і системам даних), що обґрунтувала необхідність використання дворівневого підходу, побудованого на основі виділення системного Представлення й користувацьких подань. А в 1975 році Комітет планування стандартів і норм SPARC (Standards Planning and Requirements Committee) Американського національного інституту стандартів ANSI (American National Standards Institute) запропонував узагальнену структуру систем баз даних, визнавши необхідність використання трирівневої архітектури, що і була офіційно визнана в 1978 році.

Опис структури даних на будь-якому рівні називається *схемою*. Існує три різних типи схем бази даних, які визначаються відповідно до рівнів абстракції трирівневої архітектури. На найвищому рівні є кілька *зовнішніх схем або підсхем*, які відповідають різним Представленням даних. На концептуальному рівні опис бази даних називають *концептуальною схемою*, а на найнижчому рівні абстракції – *внутрішньою схемою*.

Основним призначенням трирівневої архітектури є забезпечення незалежності від даних. Суть цієї незалежності полягає в тому, що зміни на нижніх рівнях ніяк не впливають на верхні рівні. Розрізняють два типи незалежності від даних: логічну й фізичну.

Логічна незалежність від даних означає повну захищеність зовнішніх схем від змін, внесених у концептуальну схему. Такі зміни концептуальної схеми, як додавання або видалення нових сутностей, атрибутів або зв'язків, повинні здійснюватися без необхідності внесення змін у вже існуючі зовнішні схеми для інших груп користувачів. Таким чином, тим групам користувачів, яких ці зміни не стосуються, не буде потрібно вносити зміни у свої програми.

Фізична незалежність від даних означає захищеність концептуальної схеми від змін, внесених у внутрішню схему. Такі зміни внутрішньої схеми, як використання різних файлових систем або структур зберігання, різних пристроїв зберігання, модифікація індексів або хешування, повинні здійснюватися без необхідності внесення змін у концептуальну або зовнішню схеми. Користувачем можуть бути замічені зміни тільки в загальній продуктивності системи.

Далі розглянемо кожен з трьох названих рівнів.

3.1 Зовнішній рівень

Зовнішній рівень – це користувацький рівень. Користувачем може бути програміст, або кінцевий користувач, або адміністратор бази даних. Представлення бази даних з погляду користувачів називається зовнішнім представленням. Кожна група користувачів виділяє в загальній для всієї організації предметної області ті сутності, атрибути й зв'язки, які їй цікаві. Виражаючи їх у найбільш зручній для себе

формі, вона формує своє користуvalьницьке представлення, причому ті ж самі дані можуть відображатися по-різному в різних користуvalьницьких представленнях. Наприклад, в інформаційній системі ВУЗ користуvalьча з бухгалтерії буде цікавити інформація про студентів, яким повинна бути нарахована стипендія, але його зовсім не буде цікавити інформація про успішність всіх студентів і багато чого іншого. Ці часткові або перевизначені описи БД для окремих груп користуvalьчів або орієнтовані на окремі аспекти предметної області називають *підсхемою*. При розгляді зовнішнього рівня БД важливо відзначити, що кожен тип користуvalьчів може застосовувати для роботи із БД свою мову спілкування. Кінцеві користуvalьчі вживають або мову запитів, або спеціальну мову, підтримувану додатками й утворюючи певні для користуvalьча екранні форми й користуvalьницькі меню. Прикладні програмісти частіше застосовують або мови високого рівня, наприклад C++, Pascal і т.п. , або спеціальні мови СУБД, які відносяться до мов четвертого покоління. Яка би мова високого рівня не використовувалась (вона у цьому випадку називається *базовою*), вона повинна містити в собі й підмову для роботи з даними. Система може підтримувати будь-яку кількість підмов даних, будь-яку кількість базових мов. Але мова SQL підтримується практично всіма системами. Вона може використатися і як вбудована в інші мови, і як окрема самостійна мова запитів.

3.2 Концептуальний рівень

Концептуальний рівень є проміжним рівнем у трирівневій архітектурі й забезпечує представлення всієї інформації бази даних в абстрактній формі. Опис бази даних на цьому рівні називається *концептуальною схемою*, що є результатом *концептуального проектування*. Концептуальне проектування бази даних включає аналіз інформаційних потреб користуvalьчів і визначення потрібних їм елементів даних. Таким чином, концептуальна схема – це єдиний логічний опис всіх елементів даних і відносин між ними, логічна структура всієї бази даних. Для кожної бази даних є тільки одна концептуальна схема. Концептуальна схема повинна містити:

- об'єкти і їхні атрибути;
- зв'язки між об'єктами;

- обмеження, що накладаються на дані;
- семантичну інформацію про дані;
- забезпечення безпеки й підтримки цілісності даних.

Концептуальний рівень підтримує кожне зовнішнє представлення, у тому розумінні, що будь-які доступні користувачеві дані повинні міститися (або можуть бути обчислені) на цьому рівні. Однак цей рівень не містить ніяких відомостей про методи зберігання даних.

3.3 Внутрішній рівень

Внутрішній рівень є третім рівнем архітектури БД. Внутрішнє представлення не пов'язане з фізичним рівнем, тому що фізичний рівень зберігання інформації має значну індивідуальність для кожної системи. На внутрішньому ж рівні всі ці індивідуальності не враховуються, і область зберігання представляється як нескінченний лінійний адресний простір. На нижньому рівні перебуває внутрішня схема, що є повним описом внутрішньої моделі даних. Для кожної бази даних існує тільки одна внутрішня схема. Внутрішня схема описує фізичну реалізацію бази даних і призначена для досягнення оптимальної продуктивності й забезпечення ощадливого використання дискового простору. Саме на цьому рівні здійснюється взаємодія СУБД із методами доступу операційної системи (допоміжними функціями зберігання й витягу записів даних) з метою розміщення даних на запам'ятовувальних пристроях, створення індексів, витягу даних і т.д. На внутрішньому рівні зберігається наступна інформація:

- розподіл дискового простору для зберігання даних й індексів;
- опис подробиць збереження записів (із вказівкою реальних розмірів елементів даних, що зберігаються);
- відомості про розміщення записів;
- відомості про стиск даних й обраних методах їхнього шифрування.

СУБД відповідає за встановлення відповідності між всіма трьома типами схем різних рівнів, а також за перевірку їхньої несуперечності.

Нижче внутрішнього рівня перебуває *фізичний рівень*, що контролюється операційною системою, але під керівництвом СУБД. Фізичний рівень ураховує, яким образом дані будуть представлені в машині. Він забезпечує фізичний погляд на базу даних: дисководи, фізичні адреси, індекси, покажчики й т.д. За цей рівень відповідають проектувальники фізичної бази даних, які працюють тільки з відомими операційній системі елементами. Область їхніх інтересів: покажчики, реалізація послідовного розподілу, способи зберігання полів внутрішніх записів на диску. Однак функції СУБД й операційної системи на фізичному рівні не цілком чітко розділені й можуть варіюватися від системи до системи. В одних СУБД використовуються багато передбачені в даній операційній системі методи доступу, тоді як в інші застосовуються тільки самі основні й реалізована власна файлова організація.

Реалізація трирівневої архітектури БД вимагає, щоб СУБД переводила інформацію з одного рівня на інший, тобто перетворювала адреси й покажчики у відповідні логічні імена й відношення й навпаки. Вигодою такого перекладу є незалежність логічного й фізичного представлення даних, але й плата за цю незалежність чимала — більша системна затримка. Для встановлення відповідності між будь-якими зовнішньою й внутрішньою схемами СУБД повинна використати інформацію з концептуальної схеми. Концептуальна схема пов'язана із внутрішньою схемою за допомогою концептуально-внутрішнього відображення. Воно дозволяє СУБД знайти фактичний запис або набір записів на фізичному пристрої зберігання, які утворюють логічний запис у концептуальній схемі. У той же час кожна зовнішня схема пов'язана з концептуальною схемою за допомогою зовнішньо-концептуального відображення. З його допомогою СУБД може відображати імена користувальницького представлення на відповідну частину концептуальної схеми.

Контрольні питання

- 1 Поясніть своїми словами зміст термінів:
 - база даних;
 - предметна область;
 - інформаційна система.

- 2 Назвіть основні компоненти інформаційної системи й поясните їхнє призначення.
- 3 Що являє собою банк даних?
- 4 Якими властивостями повинна володіти проектована база даних?
- 5 Якими можливостями володіє сучасна СУБД?
- 6 Опишіть компонентний склад середовища СУБД і призначення її компонентів.
- 7 Поясните переваги, які несе в собі використання СУБД.
- 8 Зобразіть схематично трирівневу архітектуру бази даних.
- 9 Поясніть основне призначення трирівневої архітектури бази даних.
- 10 Дайте порівняльну характеристику її зовнішньому, концептуальному й вну

Лекція № 2

з навчальної дисципліни Організація баз даних

- Тема** Типова організація сучасної СУБД. Функції СУБД. Архітектура багатокористувацьких СУБД
- Мета** Ознайомити студентів із принципами побудови трирівневої архітектури СУБД. Дати їй характеристику основних функцій СУБД. Ввести поняття архітектури багатокористувацьких СУБД

Час – 2 години

План проведення лекції

- 1 Типова організація сучасної СУБД
- 2 Функції СУБД
- 3 Архітектура багатокористувацьких СУБД

Література

1. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
2. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
3. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
4. Ярцев В.П. Організація баз даних та знань: навчальний посібник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

ВСТУП

Основні положення концепції, що визначає всі наступні етапи розробки СУБД, полягають у наступному:

- архітектура СУБД повинна забезпечувати розмежування користувальницького й системного рівнів;
- необхідно дати можливість кожному користувачеві мати своє, відмінне від інших, подання про властивості збережених даних.

Отже, початковою стадією проектування будь-якої конкретної ІС повинні бути абстрактні описи інформаційних потреб кожної групи користувачів, на основі яких генерується також абстрактне, але вже загальне для всієї організації опис структур збережених даних, а СУБД, за допомогою якої ця інформаційна система буде створюватися й підтримуватися, зобов'язана мати для цього певні можливості.

1 Типова організація сучасної СУБД

СУБД є досить складним видом програмного забезпечення, призначеного для надання перерахованих функцій. Природно, організація типової СУБД і склад її компонентів повинні відповідати розглянутому набору функцій. Деякі функції СУБД можуть підтримуватися використовуваною операційною системою, і СУБД у таких випадках завжди являє собою надбудову над ними. Таким чином, при проектуванні СУБД варто враховувати особливості інтерфейсу між створюваної СУБД і конкретною операційною системою.

СУБД складається з декількох програмних компонентів, кожний з яких націлений на виконання специфічної операції. Логічно в сучасній реляційній СУБД виділити внутрішню частину - ядро СУБД, компілятор мови БД (звичайно SQL), підсистему підтримки часу виконання, набір утиліт. У деяких системах ці частини виділяються явно, в інші - ні.

Ядро СУБД відповідає за керування даними в зовнішній пам'яті, керування буферами оперативної пам'яті, керування транзакціями й журналізацію. Відповідно можна виділити наступні компоненти ядра: менеджер даних, менеджер буферів, менеджер транзакцій і менеджер журналу. Функції цих компонентів взаємозалежні, і для забезпечення коректної роботи СУБД всі ці компоненти повинні взаємодіяти по ретельно продуманих і перевірених протоколах. Ядро СУБД має власний інтерфейс, недоступним користувачам прямо й використовуваним у програмах, вироблених компілятором SQL й утилітах БД. Ядро СУБД є основною резидентною частиною СУБД. При використанні архітектури "клієнт - сервер" ядро є основної складової серверної частини системи.

Узагальнена компонентна структура СУБД складається наступних основних програмних компонентів середовища СУБД:

1 Процесор запитів перетворить запити в послідовність низькорівневих інструкцій для контролера бази даних.

2 Компілятор мови визначення даних перетворить його команди в набір таблиць, що містять метадані. Потім ці таблиці зберігаються в системному каталозі, а керуюча інформація - у заголовках файлів з даними.

3 Препроцесор мови маніпулювання даними перетворить впроваджені в прикладні програми його оператори у виклики стандартних функцій базової мови. Для генерації відповідного коду препроцесор цієї мови повинен взаємодіяти із процесором запитів. Основною його функцією є компіляція операторів мови БД у деяку виконувану програму. Оскільки мови цих систем є не процедурними, то компілятор, використовуючи досить складні методи оптимізації операторів, повинен вирішити, яким образом виконувати оператор мови.

4 Контролер словника. Контролер словника управляє доступом до системного каталогу й забезпечує роботу з ним. Системний каталог доступний більшості компонентів СУБД.

5 Контролер файлів маніпулює призначеними для зберігання даних файлами й відповідає за розподіл доступного дискового простору. Він створює й підтримує список структур й індексів, певних у внутрішній схемі. Однак контролер файлів не управляє фізичним введенням і висновком даних безпосередньо, а лише передає

запити відповідним методам доступу, які зчитують дані в системні буфери або записують їх відтіля на диск.

6 Контролер бази даних взаємодіє із запущеними користувачами прикладними програмами й запитами. Контролер бази даних приймає запити й перевіряє зовнішні й концептуальні схеми для визначення тих концептуальних записів, які необхідні для задоволення вимог запиту. Потім контролер бази даних викликає контролер файлів для виконання запиту, що надійшов. До складу контролера бази даних входять наступні основні програмні компоненти:

- контроль прав доступу перевіряє наявність у даного користувача повноважень для виконання викликаної операції;
- процесор команд одержує керування після перевірки повноважень користувача для виконання викликаної операції;
- засобу контролю цілісності здійснюють у випадку виконання операцій, які змінюють уміст бази даних, перевірку того, чи задовольняє викликана операція всім установленим обмеженням підтримки цілісності даних;
- оптимізатор запитів визначає оптимальну стратегію виконання запиту;
- контролер транзакцій здійснює необхідну обробку операцій, що надходять у процесі виконання транзакцій;
- планувальник відповідає за безконфліктне виконання паралельних операцій з базою даних й управляє відносним порядком виконання операцій, викликаних в окремих транзакціях;
- контролер відновлення гарантує відновлення бази даних до несуперечливого стану при виникненні збоїв;
- контролер буфера відповідає за перенесення даних між оперативною пам'яттю й жорстким диском.

2 Функції СУБД

У СУБД часто виділяють систему розроблювача й систему часу виконання. Система розроблювача містить у собі компоненти СУБД, які використовуються на етапі створення додатка БД. До них ставляться засоби опису схем і підсхем БД, генератори

форм і коду, засобу візуальної розробки додатка. Система часу виконання - це частина СУБД, необхідна при роботі із БД. В основному її призначення складається в обробці запитів до БД й у підтримці її цілісності. Розглянемо функції, які повинна забезпечувати типова СУБД:

Керування даними в зовнішній пам'яті. Дана функція надає користувачам можливість виконання самих основних операцій, які здійснюються з даними, - це збереження, витяг і відновлення інформації. Вона містить у собі забезпечення необхідних структур зовнішньої пам'яті як для зберігання даних, що безпосередньо входять у БД, так і для службових цілей, наприклад для прискорення доступу до даних. У деяких реалізаціях СУБД активно використовуються можливості існуючих файлових систем, в інших робота виробляється аж до рівня пристроїв зовнішньої пам'яті. Користувачі СУБД у кожному разі не зобов'язані знати, чи використовує СУБД файлову систему, і як-що використовує, то як організовані файли.

Керування транзакціями. Транзакція - це послідовність операцій над БД, розглянутих СУБД як єдине ціле. Транзакція являє собою набір дій, виконуваних з метою доступу або зміни вмісту бази даних. Прикладами простих транзакцій може служити додавання, відновлення або видалення в базі даних відомостей про якийсь об'єкт. Складна ж транзакція утвориться в тому випадку, коли в базу даних потрібно внести відразу кілька змін. Ініціалізація транзакції може бути викликана окремим користувачем або прикладною програмою. Якщо під час виконання транзакції відбудеться збій, наприклад, через вихід комп'ютера з ладу, база даних попадає в суперечливий стан, оскільки деякі зміни вже будуть внесені, а інші - ще немає. Тому всі часткові зміни повинні бути скасовані для повернення бази даних у колишній, не суперечливий стан. Таким чином, або транзакція успішно виконується, і СУБД фіксує зміни БД, зроблені цією транзакцією, у зовнішній пам'яті, або жодне із цих змін не фіксується. Підтримка механізму транзакцій є обов'язковою умовою для будь-яких СУБД, але особливо важливо для багатокористувачьких СУБД. Та властивість, що кожна транзакція починається при цілісному стані БД і залишає цей стан цілісним після свого завершення, робить дуже зручним використання поняття транзакції як одиниці акти-

вності користувача стосовно БД. При відповідному керуванні паралельно, що виконуються транзакціями, з боку СУБД кожний з користувачів може в принципі відчувати себе єдиним користувачем СУБД.

Відновлення бази даних. Одним з основних вимог до СУБД є надійність зберігання даних у зовнішній пам'яті. Під надійністю зберігання розуміється те, що СУБД повинна могла відновити останній погоджений стан БД після будь-якого апаратного або програмного збою. Звичайно розглядаються два можливих види апаратних збоїв: м'які збої, які можна трактувати як раптову зупинку роботи комп'ютера (наприклад, аварійне вимикання харчування); тверді збої, що характеризуються втратою інформації на носіях зовнішньої пам'яті. Прикладами програмних збоїв можуть бути аварійне завершення роботи СУБД (через помилку в програмі або в результаті деякого апаратного збою) або аварійне завершення користувальницької програми, у результаті чого деяка транзакція залишається незавершеною. Першу ситуацію можна розглядати як особливий вид м'якого апаратного збою; при виникненні останньої потрібно ліквідувати наслідку тільки однієї транзакції. Зрозуміло, що в кожному разі для відновлення БД потрібно мати деяку додаткову інформацію. Інакше кажучи, підтримка надійності зберігання даних у БД вимагає надмірності зберігання даних, причому та частина даних, що використовується для відновлення, повинна зберігатися особливо надійно. Найпоширенішим методом підтримки такої надлишкової інформації є ведення журналу змін БД.

Підтримка мов БД. Для роботи з базами даних використовуються спеціальні мови, називані мовами баз даних. У сучасних СУБД звичайно підтримується єдина інтегрована мова, що містить всі необхідні засоби для роботи із БД, починаючи від її створення і до забезпечення базового користувальницького інтерфейсу із базами даних. Стандартною мовою найпоширеніших у наш час реляційних СУБД є мова SQL (Structured Query Language - мова структурованих запитів). Мова SQL дозволяє визначати схему реляційної БД і маніпулювати даними.

Словник даних. Системний каталог, що ще називають словником даних, є сховищем інформації, що описує дані в базі даних. Передбачається, що каталог доступний як користувачам, так і функціям СУБД. Кількість збереженої в каталозі інфор-

мації й спосіб її застосування у великому ступені залежить від типу використовуваної СУБД. Можна виділити наступну інформацію, що звичайно втримується в словнику даних:

- імена, типи й розміри елементів даних;
- імена зв'язків;
- обмеження, що накладаються на дані для підтримки цілісності;
- імена користувачів, яким надане право доступу до даних;
- зовнішня, концептуальна й внутрішня схеми й відображення між ними;
- статистичні дані, наприклад частота транзакцій і лічильники звертань до

об'єктів бази даних.

Керування паралельним доступом. Одна з основних цілей створення й використання СУБД полягає в тім, щоб безліч користувачів могло здійснювати паралельний доступ до спільно оброблюваних даних. Паралельний доступ порівняно просто організувати, якщо всі користувачі виконують тільки читання даних, оскільки в цьому випадку вони не можуть перешкодити один одному. Однак коли два або більше користувачі одночасно одержують доступ до бази даних, конфлікт із небажаними наслідками легко може виникнути, наприклад, якщо хоча б один з них спробує оновити дані. СУБД повинна гарантувати, що при одночасному доступі до бази даних багатьох користувачів подібних конфліктів не відбудеться.

Керування буферами оперативної пам'яті. СУБД звичайно працюють із БД значного розміру. Якщо при звертанні до будь-якого елемента даних буде вироблятися обмін із зовнішньою пам'яттю, то вся система буде працювати зі швидкістю пристрою зовнішньої пам'яті. Практично єдиним способом реального збільшення цієї швидкості є буферизація даних в оперативній пам'яті. При цьому для цілей СУБД недостатньо можливостей загальносистемної буферизації. Тому в розвинених СУБД підтримується власний набір буферів оперативної пам'яті із власною дисципліною заміни буферів.

Контроль доступу до даних. СУБД повинна мати механізм, що гарантує можливість доступу до бази даних тільки санкціонованих користувачів і захищаючий її від будь-якого несанкціонованого доступу. У сучасних СУБД підтримується один із

двох широко розповсюджених підходів до питання забезпечення безпеки даних: виборчий підхід або обов'язковий підхід. В обох підходах система безпеки може бути створена як для всієї бази даних у цілому, так і для деякої сукупності даних або навіть для деякого атрибута даного. У більшості сучасних систем передбачається виборчий підхід, при якому якийсь користувач має різні права при роботі з різними об'єктами. Значно рідше застосовується альтернативний, обов'язковий підхід, де кожному об'єкту даних привласнюється деякий класифікаційний рівень, а кожен користувач має деякий рівень допуску. Оскільки не буває невразливих систем безпеки, то при роботі з особливо коштовними даними виникає необхідність реєстрації контрольного сліду виконуваних операцій, що при виникненні критичної ситуації допоможе виявити порушника або переконатися в тім, що положення під контролем.

Підтримка обміну даними. Подібна функція також повинна надаватися всіма сучасними СУБД, тому що в цей час більшість користувачів здійснюють доступ за допомогою терміналів через мережу до бази даних, розглянутої як загальний ресурс для всіх існуючих користувачів. При цьому передбачається, що не база даних повинна бути розподілена в мережі, а вилучені користувачі повинні мати можливість доступу до централізованої бази даних. У такій ситуації, щоб бути комерційно життєздатною, навіть СУБД для персональних комп'ютерів повинна підтримувати роботу в локальній мережі й мати здатність до інтеграції з комунікаційним програмним забезпеченням і з різноманітними існуючими менеджерами обміну даними.

Підтримка цілісності даних. Термін цілісність використовується для опису коректності й несуперечності збережених у БД даних. Реалізація підтримки цілісності даних припускає, що СУБД повинна містити відомості про ті правила, які не можна порушувати при роботі з даними, і мати інструменти контролю за тим, щоб дані і їхньої зміни, відповідали заданим правилам. Таким чином, цілісність пов'язана з якістю самих даних й, отже, із забезпеченням безпеки й може розглядатися як ще один тип захисту бази даних. Аспекти цілісності необхідно враховувати як при проектуванні бази даних, так і під час її використання. Обмеження або правила збереження несуперечності даних, які не повинні порушуватися в базі, повинні бути задані зручною мовою й зберігатися в системному каталозі. Звичайно вони визначаються адміністратором бази даних і не залежать від користувача.

Підтримка незалежності від даних. СУБД повинна володіти інструментами підтримки незалежності програм від фактичної структури даних. Звичайно це досягається за рахунок реалізації механізму підтримки уявлень або підсхем. Цей механізм включає кілька типів допустимих змін фізичних характеристик бази даних, які ніяк не впливають на подання. Як правило, система легко адаптується до додавання нового об'єкта, атрибута або зв'язку, але не до їхнього видалення. Однак домогтися повної логічної незалежності від даних значно складніше.

Допоміжні функції. До допоміжних функцій відносяться такі, які занадто складно виконувати з використанням мови БД. Допоміжні утиліти звичайно призначені для надання допомоги АБД в ефективному адмініструванні бази даних. Наприклад, програми статистичного аналізу, що дозволяють оцінити ступінь використання бази даних, програми, призначені для завантаження або вивантаження бази даних, глобальна перевірка цілісності БД і т.д. Одні утиліти працюють на зовнішньому рівні, а тому вони, у принципі, можуть бути створені самими АБД, тоді як інші функціонують на внутрішньому рівні системи, і тому повинні бути надані самим розроблювачем СУБД.

3 Архітектура багатокористувацьких СУБД

Якщо комп'ютер працює в монопольному режимі, то й розміщена на персональному комп'ютері БД буде функціонувати також у монопольному режимі навіть у тому випадку, якщо із БД працюють кілька користувачів, оскільки вони можуть звертатися до неї тільки послідовно. При переході до багатокористувацького режиму, а тут є тільки один шлях - інтеграція комп'ютерів у локальну мережу, можливість розподілу додатків, що працюють із єдиною базою даних, і навіть самої бази даних по створеній мережі.

Традиційною архітектурою багатокористувацьких систем, що зложилася до появи ПК, уважалася схема, при якій один потужний комп'ютер з єдиним процесором був з'єднаний з декількома користувальницькими терміналами, що не мають для зберігання й обробки даних власних ресурсів. Системи розподіленої обробки даних будувалися на мультипрограмних операційних системах і використали централізоване

зберігання БД на пристроях зовнішньої пам'яті центральної ЕОМ і термінальний багатокористувацький режим доступу до неї. СУБД і додатки також розташовувалися на центральній ЕОМ. Користувальницькі додатки зверталися до необхідних служб СУБД. У такий же спосіб повідомлення поверталися назад на користувальницький термінал. Природно, що при такій архітектурі основна й надзвичайно більше навантаження покладало на центральний комп'ютер, що повинен виконувати не тільки дії прикладних програм і СУБД, але й більшу роботу з обслуговування терміналів (див. рис. 1).

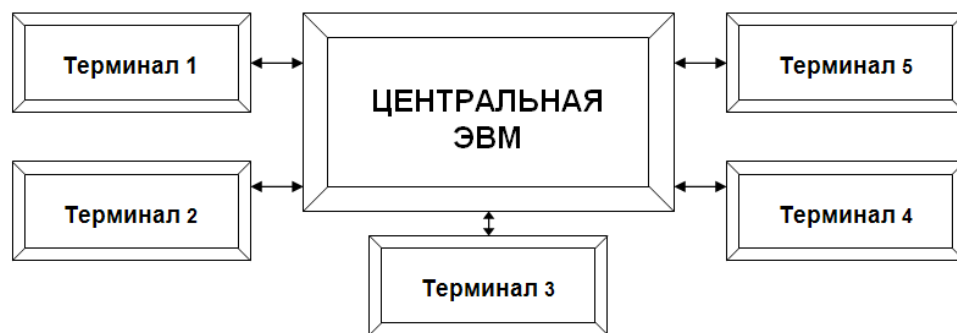


Рис. 1. Схема ранньої багатокористувацької системи

Першою системою, що працює в багатокористувацькому режимі, була СУБД SYSTEM R, розроблена фірмою IBM. У ній були реалізовані основні принципи синхронізації, застосовувані при розподіленій обробці даних, які дотепер є базисними практично у всіх комерційних СУБД. З моменту появи СУБД SYSTEM R пройшов тривалий період часу, у пліні якого відбувалися значні коливання в обчислювальних ресурсах і схемах їхнього застосування, використовуваних для зберігання й обробки інформації. Спостерігалися й окремі тенденції в цих коливаннях:

- Downsizing - децентралізація;
- UpSizing - централізація;
- RightSizing - визначення розміру й схеми відповідно до реальної ситуації.

Перша тенденція була викликана до життя рухом від окремих Mainframe-систем до відкритих розподілених систем, що використовують мережі персональних комп'ютерів, і привела до більше ефективного з погляду експлуатаційних витрат системам. Цей підхід відбиває організаційну структуру компанії, що логічно складає з окремих

підрозділів, які розподілені по різних офісах. Зустрічний стосовно розглянутої тенденції процес відбувався практично паралельно з першим і був обумовлений бурхливим розвитком персональних комп'ютерів, появою локальних мереж. Високі темпи росту продуктивності й функціональних можливостей ПК дозволили розроблювачам професійних СУБД випустити у світло ряд систем, що одержали широке поширення. У результаті цього пануюче положення зайняла тенденція створення інформаційних систем на такій платформі, що точно відповідала б її масштабам і завданням. Різні варіанти реалізації режимів роботи систем баз даних можна представити у вигляді схеми (див. рис. 2).

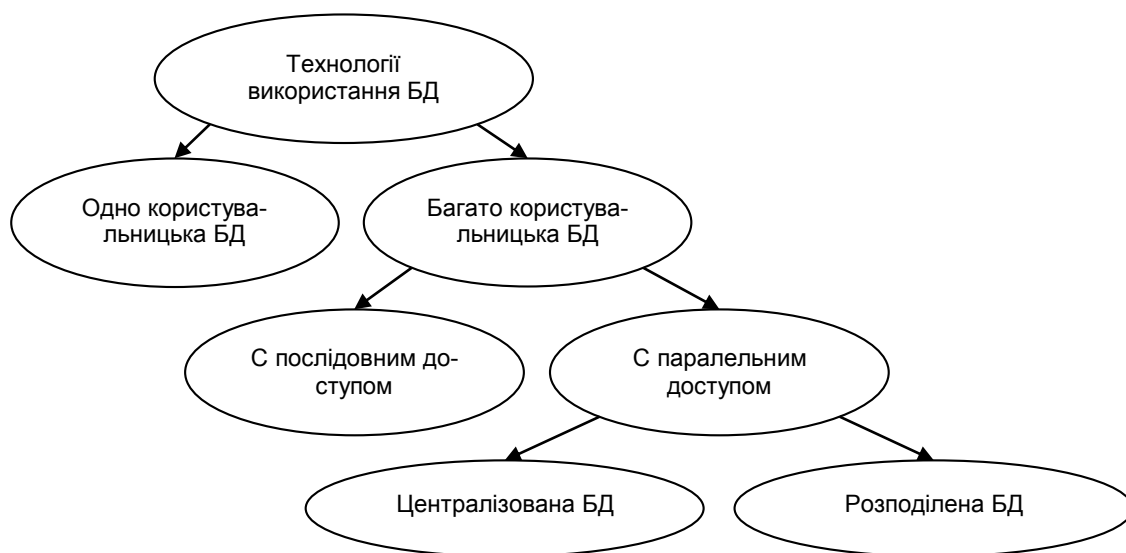


Рис. 2. Технології використання БД

У сучасному світі всі зазначені на малюнку технології використання баз даних мають своє право на життя. Різні режими можуть бути реалізовані в межах однієї й тієї ж організації й навіть на тому самому комп'ютері.

Із всіх режимів, показаних на малюнку 6, найбільші проблеми виникають при реалізації баз даних з паралельним доступом. У теперішній час для роботи із централізованими БД застосовуються технології паралельного або розподіленого доступу. Найбільше часто використовуються два типи технологій: технологія файлового сервера й технологія "клієнт - сервер", які є дворівневими структурами. Першу можна вважати часткою випадку другий. Наступний розвиток останніх, прагнення усунути їх

деякі недоліки викликали поява інших моделей і структур спільної роботи клієнтів і сервера.

Моделі дворівневої технології "клієнт - сервер". Загальна мета систем баз даних - підтримка розробки й виконання додатків баз даних. Систему баз даних можна розглядати як систему, де здійснено розподіл процесу виконання за принципом взаємодії двох програмних процесів, один із яких у цій моделі називається "клієнтом", а іншого, обслуговуючого клієнта, - сервером (машина, що зберігає бази даних). Клієнтський процес запитує деякі послуги, а серверний процес забезпечує їхнє виконання. При цьому передбачається, що один серверний процес може обслужити безліч клієнтських процесів (див. рис. 3).

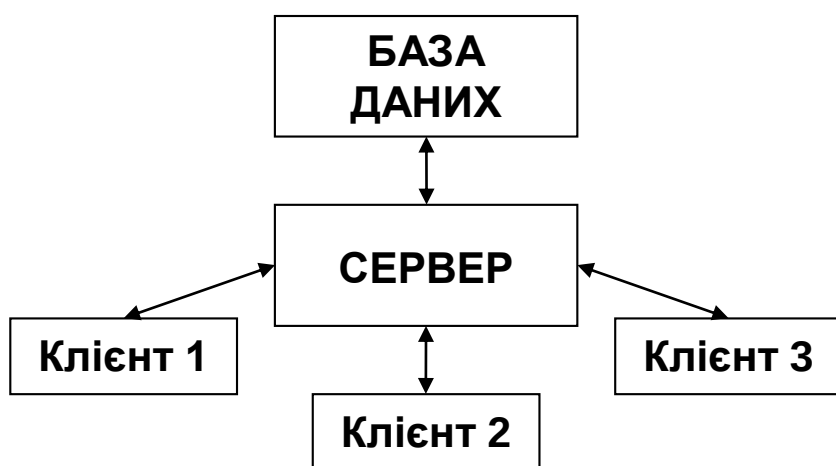


Рис. 3. Структура системи БД із виділенням клієнтів і сервера

Сервер у найпростішому випадку - це властиво СУБД. Він підтримує всі основні функції СУБД і надає повну підтримку на зовнішньому, концептуальному й внутрішньому рівнях.

Клієнти - це різні додатки, які виконуються над СУБД.

Подібна розбивка з'явилася природним продовженням отриманого в спадщину досвіду побудови багатокористувацьких систем. Однак слід зазначити, що в цю спадщину довелося внести істотні корективи. Справа в тому, що зараз у подібні мережі поєднують комп'ютери із власними чималими ресурсами й розумно так розподілити навантаження на них, щоб максимальним образом використати їхні можливості. Для втілення даного підходу треба було розбити функції єдиного додатка на окремі відносно самостійні компоненти й визначити місце розміщення кожної частини, а також

принципи взаємозв'язку між ними. Звичайно в додатку виділяються наступні групи функцій:

1 Функції введення й відображення даних - презентаційна частина додатка - визначаються тим, що користувач бачить на своєму екрані, коли працює додаток. Тому основними завданнями цієї частини додатка є:

- формування екранних зображень;
- читання й запис в екранні форми інформації;
- керування екраном;
- обробка рухів миші й натискань клавіш клавіатури.

2 Прикладні функції визначають основні алгоритми рішення конкретних завдань додатка. Звичайно код додатка пишеться з використанням різних мов програмування, таких як 3, Cobol, SmallTalk, Pascal.

3 Функції обробки даних пов'язані з обробкою даних усередині додатка. Даними управляє властиво СУБД. Для забезпечення доступу до даних використовуються мова запитів і засобу маніпулювання даними стандартної мови SQL. Звичайно оператори мови SQL вбудовуються в мови, які використовуються для написання коду додатка.

4 Функції керування інформаційними ресурсами (процесор керування даними) - це властиво СУБД, що забезпечує зберігання й керування базами даних.

5 Службові функції виконують роль зв'язувань між функціями інших груп. У монолітному виконанні всі перераховані компоненти додатків розташовуються в єдиному середовищі й комбінуються усередині однієї виконуваної програми.

У децентралізованій архітектурі ці частини додатки розподіляються по мережі. Якщо всі п'ять компонентів додатка розподіляються тільки між двома процесами, які виконуються на двох платформах: на клієнті й на сервері, то така модель називається дворівневою. Вона має кілька основних різновидів.

Файловий сервер. Модель файлового сервера називається моделлю віддаленого керування даними. Дана модель припускає наступний розподіл функцій: на клієнті розташовуються майже всі частини додатка: презентаційна частина додатка, прикладні функції, а також функції керування інформаційними ресурси. Файловий сервер

містить файли, необхідні для роботи додатків і самої СУБД і підтримує доступ до файлів (див. рис. 4).

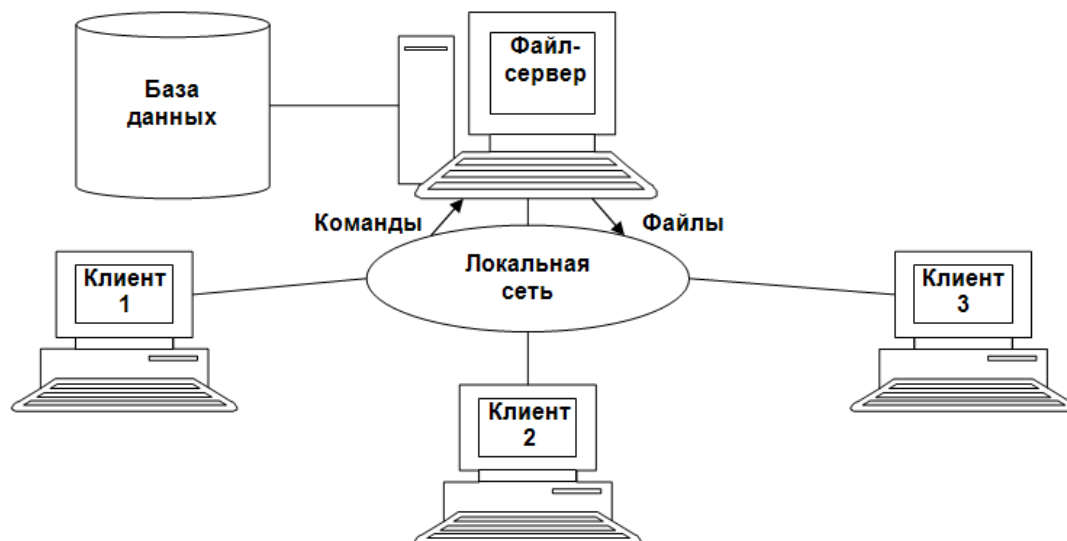


Рис. 4. Модель файлового сервера

СУБД посилає запити файлового серверу по всім необхідним їй даним. Запит клієнта формулюється в командах ЯМД. СУБД переводить цей запит у послідовність файлових команд. Кожна файлова команда викликає перекачування блоку інформації на клієнта. Далі на клієнті СУБД аналізує отриману інформацію, і якщо в отриманому блоці не втримується відповідь на запит, то приймається рішення про передачу наступного блоку інформації й т.д. Передача інформації із сервера виробляється доти , поки не буде отримана відповідь на запит клієнта. Оскільки передача файлів являє собою тривалу процедуру, такий підхід характеризується значним мережним трафіком, що може привести до зниження продуктивності всієї системи в цілому. Крім цього недоліку використання файлового сервера несе ще й інші:

- на кожній робочій станції повинна перебувати повна копія СУБД;
- керування паралельністю, відновленням і цілісністю ускладнюється, оскільки доступ до тим самим файлів можуть здійснювати відразу кілька екземплярів СУБД;
- вузький спектр операцій маніпулювання даними, що визначається тільки файловими командами;
- захист даних здійснюється тільки на рівні файлової системи.

Проте, основне достоїнство цієї моделі полягає в тому, що в ній уже здійснене поділ монопольного додатка на два взаємодіючих процеси. При цьому сервер може обслуговувати безліч клієнтів, що звертаються до нього із запитами.

Модель вилученого доступу до даних. У моделі вилученого доступу база даних також зберігається на сервері. На сервері ж перебуває і ядро СУБД. На клієнті розташовуються частини додатка, що підтримують функції введення й відображення даних і прикладні функції. Клієнт звертається до сервера із запитом мовою SQL. Структура моделі вилученого доступу наведена на рис. 5.

Стосовно файлового сервера даний підхід має серйозні переваги. З одного боку, установка компонента подання й прикладного компонента на клієнтський комп'ютер не дозволяє перевантажити сервер БД, зводячи до мінімуму загальне число процесів в операційній системі. З іншого боку, серверу БД виділяються тільки властиві йому функції; він завантажується операціями обробки даних, запитів і транзакцій.

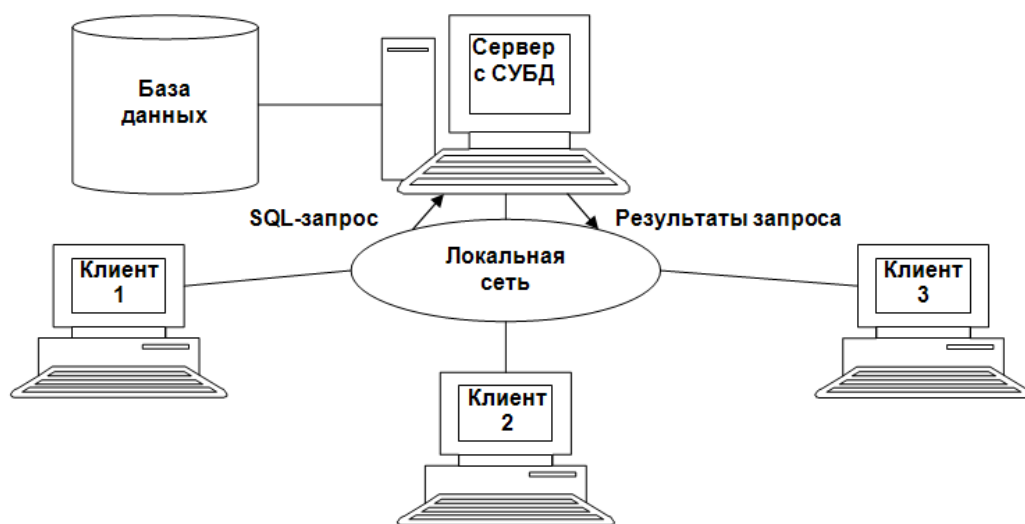


Рис. 5. Модель віддаленого доступу

Сервер приймає й обробляє запити з боку клієнтів, перевіряє повноваження користувачів, гарантує дотримання обмежень цілісності, виконує відновлення даних, виконує запити й повертає результати клієнтові, підтримує системний каталог, забезпечує паралельний доступ до бази даних й її відновлення. До того ж різко зменшується завантаження мережі, тому що по ній від клієнтів до сервера передаються не файлові команди, а запити на SQL, і їхній обсяг істотно менше. У відповідь на запити клієнт одержує тільки дані, що відповідають запиту, а не блоки файлів, як у моделі

файлового сервера. Основне достоїнство даної технології в тім, що стандартом при спілкуванні клієнта й сервера стає мова SQL. Проте, дана технологія володіє й рядом недоліків:

- запити мовою SQL при інтенсивній роботі клієнтських додатків можуть істотно завантажити мережа;
- презентаційні й прикладні функції додатка повинні бути повторені для кожного клієнтського додатка;
- сервер відіграє пасивну роль, тому функції керування інформаційними ресурсами повинні виконуватися на клієнті.

Модель сервера баз даних. Технологію "клієнт - сервер" підтримують більшість сучасних СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server. Ця технологія є подальшим розвитком моделі вилученого доступу до даних. В основу даної моделі доданий механізм збережених процедур і механізм тригерів.

Механізм збережених процедур дозволяє створювати підпрограми, що працюють на сервері й керовані його процесами. Подібні підпрограми можуть бути активізовані зухвалим їхнім додатком. Крім того, вони можуть бути викликані правилами, що підтримують цілісність даних, або тригерами. Збережені процедури тісно взаємодіють із оптимізатором сервера, що дозволяє одержати високу продуктивність при обробці даних. Таким чином, розміщення на сервері збережених процедур означає, що прикладні функції додатка розділені між клієнтом і сервером. Клієнтський додаток звертається до сервера з командою запуску збереженої процедури, а сервер виконує цю процедуру й реєструє всі зміни в БД, які в ній передбачені. Сервер повертає клієнтові дані, які потрібні клієнтові або для висновку на екран, або для виконання тієї частини прикладних функцій, які розташовані на клієнті. Трафік обміну інформацією між клієнтом і сервером різко зменшується.

Централізований контроль цілісності БД у моделі сервера БД виконується з використанням механізму тригерів. Тригери також є частиною БД. Тригер - це особливий тип збереженої процедури, що реагує на виникнення певної події в БД. Він активізується при по-катуванні зміни даних - при операціях додавання, відновлення й видалення. Тригери визначаються для конкретних таблиць БД. Вони дають можливість

підтримувати цілісність бази даних, не покладаючись на програмне забезпечення додатків. Ядро СУБД проводить контроль всіх подій, які викликають створені й описані тригери в БД, і при виникненні відповідної події сервер запускає відповідний тригер. Тригер - це фільтр, застосовуваний після виконання операції. Якщо тригер викликає помилку в запиті, відновлення інформації не виробляється, а в додаток, що виконує ця дія, повертається повідомлення про помилку. Тригер розглядається як єдине ціле - як одна транзакція, що або фіксується, або відкочується. Впровадження тригерів незначно впливає на продуктивність сервера й часто використовується для посилення додатків, що виконують багатокаскадні операції в БД. Тригери можуть викликати збережені процедури. У даній моделі (див. рис. 6) сервер є активним, тому що не тільки клієнт, але й сам сервер, використовуючи механізм тригерів, може бути ініціатором обробки даних у БД. Оскільки функції клієнта полегшені переносом частини прикладних функцій на сервер, він у цьому випадку називається "тонким".

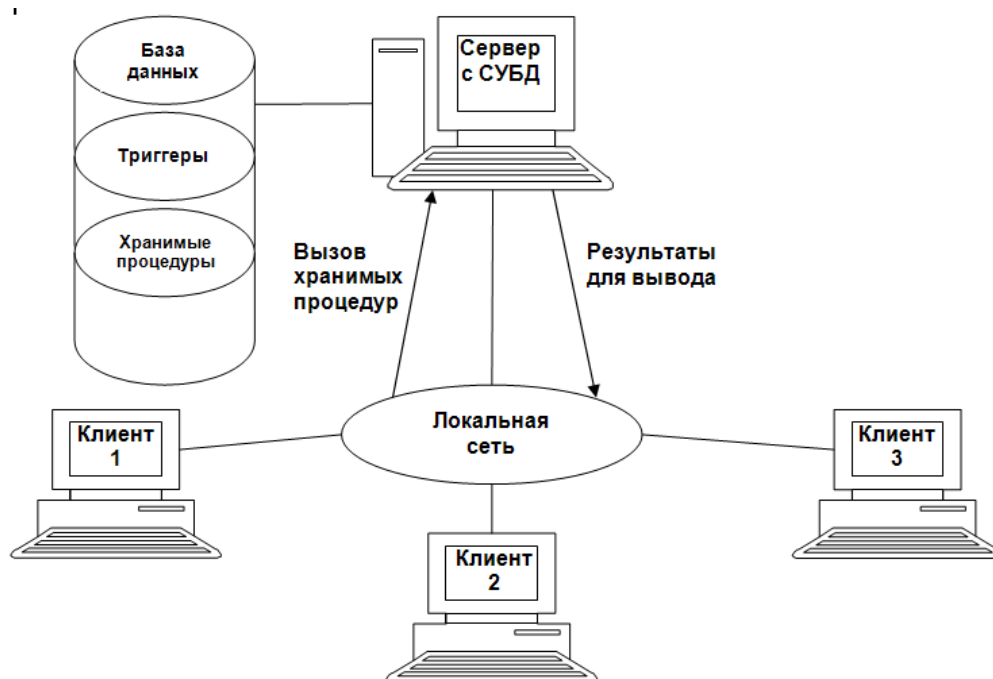


Рис. 6. Модель сервера БД

Необхідні процедури й тригери зберігаються в словнику БД, вони можуть бути використані декількома клієнтами, що істотно зменшує дублювання алгоритмів обробки даних у різних клієнтських додатках. Для написання збережених процедур і тригерів використовується розширення стандартної мови SQL - убудований SQL.

При всіх позитивних якостях даної моделі в неї все-таки є один недолік - дуже більше завантаження сервера. Для розвантаження сервера була запропонована трирівнева модель.

Завершуючи аналіз дворівневих моделей з архітектурою "клієнт-сервер", розглянемо можливі варіанти топології таких систем. Першим кроком у створенні механізму організації взаємодії процесів типу "клієнт" й "сервер" можна вважати виділення функції керування даними в самостійну групу - сервер, що дозволив, зокрема, помістити сервер на одну машину, а програмний інтерфейс із користувачем - на іншу, здійснюючи взаємодію між ними по мережі. Топологію такої моделі взаємодії користувача із сервером відносять до типу "один клієнт - один сервер" (див. рис. 7.а), де сервер обслуговує запити тільки одного клієнта й для обслуговування декількох клієнтів потрібно запустити відповідне число серверів. Природно, що реалізація такої моделі висувала підвищені вимоги до ресурсів ЕОМ, на якій запускалися всі серверні процеси, і відрізнялася складністю забезпечення взаємодії серверних процесів.

Системи з виділеним сервером здатні обробляти запити від багатьох клієнтів. Моделі з такою архітектурою мають топологію "декілька клієнтів - один сервер" (див. рис. 7. б). Вона дозволяє значно зменшити навантаження на операційну систему й потреби в пам'яті, що виникають при роботі великої кількості користувачів.

Подальший розвиток системи "клієнт - сервер" одержали при застосуванні СУБД для мультипроцесорних платформ, де кількість актуальних серверів може бути погоджене з кількістю процесорів у системі. Топологія подібних систем така, що клієнти підключаються не до реального сервера, а до проміжної ланки, називаній диспетчером, що виконує тільки функції диспетчеризації запитів до актуальних серверів. Такі системи відносять до систем з віртуальним сервером, а їхню топологію - до виду «кілька клієнтів - кілька серверів».

Контрольні питання

1 Зобразите схематично трирівневу архітектуру бази даних і поясните її основне призначення.

- 2 Дайте порівняльну характеристику її зовнішньому, концептуальному й внутрішньому рівням.
- 3 Поясніть призначення словника даних.
- 4 Перелічіть основні функції СУБД.
- 5 Представте основні програмні компоненти середовища СУБД.
- 6 Які групи функцій виділяються в додатку?
- 7 Опишіть тенденції розвитку багатокористувацьких систем.
- 8 Що собою представляє технологія "клієнт - сервер"?
- 9 Дайте характеристику моделям дворівневої технології "клієнт-сервер".
- 10 Які достоїнства й недоліки моделі вилученого доступу до даних?
- 11 Охарактеризуйте модель сервера баз даних.
- 12 Чим викликана поява трьохрівневих моделей технології "клієнт - сервер"?
Які переваги в цих моделей?

Лекція № 3

з навчальної дисципліни Організація баз даних

Тема Життєвий цикл бази даних

Мета Ознайомити студентів з поняттям життєвого циклу бази даних. Дати характеристику основних етапів.

Час – 2 години

План проведення лекції

1. Життєвий цикл БД
2. Етапи життєвого циклу БД
 - 2.1. Планування розробки бази даних
 - 2.2. Визначення вимог до системи
 - 2.3. Збір й аналіз вимог користувачів
 - 2.4. Проектування бази даних
 - 2.5. Розробка додатків
 - 2.6. Реалізація
 - 2.7. Завантаження даних
 - 2.8. Тестування
 - 2.9. Експлуатація й супровід

Література

1. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
2. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
3. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
4. Ярцев В.П. Організація баз даних та знань: навчальний посібник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Життєвий цикл БД

Як і будь-який програмний продукт, база даних має власний життєвий цикл (ЖЦБД). Головні складові у життєвому циклі БД є створення єдиної бази даних і програм, необхідних для її роботи. Життєвий цикл системи бази даних визначає й життєвий цикл всієї інформаційної системи організації, оскільки база даних є фундаментальним компонентом інформаційної системи.

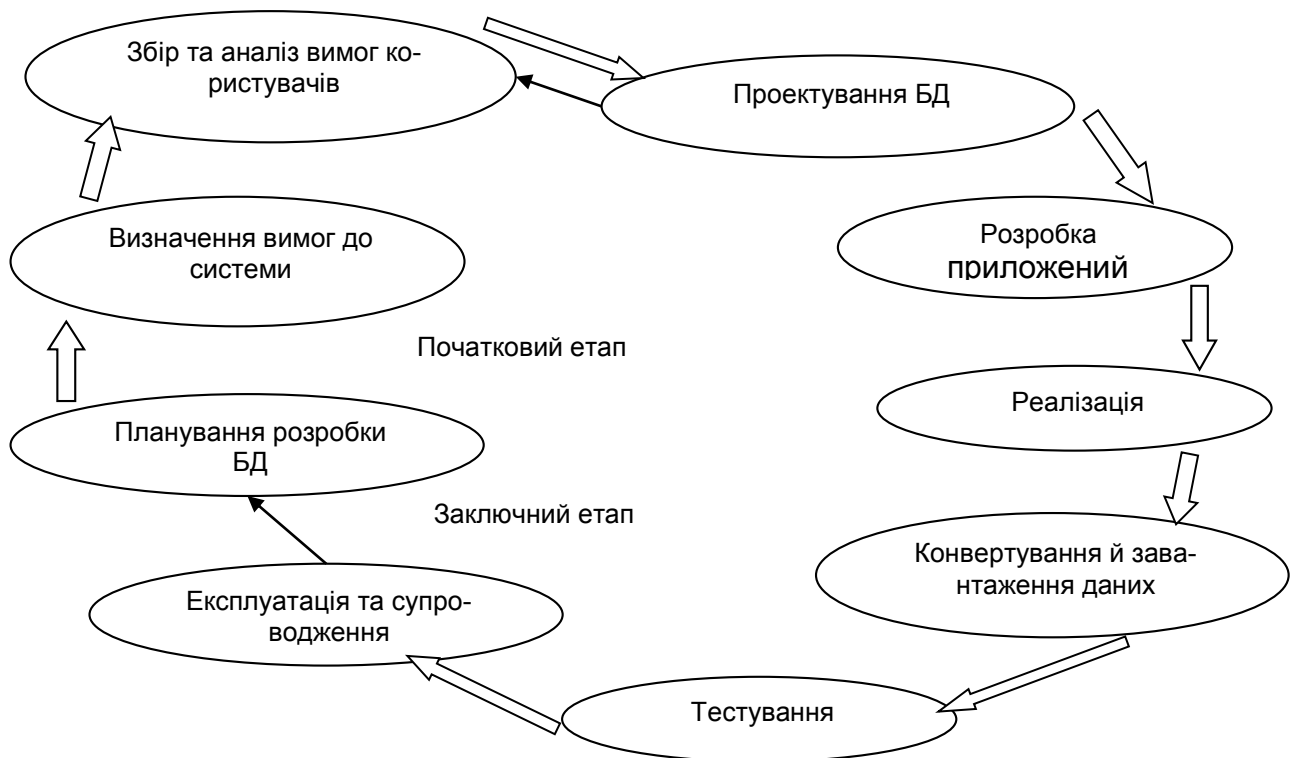


Рис. 13. Життєвий цикл БД

ЖЦБД містить у собі наступні основні етапи (див. рис. 13):

- планування розробки бази даних;
- визначення вимог до системи;
- збір й аналіз вимог користувачів;
- проектування бази даних:
 - концептуальне проектування бази даних;
 - логічне проектування бази даних;

- фізичне проектування бази даних;
- розробка додатків:
 - проектування транзакцій;
 - проектування користувальницького інтерфейсу;
- реалізація;
- завантаження даних;
- тестування;
- експлуатація й супровід:
 - аналіз функціонування й підтримка вихідного варіанта БД;
 - адаптація, модернізація й підтримка перероблених варіантів.

Конкретне наповнення кожного етапу в значній мірі залежить від складності продукту, що розроблюється. Проектування середовища великого додатка баз даних вимагає більше по-дробового опису основних дій, виконуваних на кожному етапі життєвого циклу бази даних. Для малих додатків з невеликою кількістю користувачів ЖЦБД може бути значно спрощений у результаті коректування змісту окремих етапів.

2 Етапи життєвого циклу БД

2.1 Планування розробки бази даних

Зміст даного етапу - розробка стратегічного плану, у процесі якої здійснюється попереднє планування конкретної системи керування базами даних. Планування розробки бази даних складається у визначенні трьох основних компонентів: обсягу робіт, ресурсів і вартості проекту. Важливою частиною розробки стратегічного плану є перевірка здійснюваності проекту, що складає з декількох частин.

1 Перевірка технологічної здійснюваності. Вона складається в з'ясуванні питання, чи існує устаткування й програмне забезпечення, що задовольняє інформаційним потребам фірми.

2 Перевірка операційної здійснюваності: наявність експертів і персоналу, необхідних для роботи БД.

3 Перевірка економічної доцільності здійснення проекту:

- доцільність спільного використання даних різними відділами;
- величина ризику, пов'язаного з реалізацією системи бази даних;
- очікувана вигода від впровадження підлягаючому створенню додатків;
- час окупності впровадженої БД;
- вплив системи керування БД на реалізацію довгострокових планів організації.

Планування розробки баз даних також повинне включати розробку стандартів, які визначають, як буде здійснюватися збір даних, яким буде їхній формат, яка буде потрібна документація, як буде виконуватися проектування й реалізація додатків.

2.2 Визначення вимог до системи

На даному етапі необхідно визначити діапазон дії додатка бази даних, склад його користувачів й області застосування. Визначення вимог включає вибір цілей БД, з'ясування інформаційних потреб різних відділів і керівників фірми й вимог до встановлення й програмного забезпечення. При цьому також потрібно розглянути питання, чи варто створювати розподілену базу даних або ж централізовану, і які в розглянутій ситуації знадобляться комунікаційні засоби. Написати короткий коментар, що описує мети системи. Перш ніж приступати до проектування додатка бази даних, важливо встановити границі досліджуваної області й способи взаємодії додатка з іншими частинами інформаційної системи організації. Ці границі повинні охоплювати не тільки поточних користувачів й області з розроблювальної системи, але й майбутніх користувачів і можливі області застосування.

2.3 Збір й аналіз вимог користувачів

Проектування БД засноване на інформації про ту частину організації, яка буде обслуговуватись БД. Інформаційні потреби з'ясовуються за допомогою анкет, опитувань менеджерів і працівників фірми, за допомогою спостережень за діяльністю підприємства, а також звітів і форм, якими фірма користується в сучасний момент.

На даному етапі необхідно створити для себе модель руху важливих матеріальних об'єктів й усвідомити процес документообігу. По кожному документі необхідно встановити періодичність використання, визначити дані, необхідні для виконання виділених функцій (аналізуючи існуючу й плановану документацію, з'ясовують, як виходить кожен елемент даних, ким виходить, де надалі використається, ким контролюється). Найпильніша увага повинна бути приділена дублюванню інформації, можливості появи помилкової інформації й причинам, які ведуть до їхньої появи. Також на цьому етапі бажано представити загальні параметри створюваної бази.

У підсумку зібрана інформація про кожну важливу область застосування додатка й користувацької групи повинна включати наступні компоненти: вихідну й генеровану документацію, докладні відомості про виконувані транзакції, а також список вимог із вказівкою їхніх пріоритетів. На підставі всієї цієї інформації будуть складені специфікації вимог користувачів у вигляді набору документів, що описують діяльність підприємства з різних точок зору.

2.4 Проектування бази даних

Повний цикл розробки бази даних включає концептуальне, логічне й фізичне її проектування. Основними цілями проектування бази даних є:

- подання даних і зв'язків між ними, необхідних для всіх основних областей застосування даного додатка й будь-яких існуючих груп його користувачів;
- створення моделі даних, здатної підтримувати виконання будь-яких необхідних транзакцій обробки даних;
- розробка попереднього варіанта проекту, структура якого дозволяє задовольнити вимог, пропоновані до продуктивності системи.

У створенні БД як моделі ПЗ виділяють:

- об'єктну (предметну) систему, що представляє фрагмент реального миру;
- інформаційну систему, що описує деяку об'єктну систему;
- датологічну систему, що представляє інформаційну систему за допомогою даних.

Оптимальна модель даних повинна задовольняти таким критеріям, як: структурна достовірність, простота, виразність, відсутність надмірності, розширюваність, цілісність, здатність до спільного використання.

Концептуальне проектування бази даних.

Перша фаза процесу проектування бази даних полягає в створенні для аналізованої частини підприємства концептуальної моделі даних. Побудова її здійснюється в певному порядку: на початку створюються докладні моделі користувальницьких подань даних; потім вони інтегруються в концептуальну модель даних. Концептуальне проектування приводить до створення концептуальної схеми бази даних. Існує два основних підходи до проектування систем баз даних: "спадний" й "висхідний".

При висхідному підході, що застосовується для проектування простих баз даних з відносно невеликою кількістю атрибутів, робота починається із самого нижнього рівня - рівня визначення атрибутів, які на основі аналізу існуючих між ними зв'язків групуються у відношення. Отримані відношення надалі піддаються процесу нормалізації, що приводить до створення нормалізованих взаємозалежних таблиць, заснованих на функціональних залежностях між атрибутами.

Проектування складних БД із більшою кількістю атрибутів, оскільки встановити серед атрибутів всі існуючі функціональні залежності досить важко, здійснюється використанням спадного підходу. Починається цей підхід з розробки моделей даних, які містять трохи високорівневих сутностей і зв'язків, потім робота триває у вигляді серії спадних уточнень низькорівневих сутностей, зв'язків і стосовних до них атрибутів. Спадний підхід демонструється в концепції моделі "сутність - зв'язок" (Entity-Relationship model - ER-модель) - самої популярної технології високорівневого моделювання даних, запропонованої П. Ченом. Модель "сутність - зв'язок" ставиться до семантичних моделей. Семантичне моделювання даних, зв'язане зі значенням змістом даних, незалежно від їхнього подання в ЕОМ, споконвічно виникло з метою підвищення ефективності й точності проектування баз даних. Методи семантичного моделювання виявилися застосовними до багатьох користувальницьких проблем і легко перетворюються в мережні, ієрархічні й реляційні моделі.

Крім "спадного" й "висхідного" підходів, для проектування баз даних можуть застосовуватися інші підходи, що є деякими комбінаціями зазначених.

У побудові загальної концептуальної моделі даних виділяють ряд етапів:

- 1 Виділення локальних подань, що відповідають звичайно відносно незалежним даним. Кожне таке подання проектується як під задача.
- 2 Формулювання об'єктів, що описують локальну предметну область проектованої БД, і опис атрибутів, що становлять структуру кожного об'єкта.
- 3 Виділення ключових атрибутів.
- 4 Специфікація зв'язків між об'єктами. Видалення надлишкових зв'язків.
- 5 Аналіз і додавання не ключових атрибутів.
- 6 Об'єднання локальних подань.

Побудова концептуальної моделі даних здійснюється на основі аналізу опису предметної області природною мовою, зробленого кінцевим користувачем. У процесі розробки концептуальна модель даних постійно піддається тестуванню й перевірці на відповідність вимогам користувачів. Створена концептуальна модель дані підприємства є джерелом інформації для фази логічного проектування бази даних.

Логічне проектування бази даних. Ціль другої фази проектування бази даних коштує в створенні логічної моделі даних для досліджуваної частини підприємства. Логічна модель, що відбиває особливості подання про функціонування підприємства одночасно багатьох типів користувачів, називається глобальною логічною моделлю даних. Для створення глобальної логічної моделі дані підприємства можна вибрати один із двох основних підходів - централізований підхід або підхід на основі інтеграції подань.

- 1 Відправним моментом при централізованому підході, що застосуємо тільки для не занадто складних баз даних, є утворення єдиного списку вимог шляхом об'єднання вимог всіх типів користувачів.
- 2 При використанні методу інтеграції подань здійснюється злиття окремих локальних логічних моделей даних, що відбивають подання різних груп користувачів, в єдину глобальну логічну модель даних усього підприємства.

Надалі процес проектування БД повинен опиратися на певну модель даних (реляційна, мережна, ієрархічна), що визначається типом передбачуваної для реалізації інформаційної системи СУБД. Після чого сама концептуальна модель даних уточнюється й перетворюється в логічну модель даних. У процесі розробки логічна модель

даних повинна постійно піддаватися перевірці як на відповідність вимогам користувачів, так і на відсутність надмірності даних, здатної викликати в майбутньому аномалії відновлення. Побудована логічна модель даних надалі буде затребувана на етапі фізичного проектування, а також на етапі експлуатації й супроводу вже готової системи, дозволяючи наочно представити будь-які внесені в базу дані зміни. На цьому кроці бажане створення наступних документів:

- набору підсхем;
- специфікацій для фізичного проектування додатків;
- посібника з розробки програм (інтерфейси з користувачем і між програмні інтерфейси);
- посібника із супроводу БД.

Концептуальне й логічне проектування - це ітеративні процеси, які містять у собі ряд уточнень, що тривають доти, поки не буде отриманий найбільш відповідний структурі підприємства продукт.

Фізичне проектування бази даних. Метою проектування на даному етапі є створення опису СУБД-орієнтованої моделі БД. Варто враховувати, що на цій стадії розробки можливі повернення на більше ранні етапи ЖЦБД. Наприклад, рішення, прийняті на етапі фізичного проектування з метою підвищення продуктивності системи, можуть привести до необхідності внести зміни в структуру логічної моделі даних. Дії, виконувані на цьому етапі, занадто специфічні для різних моделей даних. Для реляційної моделі даних під фізичним проектуванням мається на увазі:

- створення опису набору реляційних таблиць й обмежень для них на основі інформації, представленої в глобальній логічній моделі даних;
- визначення конкретних структур зберігання даних і методів доступу до них, що забезпечують оптимальну продуктивність системи з базою даних;
- розробка засобів захисту створюваної системи.

2.5 Розробка додатків

Паралельно із проектуванням системи БД виконується розробка додатків. Головні складові даного процесу - це проектування транзакцій і користувацького інтерфейсу.

Проектування транзакцій. Транзакції представляють деякі події реального миру. Всі транзакції повинні звертатися до бази даних з тією метою, щоб збережені в ній дані завжди гарантовано відповідали поточній ситуації в реальному світі. Транзакція може складатися з декількох операцій, однак з погляду користувача ці операції являють собою єдине ціле, що переводить базу даних з одного несуперечливого стану в інше. Реалізація транзакцій опирається на той факт, що СУБД здатне забезпечувати схоронність внесених під час транзакції змін у БД і несуперечність бази даних навіть у випадку виникнення збою. Для незавершеної транзакції СУБД гарантує скасування всіх внесених нею змін. Проектування транзакцій полягає у визначенні:

- даних, які використовуються транзакцією;
- функціональних характеристик транзакції;
- вихідних даних, формованих транзакцією;
- ступеня важливості й інтенсивності використання транзакції.

Існує три основних типи транзакцій: витягу, відновлення й змішані:

- 1 Транзакції витягу використовуються для вибірки деяких даних з метою відображення їх на екрані або приміщення у звіт.
- 2 Транзакції відновлення використовуються для вставки нових, видалення старих або корекції існуючих записів бази даних.
- 3 Змішані транзакції включають як операції витягу, так й операції відновлення даних.

Транзакції можуть являти собою складні операції, які розкладаються на трохи більше простих операцій, кожна з яких являє собою окрему транзакцію.

Проектування користувацького інтерфейсу. Користувацький інтерфейс додатків бази даних є одним з найважливіших компонентів системи. Інтерфейс повинен бути зручним і забезпечувати всі функціональні можливості, передбачені в

специфікаціях вимог користувачів. Фахівці рекомендують при проектуванні користувацького інтерфейсу використати наступні основні елементи і їхні характеристики:

- змістовна назва;
- ясні й зрозумілі інструкції;
- логічно обґрунтовані угруповання й послідовності полів;
- візуально привабливий вид вікна форми або поля звіту;
- легко пізнавані назви полів;
- погоджену термінологію й скорочення;
- погоджене використання квітів;
- візуальне виділення простору й границь полів введення даних;
- зручні засоби переміщення курсору;
- засобу виправлення окремих помилкових символів і цілих полів;
- засобу висновку повідомлень про помилки при введенні неприпустимих значень;
- особливе виділення необов'язкових для введення полів;
- засобу висновку пояснювальних повідомлень із описом полів;
- засобу висновку повідомлення про закінчення заповнення форми.
-

2.6 Реалізація

На даному етапі здійснюється фізична реалізація бази даних і розроблених додатків, що дозволяють користувачеві формулювати необхідні запити до БД і маніпулювати даними в БД. База даних описується мовою визначення даних обраної СУБД. У результаті компіляції його команд й їхнього виконання створюються схеми й порожні файли бази даних. На цьому ж етапі визначаються й всі специфічні користувацькі подання.

Прикладні програми реалізуються за допомогою мов третього або четвертого покоління. Не-котрі елементи цих прикладних програм будуть являти собою транзакції обробки бази даних, записувані мовою маніпулювання даними СУБД і викликувані із програм на базовій мові програмування. Крім того, на цьому етапі створюються

інші компоненти проекту додатка - наприклад, екрани меню, форми введення даних і звіти. Варто враховувати, що багато існуючі СУБД мають свої власні інструменти, розробки четвертого покоління, що дозволяють швидко створювати додатки за допомогою не процедурних мов запитів, різноманітних генераторів звітів, генераторів форм, генераторів графічних зображень і генераторів додатків. На цьому етапі реалізуються також використовувані додатком засоби захисту бази даних і підтримки її цілісності.

2.7 Завантаження даних

На цьому етапі створені у відповідності зі схемою бази даних порожні файли, призначені для зберігання інформації, повинні бути заповнені даними. Наповнення бази даних може протікати по-різному, залежно від того, чи створюється база даних знову або нова база даних призначена для заміни старої. У першому випадку для введення інформації використовуються створені в процесі проектування БД зручні спеціальні форми, які дозволяють АБД занести в базу заздалегідь підготовлені дані. Якщо ж нова база даних призначена для заміни старої, то виникає проблема перенесення даних у нову систему, причому дуже часто з перетворенням формату їхнього подання. Дане з одержало назву - конвертування даних. У цей час будь-яка СУБД має утиліту завантаження вже існуючих файлів у нову базу даних. Їй звичайно потрібно вказати тип джерела й цільової бази даних, після чого вона автоматично перетворить дані в потрібний формат.

2.8 Тестування

Ретельне тестування повинен з будь-який програмний продукт. Стратегія тестування припускає використання реальних даних і побудована таким чином, що весь процес виконується строго послідовно й методично правильно. Крім виявлення наявних у прикладних програмах й, можливо, у структурах БД помилок, збір статистичних даних на стадії тестування дозволяє встановити показники надійності і якості створеного ПЗ. Користувачі нової системи повинні приймати в процесі її тестування

сама активна участь із метою з'ясування їх неврахованих інформаційних потреб. І якщо такі виявляться, якщо буде потреба здійснюється відкіт назад у процесі проектування на ті стадії, де можливо внести необхідні зміни. Для оцінки закінченості й коректності виконання додатка бази даних може використатися кілька різних стратегій тестування:

1 Спадне тестування з на рівні підсистем з модулями, які представлені заглушками, тобто простими компонентами, що мають такий же інтерфейс, як модуль, але без функціонального коду. Кожен модуль низького із представляється заглушкою. Поступово всі програмні компоненти замінюються фактичним кодом і після кожної заміни знову тестуються. Перевагою цього підходу є те, що помилки проектування можуть бути виявлені ще на ранній стадії тестування, що дозволяє виключити дорогі роботи з повторного проектування й реалізації. Крім того, уже на ранній стадії створення можна одержати працюючу систему, хоча й з обмеженою функціональністю, здатну продемонструвати гнучкість обраної схеми. Недоліком цієї стратегії тестування є необхідність створення численних заглушок модулів для моделювання низькорівневих компонентів системи.

2 Висхідне тестування виконується в протилежному напрямку стосовно спадного. Воно починається з тестування модулів на найнижчих рівнях ієрархії системи, продовжується на більше високих рівнях і закінчується на найвищому рівні. Переваги й недоліки при цьому мають зворотний сенс переваг і недоліків, якими володіє стратегія спадного тестування.

3 Тестування потоків здійснюється при тестуванні працюючих у реальному масштабі часу систем, які звичайно складаються з великої кількості взаємодіючих процесів, керуємих за допомогою переривань. Стратегія тестування потоків спрямована на спостереження за окремими процесами. При цьому "потік" обробки кожної зовнішньої події "проходить" через різні системні процеси. Дана стратегія включає ідентифікацію й виконання кожного можливого "потoku" обробки в системі. Однак виконати вичерпне тестування потоків системи просто нереально через величезну кількість можливих комбінацій вхідних і вихідних умов.

Стратегія інтенсивного тестування часто включає серію тестів з поступово зростаючим навантаженням і триває доти, поки система не вийде з ладу. Ця стратегія призначена для перевірки можливості системи справлятися з навантаженням і володіє двома основними перевагами: вона перевіряє поведінку системи, а також натискає на систему, викликаючи появу збоїв, які не могли б бути виявлені у звичайних умовах експлуатації.

Різні стратегії можуть використатися для різних частин системи й на різних етапах загального процесу. При тестуванні системи доцільно вибрати таку стратегію, де система тестується не як єдине ціле, а помодульно.

2.9 Експлуатація й супровід

Даний етап є останнім і самому тривалим у життєвому циклі правильно спроектованої БД. Основні дії до спостереження за створеною системою й підтримці її нормального функціонування по закінченні розгортання. Підтримка БД припускає дозвіл проблем, що виникають у процесі експлуатації БД і пов'язаних з помилками реалізації БД, зі змінами в самій Про, створенням додаткових програмних компонентів або модернізації самої БД.

Дії, які повинен включати даний етап:

- 1 Аналіз функціонування й підтримка вихідного варіанта БД. Контроль продуктивності системи повинен вироблятися для того, щоб переконатися, що продуктивність й інші експлуатаційні показники постійно перебувають на прийнятному рівні. Для цієї мети використовуються різні утиліти адміністрування бази даних, які звичайно надає типова СУБД. При виявленні падіння продуктивності нижче прийнятного рівня адміністратор бази даних може використати отриману інформацію для додаткового налаштування або реорганізації системи з метою підвищення її продуктивності, прискорення виконання запитів шляхом зміни структур зберігання, об'єднання або розбивки окремих таблиць.

- 2 Адаптація й модернізація системи. Протягом функціонування системи в організації можуть виникнути різноманітні зміни й, якщо їх не відбити в інформацій-

ній системі, то вона, природно, буде видавати некоректну інформацію. Процес моніторингу повинен підтримуватися протягом всього життя додатків, що дозволить у будь-який момент часу провести ефективну реорганізацію бази даних з метою задоволення вимог, що змінюються.

Контрольні питання

- 1 Поясніть своїми словами зміст термінів:
 - база даних;
 - предметна область;
 - інформаційна система;
 - життєвий цикл бази даних.
- 2 Охарактеризуйте життєвий цикл бази даних. Які основні етапи він включає?
- 3 Назвіть основні цілі процесу проектування бази даних і дайте характеристику його фазам.
- 4 Які стратегії тестування прикладних програм інформаційних систем існують?

Лекція № 4

з навчальної дисципліни Організація баз даних

Тема Методологія проектування бази даних. Поняття моделей даних

Мета Ввести основні поняття та правила процесу проектування бази даних.
Розглянути класифікацію моделей даних

Час – 2 години

План проведення лекції

1. Методологія проектування БД.
 - 1.1 Процес проектування.
 - 1.2 Критерії оцінювання
 - 1.3 Інформаційні вимоги
2. Поняття та класифікація моделей даних
 - 2.1 Дані та їхня семантика
 - 2.2 Моделювання даних
 - 2.3 Ієрархічна модель
 - 2.4 Мережна модель

Література

1. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
2. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
3. Харів Н. О. Базы даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
4. Ярцев В.П. Організація баз даних та знань: навчальний посібник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1. Методологія проектування БД

Методологія проектування баз даних — це сукупність принципів, методів, інструментів і засобів, що застосовуються для послідовного розроблення структури бази даних. Оскільки система баз даних складається з програм і даних, методологія проектування баз даних розглядається як невід’ємна частина загальної методології проектування програмних систем.

До методології проектування баз даних висуваються певні вимоги. Прийнятною вважається база даних, яка відповідає вимогам користувачів (ефективність, адаптивність, незалежність, захищеність, цілісність тощо) і вимогам до апаратного забезпечення. Методологія має бути достатньо гнучкою, доступною розробникам із різним досвідом проектування, що використовують різні моделі даних і різне програмне забезпечення СКБД.

Методологія проектування баз даних визначає:

- процес проектування;
- методику виконання розрахунків і критеріїв оцінювання альтернативних рішень на кожному етапі проектування;
- інформаційні вимоги як вихідні дані для процесу проектування;
- засоби опису вихідних даних і відображення результатів кожного етапу проектування.

1.1 Процес проектування.

Для баз даних можна застосувати ітераційне низхідне проектування. Процес проектування добре структурований, оскільки кожний його етап завершується певним результатом, а також тому, що допускається ітераційне повторення попередніх етапів, якщо отриманий результат не відповідає вимогам замовника або системним

вимогам. Це дає можливість переглядати й змінювати проектні рішення на будь-якому етапі.

З проектуванням тісно пов'язане експертне оцінювання проекту. Мета експертизи - знайти помилки й виправити їх на ранніх етапах проектування. Зазвичай експертиза виконується після завершення кожного з етапів.

Етап проектування БД вважається одним із самих складних етапів створення БД, який не має явно вираженого початку й закінчення. У порівнянні з аналізом вимог до БД або розробкою додатків, проектування БД, на думку багатьох провідних фахівців, є невдало структурованим завданням. Якщо всі етапи створення БД перекриваються один з одним у своїй послідовності, то етап проектування перекривається з усіма іншими етапами. Проектування починається з моменту прийняття стратегічних рішень і триває на етапах реалізації й тестування.

Процес проектування БД охоплює кілька основних сфер:

- проектування об'єктів БД (таблиці, подання, індекси, тригери, збережені процедури, функції, пакети) для подання даних ПО в БД;
- проектування інтерфейсу взаємодії з БД (форми, звіти й т.д.), тобто проектування додатків, які будуть супроводжувати дані в БД і реалізовувати питально-відповідні відношення на цих даних;
- проектування БД під конкретне обчислювальне середовище або інформаційну технологію (архітектура "клієнт-сервер", паралельні архітектури, розподілене обчислювальне середовище);
- проектування БД під призначення системи (інтелектуальний аналіз даних, OLAP, OLTP і т.д.).

1.2 Критерії оцінювання

Оцінювання необхідне для ухвалення рішень за наявності альтернатив. Труднощі у визначенні критеріїв і виборі альтернатив пов'язані з тим, що часто розробляється кілька проектів структури бази даних і потрібно оцінити, який з них є кращим. Зробити це буває досить складно.

Критерії є кількісні (час обробки запитів, вартість операцій маніпулювання даними, витрати пам'яті тощо) та якісні (гнучкість, адаптивність, сприйнятливність та сумісність).

1.3 Інформаційні вимоги

Визначаючи вимоги до інформації, врахуйте, що є інформація, яка стосується структури даних (опис даних та зв'язків безвідносно до конкретних способів їхнього використання й обробки), та інформація про спосіб використання даних (опис вимог до обробки даних).

Засоби опису

Це мовні засоби, призначені для опису результатів виконання кожного етапу проектування. А саме, йдеться про такі засоби.

Природна мова, якою строго означуються всі необхідні для опису результатів проектування поняття. Використовується, як правило, на етапі визначення стратегії.

Стандартні форми, анкети та бланки. Використовуються переважно на етапі аналізу.

Спеціальні формалізовані мови концептуального моделювання (семантичні мережі, числення предикатів та ER-мови). Використовуються переважно на етапі концептуального моделювання.

Формалізована мова означення даних (МОД) і мова маніпулювання даними (ММД). Використовуються на етапі логічного проектування. Зазвичай з цією метою застосовують мову SQL.

2 Поняття та класифікація моделей даних

2.1 Дані та їхня семантика

Слово «дані» походить від латинського «datum» – факт, проте дані не завжди відповідають конкретним чи навіть реальним фактам. Іноді вони неточні або описують те, чого насправді не існує. Даніни ми вважатимемо опис будь-якого явища (чи

ідеї), що викликає зацікавленість через певні потреби. З даними нерозривно пов'язані їхня інтерпретація (або семантика), тобто той вміст, який їм приписується.

Дані описуються тією чи іншою мовою і фіксуються на певному носії (скажімо, папері). Зазвичай далі (факти) та їхня семантична інтерпретація фіксуються спільно. Проте в деяких випадках дані й інтерпретація розділяються. Так у зведеній статистичній таблиці зверху, в її шапці, міститься інформація стосовно того, чим є дані (заголовки таблиці, описова інформація, назви стовпців тощо), а нижче розташовуються власне рядки чисел.

Застосування комп'ютерів для введення й обробки даних сприяло ще більшому розділенню даних та їхньої інтерпретації. Оскільки комп'ютери оперують даними як такими, велика частіша інтерпретуючої інформації взагалі явно не фіксується. Програма одержує певні вхідні дані, обробляє їх і видає результат у вигляді сукупності вихідних даних.

З розвитком обчислювальної техніки з'явилася можливість фіксувати за допомогою комп'ютерів семантику даних, тобто закладати інтерпретацію в програми, що оперують даними. Проте за умов спільного використання даних різноманітними прикладними програмами цей підхід можна застосовувати лише частково. Інакше процес написання програм, які містять схожі, хоча й не ідентичні механізми інтерпретації, стає неефективним. У подібній ситуації доцільно зв'язувати дані з механізмами інтерпретації, що має забезпечувати уніфікованість подання семантичної інформації. У результаті змінюється роль даних: їх уже не можна розглядати лише як сукупність бітів, вони отримують певне семантичне навантаження.

Засоби інтерпретації мають бути гнучкими, аби разом з незмінними параметрами даних відображувати й аспекти їхньої еволюції. Цього досягають двома способами. По-перше, можна відтворювати різні погляди на одні й ті самі дані. Наприклад, розглядати певну особу з точки зору кадрової системи як службовця, з погляду виробничих інтересів – як виконавця робіт, а з боку медичного обслуговування – як пацієнта. По-друге, різні дані можна подавати одноманітно. Так, адміністратори, клерки, агенти, секретарі незалежно від роду своєї діяльності можуть розглядатися в кадровій системі як службовці.

2.2 Моделювання даних

Існує багато типів моделей - фізичні, математичні, економічні тощо, які відображують різні аспекти реального світу. Модель даних відображує уявлення про реальний світ. Проте важливо, аби обсяг знань і семантика даних, відтворені в моделі, були адекватні способу використання даних. Вважатимемо, що модель даних – це сукупність структури даних, операцій над ними (операції маніпулювання даними) та обмежень цілісності. Іншими словами, модель визначає, в який спосіб відбувається об'єднання даних у структури різної складності які існують обмеження на значення даних і як здійснюється оперування цими даними.

Основою для будь-якої структури даних є відображення елементарної одиниці даних у вигляді такої трійки: об'єкт, властивість об'єкта, значення властивості. Сукупність взаємопов'язаних між собою елементарних одиниць даних може відображуватися різноманітними способами, що приводить до формування різних структур, а відтак - різних моделей даних. Моделі даних поділяються на два класи; сильно та слабо типізовані.

У сильно типізованих моделях усі дані мають належати до певної категорії, або типу. Якщо дані не підпадають під жодну з категорій, їх потрібно типізувати штучно. Деякі моделі будуються у такий спосіб, що категорії визначаються наперед і не можуть змінюватися динамічно.

Сильно типізовані моделі мають значні переваги, бо дають змогу побудувати абстракції властивостей даних і дослідити їх у термінах категорій. Більшість моделей, що використовуються в автоматизованих системах, зокрема й базах даних, належать до сильно типізованих.

Для слабо типізованих моделей належність даних до тієї чи іншої категорії немає жодного значення. Категорії використовуються настільки, наскільки це доцільно в кожному конкретному випадку. Окремі дані можуть існувати як незалежно, так і у зв'язку з іншими. Інформація про категорії розглядається як додаткова.

На відміну від сильно типізованих моделей, слабо типізовані забезпечують інтеграцію даних і категорій. Найкращі можливості такої інтеграції надаються численним предикатів, яке у багатьох моделях даних використовується для зображення знань, що не підтримуються базовими засобами моделювання.

2.3 Ієрархічна модель

У ієрархічній моделі зв'язку між даними можна описати за допомогою упорядкованого графа (або дерева). Спрощено представлення зв'язків між даними в ієрархічній моделі показане на рисунку 1. Для опису структури (схеми) ієрархічної БД на деякій мові програмування використовується тип даних «дерево». Тип «дерево» схожий з типами даних «структура» мов програмування C++ і «запис» мови Delphi. У них допускається вкладеність типів, кожен з яких знаходиться на деякому рівні.

Тип «дерево» є складеним. Він включає підтипи («під дерева»), кожен з яких, у свою чергу, є типом «дерево». Кожен з типів «дерево» складається з одного «кореневого» типу впорядкованого набору (можливо порожнього) підлеглих типів. Кожен з елементарних типів, включених в тип «дерево», є простим або складеним типом «запис». Простий «запис» складається з одного типу, наприклад, числового, а складений «запис» об'єднує деяку сукупність типів, наприклад, ціле, рядок символів і покажчик (посилання).

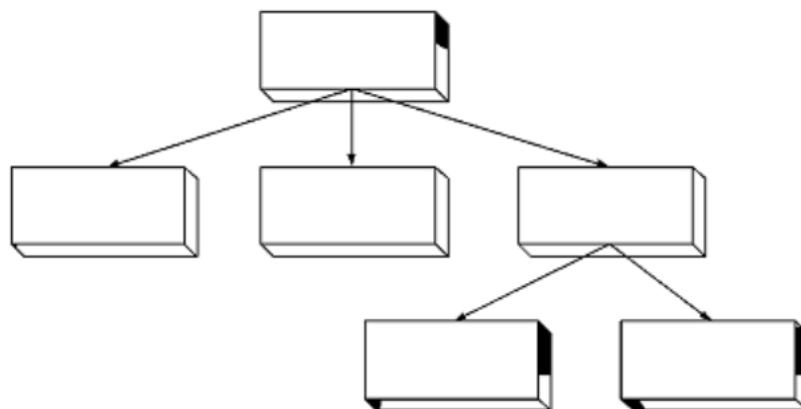


Рисунок 1. Представлення зв'язків в ієрархічній моделі

Кореневим називається тип, який має підлеглих типів і сам не є підтипом. Підлеглий тип (підтип) є нащадком по відношенню до типу, який виступає для нього в ролі предка (батька).

Ієрархічна БД є впорядкованою сукупністю екземплярів даних типу «дерево» (дерева), що містять екземпляри типу «запис» (записи). Часто стосунки спорідненості між типами переносять на стосунки між самими записами. Поля записів зберігають власне числові або символічні значення, які складають основний вміст БД. Обхід всіх елементів ієрархічною БД звичайно виробляється зверху вниз і зліва направо.

Для організації фізичного розміщення ієрархічних даних в пам'яті ЕОМ можуть використовуватися наступні групи методів:

- представлення лінійним списком з послідовним розподілом пам'яті (адресна арифметика, лівоспискові структури);
- представлення лінійними списками (методи, що використовують покажчики і довідники).

До основних операцій маніпулювання ієрархічно організованими даними відносяться наступні:

- пошук вказаного екземпляра БД;
- перехід від одного дерева до іншого;
- перехід від одного запису до іншої усередині дерева;
- вставка нового запису у вказану позицію;
- видалення поточного запису і так далі

Відповідно до визначення типу «дерево», можна укласти, що між предками і нащадками автоматично підтримується контроль цілісності зв'язків. Основне правило контролю цілісності формулюється таким чином: нащадок не може існувати без батька, а у деяких батьків може не бути нащадків. Механізми підтримки цілісності зв'язків між записами різних дерев відсутні.

До достоїнств ієрархічної моделі даних відносяться ефективне використання пам'яті ЕОМ і непогані показники часу виконання основних операцій над даними. Ієрархічна модель даних зручна для роботи з ієрархічно впорядкованою інформацією.

Недоліком ієрархічної моделі є її громіздкість для обробки інформації з досить складними логічними зв'язками, а також складність розуміння для звичайного користувача.

2.4 Мережна модель

Мережна модель даних дозволяє відображувати всілякі взаємозв'язки елементів даних у вигляді довільного графа, узагальнюючи тим самим ієрархічну модель даних (рисунок 2).

Для опису схеми мережної БД використовується дві групи типів: «запис» і «зв'язок». Тип «зв'язок» визначається для двох типів «запис»: предка і нащадка. Змінні типу «зв'язок» є екземплярами зв'язків (рисунок 3).

Мережна БД складається з набору записів і набору відповідних зв'язків. На формування зв'язку особливих обмежень не накладається. Якщо в ієрархічних структурах запис-нащадок міг мати лише один запис-предка, то в мережній моделі даних запис-потомок може мати довільне число записів-предків (звідних батьків).

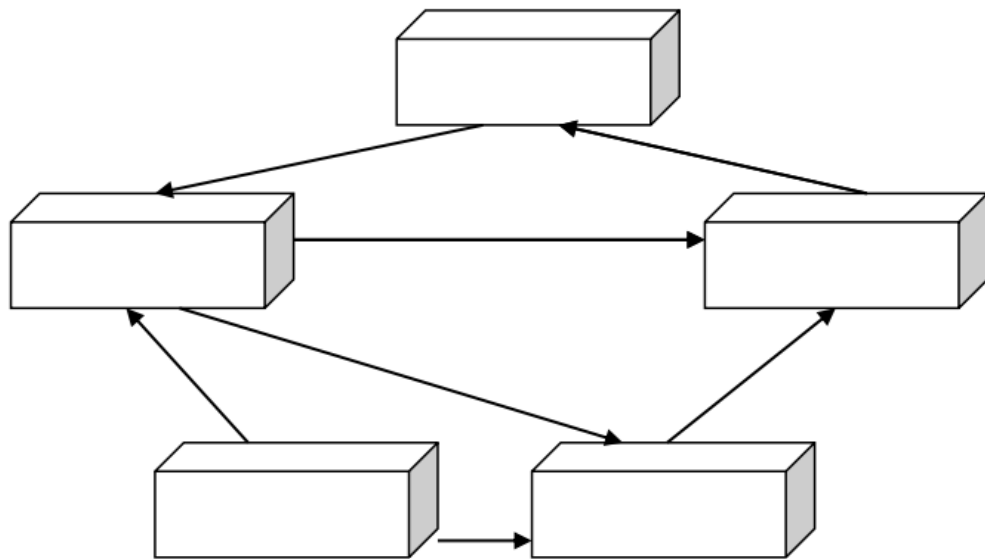


Рисунок 2. Представлення зв'язків в мережній моделі

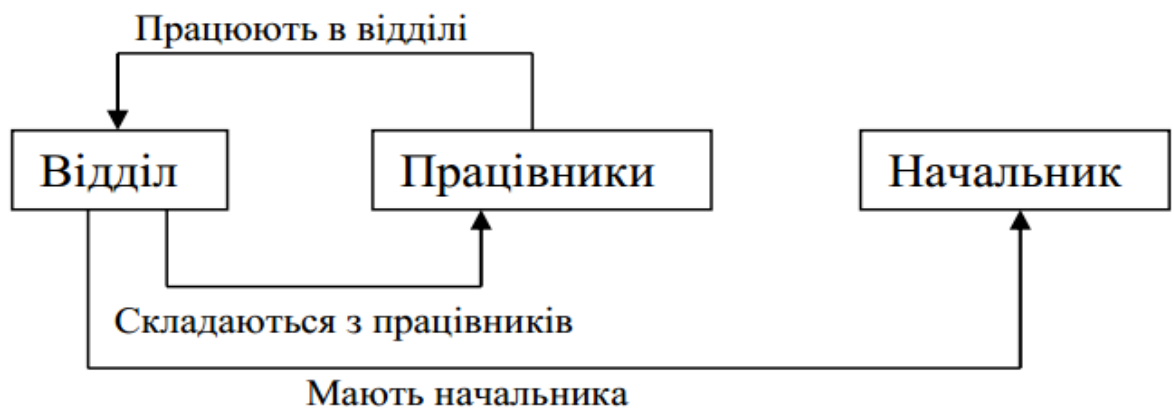


Рисунок 3. Приклад схеми мережевої БД

Фізичне розміщення даних в базах мережного типу може бути організоване практично тими ж методами, що і в ієрархічних базах даних.

До найважливіших операцій маніпулювання даними баз мережевого типу можливо віднести наступні:

- пошук запису в БД;
- перехід від предка до першого нащадка;
- перехід від нащадка до предка;
- створення нового запису;
- видалення поточного запису;
- оновлення поточного запису;
- включення запису в зв'язок;
- виключення запису із зв'язку;
- зміна зв'язків і так далі.

Достоїнством мережної моделі даних є можливість ефективної реалізації за показниками витрат пам'яті і оперативності. Порівняно з ієрархічною моделлю мережева модель надає великі можливості в сенсі допустимості утворення довільних зв'язків.

Недоліком мережевої моделі даних є висока складність і жорсткість схеми БД, побудованої на її основі, а також складність для розуміння і виконання обробки інформації в БД звичайним користувачем. Крім того, в мережевій моделі даних ослаблений контроль цілісності зв'язків унаслідок допустимості встановлення довільних зв'язків між записами.

Контрольні питання:

1. Які кроки включає процес розробки бази даних?
2. Що таке предметна область?
3. Дайте означення моделі предметної області.
4. Які моделі баз даних розглядаються на концептуальному рівні?
5. Опишіть ієрархічну модель даних.
6. Опишіть мережну модель даних.

Лекція № 5

з навчальної дисципліни Організація баз даних

Тема Побудова концептуальної моделі даних.

Мета заняття Ввести основні поняття та правила побудови концептуальної (семантичної моделі даних).

Час – 2 години

План проведення лекції

1 Фундаментальні поняття семантичної моделі даних

1.1 Об'єкти

1.2 Атрибути

1.3 Ключі

1.4 Зв'язки між об'єктами

Література

1. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
2. Трофименко О. Г. Організація баз даних : Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
3. Харів Н. О. Базы даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
4. Ярцев В.П. Організація баз даних та знань: навчальний посібник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

ВСТУП

Після завершення початкових етапів життєвого циклу бази даних, таких, як: планування розробки БД, визначення вимог до системи, збір й аналіз вимог користувачів, які дуже специфічні для кожної інформаційної системи, починається процес проектування БД. Цей процес містить у собі повний цикл розробки бази даних і починається з концептуального проектування, у результаті виконання якого створюється повна концептуальна модель даних.

1 Фундаментальні поняття семантичної моделі даних

Для нормального функціонування інформаційної системи необхідно, щоб концептуальна модель адекватно відображала реалії тієї предметної області, для якої вона розробляється. *Фундаментальними реаліями в концептуальному моделюванні є дані з їхніми властивостями й зв'язку між ними.* Причому таке моделювання крім інформації про наявність певних даних і зв'язків потребує інформацію щодо значення, семантичного змісту, незалежного від подання в ЕОМ. Методології, що дозволяють ефективно відображати існуючу значеннєву змістовність реальності в конструкції моделі, ставляться до так називаних *семантичних методологій*. Методології проектування, засновані на ідеях семантичного моделювання, ставляться до спадних методологій, оскільки вони починаються на вищому рівні абстракції й закінчуються на рівні абстракції, представленому конкретною структурою макета бази даних.

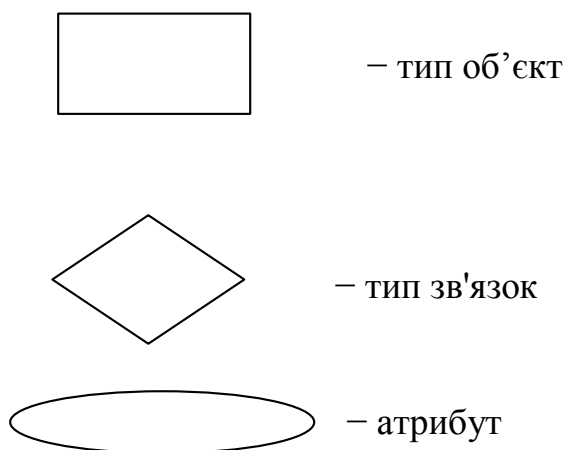
З початку сімдесятих років було запропоновано кілька концептуальних семантичних моделей даних, які підходили до проблеми моделювання даних для проектування БД із різних точок зору. Найбільш популярною семантичною моделлю стала *модель "сутність - зв'язок" (ER-модель)*, запропонована П. Ченом в 1976 році, що з тих пор неодноразово вдосконалилася самим Ченом і багатьма іншими фахівцями. Дана модель даних ставиться до високорівневих моделей і базується на ряді концепцій, використовуваних для опису структури бази даних.

Ми будемо користуватися методологією, скомбінованої з моделі Чена й понять об'єктно-орієнтованого моделювання. Модель Чена формує базис концептуальної моделі даних, а об'єктно-орієнтоване моделювання вносить деякі істотні вдосконалення. Методологія моделювання даних, що представляє комп'ютерне відображення взаємозалежних категорій реального миру у вигляді "об'єктів", що володіють певними "посвідченнями особи" й атрибутами, може з названа об'єктно-орієнтованою.

Семантичне моделювання даних у першу чергу приділяє увагу структурі даних, а предметом інтересів об'єктно-орієнтовного підходу головним чином служить поводження об'єктів даних і способи маніпуляції даними, які можна з б зосередити в можливостях мови (запитах, обчисленнях, відновленні даних). Зближення цих двох областей відбулося, коли дослідники почали застосовувати поняття об'єктно-орієнтовних мов до семантичних структур даних. У результаті чого й з'явилося поняття об'єктно-орієнтовної бази даних. На цьому стику дисциплін переважає об'єктно-орієнтовна термінологія, тому тут говориться про об'єкти, а не про сутності, як це прийнято в семантичній термінології Чена. Крім того, об'єктно-орієнтовні мови виділяють кілька понять, відсутніх у вихідній моделі Чена: "посвідчення особи" об'єкта, ієрархія над об'єктів і під об'єктів, спадкування.

Головними елементами семантичної моделі даних є *типи об'єктів, їхні атрибути й типи зв'язків*. Типи об'єктів часто представляють у вигляді іменників, а типи зв'язків - у вигляді дієслів.

Семантична модель предметної області зображується у вигляді діаграми з урахуванням прийнятих позначень для її елементів:



1.1 Об'єкти

У процесі концептуального проектування предметна область розглядається як об'єктна система, що має наступні складові:

- об'єкт;
- час;
- засіб;
- зв'язок.

Об'єкти позначають речі, які користувачі вважають важливими в моделюємії частини реальності. Об'єкт - це те, про що накопичується інформація в інформаційній системі й що може бути однозначно ідентифіковано. Об'єкти можуть бути атомарними або складеними. Для складеного об'єкта визначається його внутрішня структура. Кожен об'єкт у конкретний момент часу характеризується своїм станом. Цей стан визначається за допомогою обмеженого набору засобів і зв'язків з іншими об'єктами. Складова час дозволяє моделювати динамічні системи.

Об'єкт-тип (надалі просто об'єкт) характеризується незалежним існуванням і представляє безліч об'єктів реального миру з однаковими властивостями. Окремі об'єкти, які входять у даний тип, називають екземплярами об'єкта. Розрізняють реальні й концептуальні об'єкти. Прикладами об'єктів можуть бути люди, товари, будинку, деталі, книги й т.д. Це реальні об'єкти. Концептуальними об'єктами будуть навички, організації, ділові операції, штатний розклад і т.д. Кожен об'єкт має ім'я й зображується на діаграмах у вигляді прямокутника, а екземпляр об'єкта - у вигляді крапки в прямокутнику даного об'єкта (див. рис. 1.1).



Рисунок 1.1. Позначення об'єкта на ER-діаграмі

У концептуальній моделі можуть бути присутнім об'єкти двох видів: *сильні й слабкі*. Об'єкт, існування якого не залежить від існування іншого об'єкта, називається сильним. Слабкий об'єкт, навпаки, це той об'єкт, що є залежним від деякого іншого об'єкта, тобто він не може існувати в моделі, якщо в ній не існує цей інший об'єкт. Оскільки в концептуальній моделі все взаємозалежно, те той самий об'єкт стосовно одного об'єкта може бути сильним, а стосовно іншого - слабким.

1.2 Атрибути

Атрибут - це пойменована характеристика об'єкта, за допомогою якої моделюється його властивість. Кожному об'єкту властиві свої атрибути. Наприклад, об'єкт КНИГА повинен мати такі атрибути: найменування, автор, видавництво, рік видання. У людини є ім'я, дата народження, ріст, вага, підлога, колір волосся, мати, батько й так далі. Якщо для деякого екземпляра об'єкта значення деякого атрибута не визначено, то цей атрибут для даного екземпляра об'єкта має порожнє значення.

На діаграмах атрибути об'єкта поєднуються з ним лініями (див. рис. 1.2).

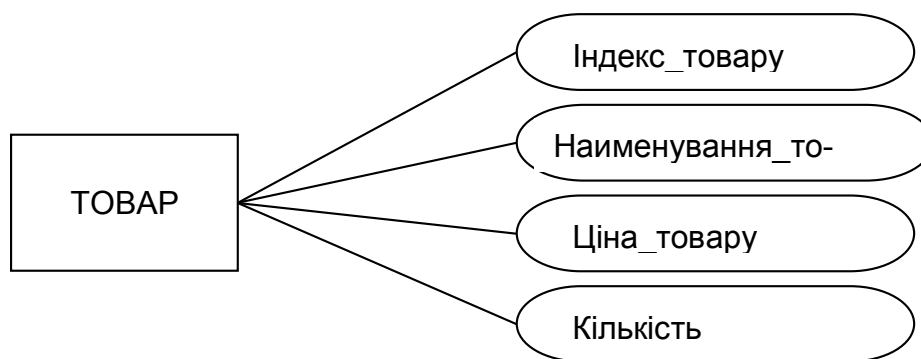


Рисунок 1.2. Діаграма подання об'єкта ТОВАР і його атрибути

Значення кожного атрибута вибираються з відповідної множини значень, що включає всі потенційні значення, які можуть бути привласнені атрибуту. Ця множина значень називається *доменом*. Так, наприклад, допустимо, що кількість товару визначається в одиницях і може варіюватися від 0 до 1 000 одиниць. Отже, набір припустимих значень для даного атрибута можна визначити як набір цілих чисел від 0 до 1000.

Об'єкт й екземпляр об'єкта можуть бути визначені в такий спосіб:

Об'єкт: СТУДЕНТ (ФІО, Група, Рік_народження)

Екземпляр об'єкта: (Петров П.И., КС – 13 – 1, 1974)

Атрибути діляться на прості й складні. Прості атрибути не можуть бути розділені на більше дрібні компоненти. Наприклад, атрибут *Кількість об'єкта* ТОВАР є простим атрибутом. Простий атрибут ще називають атомарним. Якщо ж атрибут можна розбити на більше дрібні складові, то такий атрибут називається складним. Гарним прикладом складеного атрибута є *Дата народження* (Рік, Місяць, Число). Кожен з його компонентів можна використати окремо. Але в тих ситуаціях, де не потрібно окремо використати складові частини такого атрибута, його можна розглядати як простий атрибут і визначати, допустимо, як строковий тип. Якщо атрибут є складним, то на діаграмах його атрибути-компоненти приєднуються до нього лініями (див. рис. 1.3).

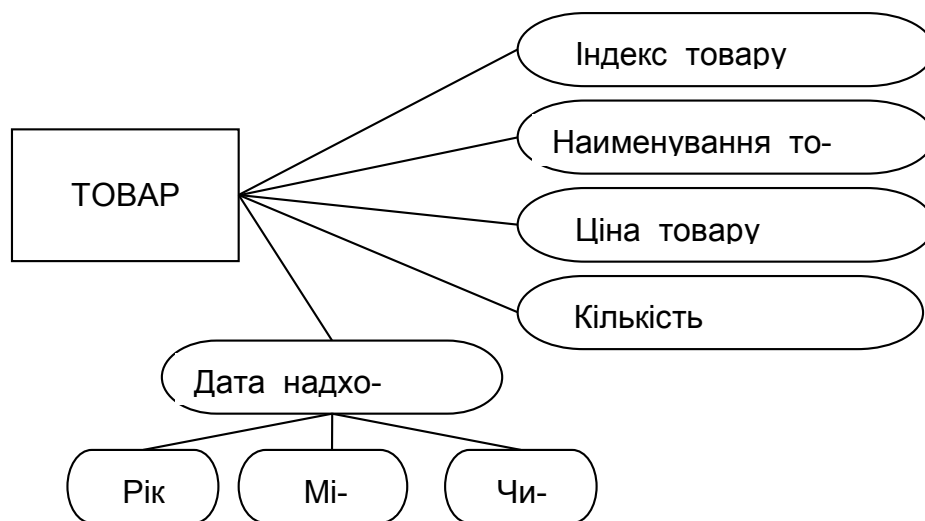


Рисунок 1.3. Діаграма подання об'єкта ТОВАР зі складним атрибутом Дата_надходження

Значення атрибутів можуть часто мінятися, у той час як описуваний ними об'єкт залишається тим же самим. Так, в екземпляра об'єкта ТОВАР може змінитися значення атрибута Кількість, але сам об'єкт залишиться тим же.

Якщо атрибут кожного окремого екземпляра об'єкта може мати тільки одне значення, то такий атрибут називається *однозначним*. Наприклад, атрибути Прізвище, Рік народження, Ріст кожного екземпляра об'єкта СТУДЕНТ можуть мати тільки одне

значення. Більшість атрибутів ставляться саме до цього різновиду. Деякі атрибути можуть мати кілька значень для кожного екземпляра об'єкта. Наприклад, деяка фірма може мати кілька телефонних номерів або кілька рівнозначних представників. Такий атрибут є *багатозначним*. Багатозначний атрибут на діаграмах обводиться подвійним контуром.

Атрибут може бути *базовим*, а може бути *похідним*. Похідним вважається такий атрибут, значення якого визначається за значенням іншого атрибута або інших атрибутів. Наприклад, значення атрибута Вік студента можуть бути обчислені за значеннями атрибута Дата народження об'єкта СТУДЕНТ. Для того щоб задати атрибут, потрібно дати йому ім'я, описати його й задати множину припустимих значень, тобто надати специфікацію.

1.3 Ключі

Серед атрибутів особливе положення займають ті, за допомогою яких можна ідентифікувати екземпляр об'єкта. Такі атрибути називаються *ключами*. Атрибут або кілька атрибутів, значення яких унікальним образом ідентифікують кожен екземпляр об'єкта, є потенційним ключем даного об'єкта. Потенційних ключів може бути кілька.

Наприклад, екземпляр об'єкта

ФАКУЛЬТЕТ (Код факультету, Назва_факультету, ПІБ_Декана)

може однозначно ідентифікуватися кожним з перших двох зазначених атрибутів. Третій атрибут не рекомендується використати в якості ідентифікаційного, оскільки не можна гарантувати відсутність повного збігу значень атрибута ПІБ_Декана для декількох екземплярів даного об'єкта. Розглянемо об'єкт:

СТУДЕНТ (Номер залікової книжки, ПІБ_студента, Дата_народження).

Із трьох перерахованих атрибутів у наведеному прикладі тільки атрибут Номер_залікової_книжки однозначно ідентифікує кожен екземпляр об'єкта СТУДЕНТ. Атрибут Дата_Народження, навпроти, не може служити ключем, тому що будь-яка дана дата може бути вдень народження декількох різних людей. Один з потенційних ключів може бути обраний як *первинний ключ*. Звичайно як первинний ключ вибира-

ється той, котрий має найменшу довжину. Інші потенційні ключі називаються *альтернативними*. У розглянутому вище прикладі атрибут Код_факультету має меншу довжину, чим атрибут Назва_факультету, тому його варто вибрати як первинний ключ.

Той факт, що атрибут служить первинним ключем, відзначається його підкресленням. Ідентифікацію деяких об'єктів іноді доводиться здійснювати за допомогою складених ключів, які включають кілька атрибутів.

Наприклад, об'єкт:

ЛІКУВАННЯ (ФІО лікаря, ФІО пацієнта, Дата призначення, Ліки)

однозначно ідентифікувати можна тільки складеним ключем:

(ФІО лікаря, ФІО пацієнта, Дата призначення).

З іншого боку, якщо потрібно, завжди можна створити ідентифікаційний номер, однозначність якого буде закладена в системі. Щоб уникнути зайвої деталізації на діаграмах часто атрибути зовсім не зображують або зображують тільки первинні атрибути, опускаючи інші.

1.4 Зв'язки між об'єктами

Два об'єкти можуть бути зв'язані між собою. Подібний зв'язок здійснюється через зв'язок екземплярів одного об'єкта з екземплярами іншого об'єкта, створюючи набір екземплярів зв'язку між двома об'єктами, що називається *типом зв'язку*. Кожному типу зв'язку привласнюється ім'я, що повинне представляти його функцію.

Розглянемо об'єкти ВИКЛАДАЧ і КУРС. Між цими об'єктами можна визначити зв'язок ЧИТАЄ, зіставивши кожному викладачеві ту дисципліну, по якій він читає лекції, або, навпаки, кожній дисципліні - викладача. Зв'язок ЧИТАЄ складений з множини пар, у кожній з яких викладач - з об'єкта ВИКЛАДАЧ, а дисципліна - з об'єкта КУРС (див. рис. 1.4).

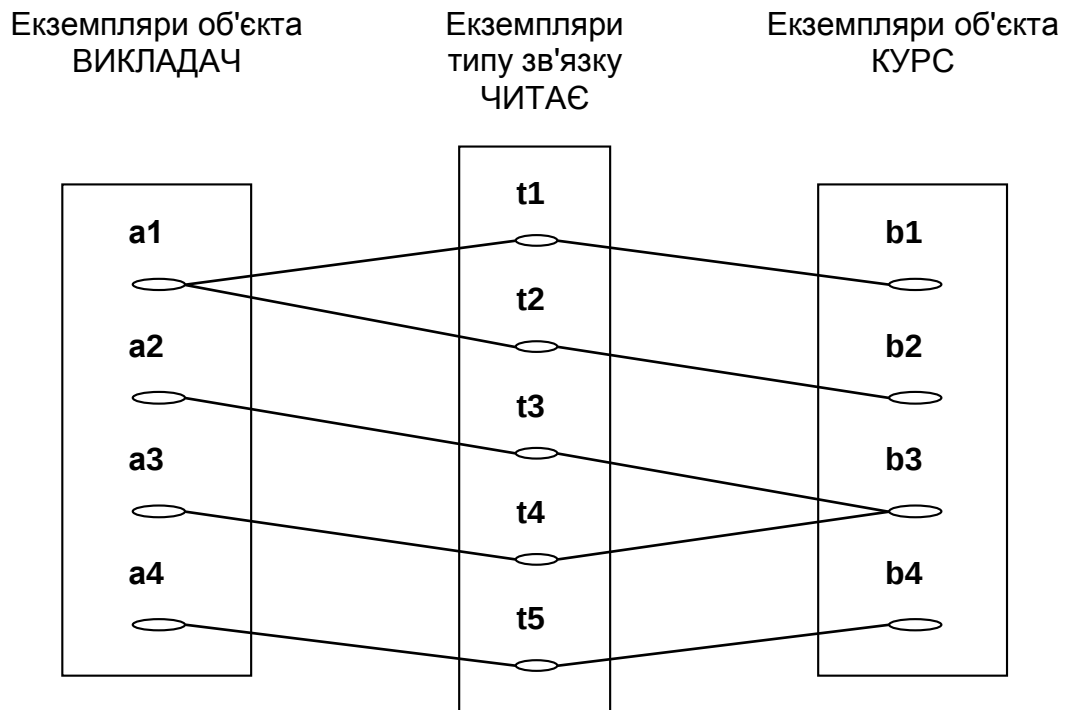


Рисунок 1.4. Екземпляри типу зв'язку ЧИТАЄ

Отримана структура сама по собі є об'єктом, що складається з пар екземплярів, узятих із двох об'єктів, зв'язаних між собою. Об'єкт ЧИТАЄ, отриманий шляхом зв'язку між об'єктами ВИКЛАДАЧ і КУРС, називається *складеним об'єктом*.

Описана ситуація на діаграмах має своє графічне зображення, де тип зв'язку позначається у вигляді ромбика із зазначеним на ньому ім'ям зв'язку, що з'єднаний лініями з об'єктами, зв'язаними з ним (див. рис. 1.5).



Рисунок 1.5. Діаграма типу зв'язку ЧИТАЄ

У зв'язку можуть брати участь не два, а більша кількість об'єктів, які в цьому випадку є учасниками цього зв'язку. Кількість учасників деякого зв'язку називається *ступенем зв'язку*. Зв'язки між двома об'єктами називаються *бінарними*. Крім бінарних зв'язків існують й інші типи зв'язків:

- тернарні - між трьома об'єктами;
- кватернарні - між чотирма об'єктами;

- N-арні - між N об'єктами.

У більшому числі випадків проектування БД можна обмежитися розглядом бінарних зв'язків. Для характеристики властивостей зв'язку, також можна використати атрибути.

Потужність зв'язку. Важливою характеристикою зв'язку є її потужність, що позначає максимальну кількість екземплярів одного об'єкта, пов'язаних з одним екземпляром іншого об'єкта. Наприклад, якщо допустити, що в людини може бути тільки один чоловік, то потужність зв'язку ОДРУЖЕНІ буде дорівнює одному в кожному напрямку (див. рис. 1.6).



Рисунок 1.6. Діаграма зв'язку ОДРУЖЕНІ із вказівку мінімальної й максимальної потужності.

Іноді крім максимальної потужності корисно визначати й мінімальну потужність. У розглянутому прикладі, не виключаються самотні чоловіки й жінки, тому мінімальна потужність дорівнює нулю в кожному напрямку.

Деякі зв'язки не мають конкретного значення максимальної потужності. Наприклад, викладач може читати не один курс, а, можливо, більше. Таку потужність позначають: 1,*, де 1 позначає мінімальну потужність, а * позначає "багато" (існує й інший спосіб позначення "багато" - замість * ставиться N). З іншого боку, якщо допустити, що кожен даний курс читається одним і тільки одним викладачем, то потужність у зворотному напрямку буде 1,1.

Максимальна потужність є значно більше важливим поняттям, чим мінімальна. Тому для спрощення діаграм мінімальна потужність вказується тільки тоді, коли це необхідно.

Показник кардинальності. Зв'язок, що має максимальну потужність в одному з напрямків, рівну одному, називається *функціональним в цьому напрямку*.

В останній розглянутій ситуації зв'язок між викладачем і курсом є функціональним в напрямку від курсу до викладача. Це означає, що, знаючи курс, можна однозначно визначити викладача, що читає його. Це відношення не є функціональним у зворотному напрямку, оскільки викладач може читати кілька курсів.

Для того щоб указати кількість можливих зв'язків для кожного екземпляра об'єкта, що бере участь у зв'язку, використовують показник кардинальності. *Показники кардинальності* зв'язків між об'єктами визначаються, насамперед, установленими на виробництві правилами й ставляться до бізнесів-правил. Для бінарних зв'язків показник кардинальності може мати наступні значення: "один до одного" (1:1), "один до багатьох" (1:N), "багато до багатьох" (M:N).

Якщо максимальна потужність в обох напрямках дорівнює одному, то вона називається зв'язком "один до одного" (1:1). Наприклад, на факультеті може бути один декан, і назад, той самий декан може керувати тільки одним факультетом, що може бути позначене в такий спосіб:

ФАКУЛЬТЕТ $\longleftrightarrow$ ДЕКАН

Якщо максимальна потужність в одному напрямку дорівнює одному, а в іншому - декілька, то зв'язок називається "один до багатьох" (1:N). Наприклад, у групі вчиться багато студентів, але кожен студент учиться тільки в одній групі:

ГРУПА $\longleftrightarrow$ СТУДЕНТ.

Концептуальна модель подібного зв'язку наведена на рисунку 1.7.

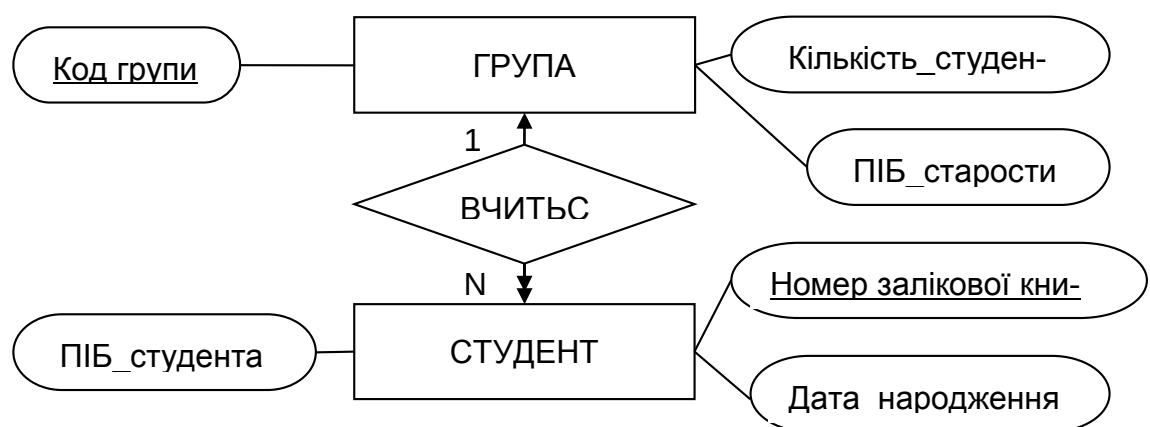


Рисунок 1.7. Діаграма типу зв'язку ВЧИТЬСЯ із вказівкою показника кардинальності

На діаграмі використано два способи позначення виду бінарного зв'язку: символічна (з боку об'єкта ГРУПА вихід зв'язку позначений символом 1, а з боку об'єкта

СТУДЕНТ - символом N) і стрілками (у напрямку, де максимальна потужність дорівнює багатьом, проставлена подвійна стрілка, а з боку, де вона дорівнює одиниці - одинарна). Реально при побудові діаграм вибирають один з них.

Якщо максимальна потужність в обох напрямках дорівнює багатьом, то такий зв'язок ставиться до типу "багато до багатьох" (M:N). Наприклад, викладач працює в різних групах, і в одній і тій же групі працюють різні викладачі:

ВИКЛАДАЧ $\longleftrightarrow$ ГРУПА

Зв'язок між об'єктами здійснюється за допомогою атрибутів. Наприклад, розглянемо два об'єкти:

Об'єкт:	СТУДЕНТ	ГРУПА
Атрибути:	<u>Номер залікової книжки</u> ПІБ студента	<u>Код групи</u> Кількість_студентів ПІБ старости

Зараз ці два об'єкти не зв'язані між собою. Для їхнього зв'язку в число атрибутів об'єкта СТУДЕНТ необхідно додати код групи, у якій він учить, і значення якого буде використано для зв'язку екземпляра одного об'єкта з екземпляром іншого об'єкта.

Атрибут об'єкта - окремий випадок зв'язку одного об'єкта з іншим об'єктом. При нормальному використанні атрибути мають функціональні зв'язки в напрямку від об'єкта до атрибута. Це означає, що значення атрибута однозначно визначено для кожного екземпляра об'єкта. Наприклад, у кожної людини є рівно одна дата народження й одна мати. Максимальна потужність зв'язку з боку атрибута в такій ситуації завжди дорівнює одному, тому в діаграмах її можна опустити. Якщо немає необхідності використати атрибут як об'єкт, що бере участь в інших зв'язках, то для його зображення на діаграмах застосовують уже розглянутий короткий запис.

Ступінь участі. У методології проектування БД досить важливою характеристикою типів зв'язків між об'єктами є ступінь участі об'єкта у зв'язку. Якщо кожен екземпляр деякого об'єкта обов'язково повинен брати участь у зв'язку, то ступінь участі цього об'єкта в даному зв'язку є повною. Про такий об'єкт ще говорять, що його *клас приналежності обов'язковий*. Якщо ж для об'єкта припустима неучасть його деяких екземплярів у зв'язку, то ступінь участі даного об'єкта в цьому зв'язку є частковою, а

його клас приналежності - *необов'язковий*. Розглянемо зв'язок, зображений на рисунку 1.4, між об'єктами ВИКЛАДАЧ і КУРС. Відомо, що до складу викладачів кафедри входять фахівці різних категорій: асистенти, старші викладачі, доценти, професори. Асистентам, як правило, не поручається читання лекцій, тому ступінь участі об'єкта ВИКЛАДАЧ у зв'язку ЧИТАЄ є частковою. У той же час для якісної підготовки фахівців необхідно, щоб всі дисципліни, передбачені навчальним планом, були вивчені. Припустимо, що для всіх дисциплін заплановане читання лекцій. Якщо це так, то ступінь участі об'єкта КУРС у зв'язку ЧИТАЄ є повною (див. рис. 1.8). Однак якщо навчальний план містить вивчення дисциплін, по яких читання лекцій не передбачено, то в цій ситуації ступінь участі об'єкта КУРС у зв'язку ЧИТАЄ буде частковою. На діаграмах учасники зв'язку з повною участю з'єднуються зі знаком зв'язку подвійною лінією, а учасники зв'язку із частковою участю - одинарною лінією.



Рисунок 1.8. Діаграма зв'язку з повною й частковою участю об'єктів

Рекурсивний зв'язок. Рекурсивний зв'язок - це особливий вид зв'язку, у якій ті самі екземпляри об'єкта беруть участь кілька разів й у різних ролях. Наприклад, один зі співробітників кафедри є її завідувачем. Різним ролям тут привласнюються різні імена (див. рис. 1.9).

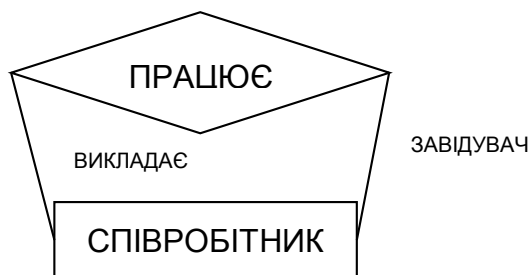


Рисунок 1.9. Діаграма рекурсивного зв'язку

Питання та завдання до контролю знань студентів

- 1 Охарактеризуйте життєвий цикл бази даних. Які основні етапи він включає?
- 2 Назвіть основні цілі процесу проектування бази даних і дайте характеристику його фазам.
- 3 Які стратегії тестування прикладних програм інформаційних систем існують?
- 4 Поясніть своїми словами зміст термінів:
 - об'єкт;
 - атрибут;
 - бінарний зв'язок;
 - "один-до-багатьох";
 - потужність;
 - показник кардинальності;
 - ступінь участі;
 - ключ;
 - рекурсивний зв'язок;
 - складовий об'єкт.
- 5 Поясніть, якою інформацією можна скористатися для визначення наступних конструкцій концептуальної моделі даних:
 - об'єктів;
 - атрибутів;
 - зв'язків.
- 6 Приведіть приклад концептуальної моделі деякої предметної області.
- 7 Побудуйте концептуальну модель деякої предметної області, використовуючи поняття складеного об'єкта.
- 8 Поясніть поняття спеціалізації, генералізації, категоризації.

9 Створіть концептуальну модель даних деякого університету, яка б дозволяла одержати наступну інформацію:

- кількість викладачів, що працюють на факультеті;
- список викладачів факультету;
- дисципліни, досліджувані студентами певної спеціальності в шостому семестрі;
- обсяг лекцій по дисциплінах другого семестру для певної спеціальності факультету;
- прізвища викладачів, ведучих вищу математику на факультеті.

Лекція № 6

з навчальної дисципліни Організація баз даних

Тема Реляційна модель даних.

Мета Дати характеристики основним поняттям реляційної моделі даних.

Час – 2 години

План проведення лекції

1 Реляційна модель даних. Основні поняття

2 Структурна частина реляційної моделі

2.1 Реляційне відношення.

2.2 Властивості й види відносин.

2.3 Реляційні ключі.

3 Відновлення відносин.

4 Цілісність бази даних

5 Підсумки

Література

1. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
2. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
3. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
4. Ярцев В.П. Організація баз даних та знань: навчальний посіб-ник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Реляційна модель даних. Основні поняття

У питаннях обробки даних в інформаційних системах на сьогоднішній день реляційні бази даних займають домінуюче положення.

Багато розповсюджених термінів, застосовувані в суміжних областях технологій обробки даних, відрізняються значною розмитістю й неточністю. Щоб уникнути неоднозначного тлумачення тих або інших термінів, потрібно більш чітко позначити необхідні для такої формальної теорії, як реляційна теорія, поняття. До того ж широке впровадження за останні роки об'єктно-орієнтовного підходу в методику побудови БД додало ряду понять трохи інше фарбування.

Реляційний підхід позначає певну ідеологію створення баз даних. Наполегливе "Бажання користувачів оперувати більшими об'єктами, чим елементи даних ТГ-моделей (макрооб'єктами), визначило її появу й сприяло тому, що ця ідеологія досить швидко завоювала мир. На швидкість поширення ідей реляційного підходу значний вплив в основному зробили два фактори.

По-перше, БД представляється на зовнішньому, що не залежить від структури ЕОМ рівні у вигляді сукупності двовимірних таблиць, що повсякденно зустрічаються в людській практиці. Робота з таблицями звична й зрозуміла кожному користувачеві. При цьому досить важливо, що пошук й обробка інформації, що зберігається в таблицях, не залежить від організації зберігання даних у пам'яті ЕОМ, що значно спрощує взаємодію користувача з банком даних й істотно підвищує продуктивність його праці.

По-друге, маніпулювання даними реляційної БД, що з математичної точки зору являє собою кінцевий набір кінцевих відносин різної арності між заздалегідь певною безліччю елементарних даних, здійснюється у відповідності зі спеціально розробленою для цієї мети реляційною теорією. Над відносинами моделі можна здійснювати різні алгебраїчні операції. Теорія РБД і визначає, які операції і який образ необхідно виконувати над відносинами, щоб досягти заданої мети.

Творцем реляційної моделі є співробітник фірми IBM доктор Э. Ф. Кодд. Будучи по утворенню математиком, Э. Кодд запропонував використати для обробки даних апарат теорії множин. У статті "A Relational Model of Data for Large Shared Data Banks", що вийшла у світло в 1970 році, він показав, що будь-яке подання даних зводиться до сукупності двовимірних таблиць особливого виду, відомого в математику як відношення (relation).

Поклавши теорію відносин в основу реляційної моделі, Э. Кодд обґрунтував реляційну замкнутість відносин і ряду деяких спеціальних операцій, які застосовуються відразу до всієї безлічі рядків відношення, а не до окремого рядка. Зазначена реляційна замкнутість означає, що результатом виконання операцій над відносинами є також відношення, над яким у свою чергу можна здійснити деяку операцію. Із цього треба, що в даній моделі можна оперувати реляційними виразами, а не тільки окремими операндами у вигляді простих імен таблиць. Опираючись на абстрактну алгебру, яка розглядає відношення разом з певними над ними операціями, Кодд зміг запропонувати таку формальну теорію реляційної моделі даних, що відрізняють строгі принципи математики і її точність. Одним з основних переваг реляційної моделі є її однорідність. Всі дані розглядаються як збережені в таблицях і тільки в таблицях. Кожен рядок такої таблиці має той самий формат.

Кодд увів дві мови маніпулювання даними, що пропонують більше ефективні засоби доступу до даних й їхній обробки. Сьогодні вони є основою комерційних реляційних мов баз даних, використовуваних у більшості найбільш популярних комерційних СУБД. Серед них найпоширеніші SQL (Structured Query Language - структуризована мова запитів) і QBE (Query-By-Example -запити за зразком). Створені мови маніпулювання даними дозволяють реалізувати всі операції реляційної алгебри й практично будь-які їхні сполучення. Обоє ставляться до мов дуже високого рівня, за допомогою яких користувач указує, які дані необхідно одержати, не уточнюючи процедуру їхнього формування. За допомогою єдиного запиту на кожній із цих мов можна з'єднати кілька таблиць у тимчасову таблицю й вирізати з її необхідні рядки й стовпці.

У цей час реляційний підхід до побудови інформаційних систем є найпоширенішим . До числа достоїнств реляційного підходу можна віднести:

- наявність невеликого набору абстракцій, які дозволяють порівняно просто моделювати більшу частину розповсюджених предметних областей і допускають точні формальні визначення, залишаючись інтуїтивно зрозумілими;
- наявність простого й у той же час потужного математичного апарата, що опирається головним чином на теорію множин і математичну логіку й теоретичний базис, що забезпечує, реляційного підходу до організації БД;
- можливість ненавігаційного маніпулювання даними без необхідності знання конкретної фізичної організації баз даних у зовнішній пам'яті.

Внесок, що Е. Кодду вніс у розвиток реляційної алгебри, реляційного вираховання й нормалізації відносин, важко переоцінити. За "тривалий фундаментальний внесок у теорію й практику розвитку СУБД" Генеральним комітетом Асоціації обчислювальних машин (АСМ) в 1981 році Кодду була вручена премія Тьюринга. Серед цілей створення РМ Кодд називав розробку теоретичних основ організації керування базами даних. У своїй статті, опублікованій в журналі "Computer Word", Кодд сформулював дванадцять правил, яким повинна відповідати дійсна реляційна база даних.

1 правило : Правило інформації. Вся інформація в базі даних повинна бути представлена винятково на логічному рівні й тільки одним способом - у вигляді значень, що втримуються в таблицях.

2 правило : Правило гарантованого доступу. Логічний доступ до всіх і кожним елементом даних (атомарному значенню) у реляційній базі даних повинен забезпечуватися шляхом використання комбінації імені таблиці, первинного ключа й імені стовпця.

3 правило : Правило підтримки недійсних значень. У дійсній реляційній базі даних повинна бути реалізована підтримка недійсних значень, які відрізняються від рядка символів нульової довжини, рядка символів, що є пробілами і від нуля або будь-якого іншого числа й використовуються для подання відсутніх даних незалежно від типу цих даних.

4 правило : Правило динамічного каталогу, заснованого на реляційній моделі. Опис бази даних на логічному рівні повинне бути представлене в тім же виді, що й

основні дані, щоб користувачі, що володіють відповідними правами, могли працювати з ним за допомогою того ж реляційного мови, що вони застосовують для роботи з основними даними.

5 правило : Правило вичерпної підмови даних. Реляційна система може підтримувати різні мови й режими взаємодії з користувачем (наприклад, режим питань і відповідей). Однак повинен існувати принаймні одна мова, оператори якого можна представити у вигляді рядків символів відповідно до деяким чітко певним синтаксисом й який повною мірою підтримує наступні елементи:

- визначення даних;
- визначення подань;
- обробку даних (інтерактивну й програмну);
- умови цілісності;
- ідентифікацію прав доступу;
- границі транзакцій (початок, завершення й скасування).

6 правило : Правило відновлення подань. Всі подання, які теоретично можна оновити, повинні бути доступні для відновлення.

7 правило : Правило додавання, відновлення й видалення. Можливість працювати з відношенням як з одним операндом повинна існувати не тільки при читанні даних, але й при додаванні, відновленні й видаленні даних.

8 правило : Правило незалежності фізичних даних. Прикладні програми й утиліти для роботи з даними повинні на логічному рівні залишатися недоторканими при будь-яких змінах способів зберігання даних або методів доступу до них.

9 правило : Правило незалежності логічних даних. Прикладні програми й утиліти для роботи з даними повинні на логічному рівні залишатися недоторканими при внесенні в базові таблиці будь-яких змін, які теоретично дозволяють зберегти недоторканими дані, що втримуються в цих таблицях.

10 правило : Правило незалежності умов цілісності. Повинна існувати можливість визначати умови цілісності, специфічні для конкретної реляційної бази даних, на підмові реляційної бази даних і зберігати їх у каталозі, а не в прикладній програмі.

11 правило : Правило незалежності поширення. Реляційна СУБД не повинна залежати від потреб конкретного клієнта.

12 правило : Правило одиничності. Якщо в реляційній системі є низькорівнева мова (що обробляє один запис за один раз), то повинна бути відсутньою можливість використання його для того, щоб обійти правила й умови з, виражені на реляційній мові високого рівня (що обробляє кілька записів за один раз).

Формулюючи ці правила, Кодд тим самим прагнув закласти фундамент для побудови ефективних систем. Проте , слід зазначити, що реляційні системи далеко не відразу одержали широке поширення. У той час як основні теоретичні результати в цій області були отримані ще в 70-х, і тоді ж з'явилися перші прототипи реляційних СУБД, довгий час уважався неможливим домогтися ефективною реалізації таких систем. Зараз же, завдяки популярності реляційної моделі, багато хто не реляційні системи, незалежно від використовуваної моделі, намагаються забезпечити реляційним користувацьким інтерфейсом. Однак дійсність така, що не всі деталі реляційної теорії знайшли широке визнання й застосування. Деякі аспекти цієї теорії дотепер є предметом розбіжностей, і на ринку в цей час не існує продукту, що підтримував би все до єдиної з реляційної технології, а користувачів і розроблювачів продовжують турбувати її вузькі місця. У цей час основним предметом критики реляційних СУБД є:

- властивим цим системам деяка обмеженість при використанні в так званих нетрадиційних областях (найпоширенішими прикладами є системи автоматизації проектування), у яких потрібні гранично складні структури даних;
- обмежені можливості адекватного відбиття семантики предметної області.

Сучасні дослідження в області пост реляційних систем головним чином присвячені усуненню саме цих недоліків.

2 Структурна частина реляційної моделі

Будь-яка модель даних може розглядатися як сполучення трьох компонентів, якими є: визначення структур моделі, керування структурами моделі, підтримка цілісності дані моделі. Все перераховане ставиться й до реляційної моделі.

2.1 Реляційне відношення.

Реляційна база даних - це кінцевий (обмежений) набір відносин. Відношення використовуються для подання об'єктів, а також для подання зв'язків між об'єктами. Поняття "відношення" й "таблиця" іноді розглядаються як синоніми, але їх варто розрізняти: відношенням є не будь-яка таблиця, а таблиця, що володіє певними властивостями.

Відношення - це двовимірна таблиця, що має унікальне ім'я й складається з рядків і стовпців, де рядка відповідають записам, а стовпці - атрибутам. Кожен рядок у таблиці представляє деякий об'єкт реального миру або співвідношення між об'єктами.

Атрибут - це поійменованний стовпець відношення. Властивості об'єкта, його характеристики визначаються значеннями атрибутів. Порядок проходження атрибутів не впливає на саме відношення, воно має той самий зміст при будь-якому порядку їхнього проходження.

Нехай є відношення r . Схемою відношення r називається кінцева безліч імен атрибутів $\{A_1, A_2, \dots, A_n\}$. Заголовки стовпців відношення містять імена його атрибутів й, отже, усе разом відбивають його схему. Схема відношення ВИКЛАДАЧ може бути представлена в такий спосіб: {Табельний номер викладача, Прізвище, Посада} або:

ВИКЛАДАЧ
<u>Табельний номер викладача</u>
Прізвище
Посада

Тоді заголовок відношення ВИКЛАДАЧ прийме вид:

<u>Табельний номер викладача</u>	Прізвище	Посада
----------------------------------	----------	--------

Відношення будується з урахуванням ряду факторів. Кожному імені атрибута A_i , $1 \leq i \leq n$ ставиться у відповідність безліч припустимих для відповідного стовпця значень. Ця множина D_i називається *доменом* даного імені атрибута. У самому загальному виді домен визначається завданням деякого базового типу даних, до якого

ставляться елементи домену, і довільного логічного вираження, застосовуваного до елемента типу даних. Якщо обчислення цього логічного вираження дає результат "істина", то елемент даних є елементом домену. Домени досить важливий компонент Реляційної моделі. Домени можуть відрізнятися для кожного з атрибутів, але в той же час кілька атрибутів можуть визначатися на тому самому домені. У семантичному плані поняття домену відбиває той факт, що дані вважаються порівнянними тільки в тому випадку, коли вони ставляться до одного домену. Наприклад, значення доменів Номер пропуску й Номер паспорта ставляться до типу цілих чисел, але не є порівнянними. Для порівнянних же даних можуть бути визначені й різний рід операції, наприклад: з'єднання, перетинання й інших. Перевага системи підтримки доменів полягає в тім, що такій системі доступно більше інформації. Цю інформацію вона може використати з певною метою: більш точно відбивати семантику предметної області й запобігати грубі помилки. Домени по своїй природі є більшою мірою поняттями концептуальними. У більшості реляційних СУБД понять домену повною мірою не використовується, але при визначенні БД специфікація кожного атрибута повинна містити посилання на відповідних домен. У сучасних СУБД підтримка доменів здійснюється на примітивному рівні й так, як це забезпечується в реалізаціях мов програмування, які надають певні убудовані типи даних. Підтримка доменів у контексті Реляційних систем припускає щось більше. Наприклад, користувач повинен мати можливість конструювати необхідні йому складні типи даних, а система повинна бути здатною забезпечити всі аспекти роботи з ними.

Відношення зображуються у вигляді таблиць, де імена атрибутів виносяться в шапку таблиці. Кожен рядок відношення є безліччю значень, узятих по одному з домену кожного імені атрибута. Домени є довільними непустими кінцевими або рахунковими множинами й утворюють множину: $D = D_1 \ D_2 \ \dots \ D_n$. Відношення r зі схемою R - це кінцева безліч відображень $\{t_1, t_2, \dots, t_p\}$ з R в D . Причому кожне відображення $t \in r$ повинне задовольняти наступному обмеженню: $t(A_i)$ належать D_i , де $1 \leq i \leq n$. Ці відображення називаються кортежами. Кожен кортеж відношення відображає екземпляр об'єкта, а атрибут відношення відображає атрибут об'єкта. Кортежі можуть розташовуватися в будь-якому порядку, при цьому відношення буде залишатися тим же самим, а значить мати той же самий зміст.

Множину кортежів називаються тілом відношення. Тіло відношення відбиває стан об'єкта, тому в часі воно постійно міняється. Тіло відношення характеризується кардинальним числом, що дорівнює кількості кортежів, що містяться в ньому.

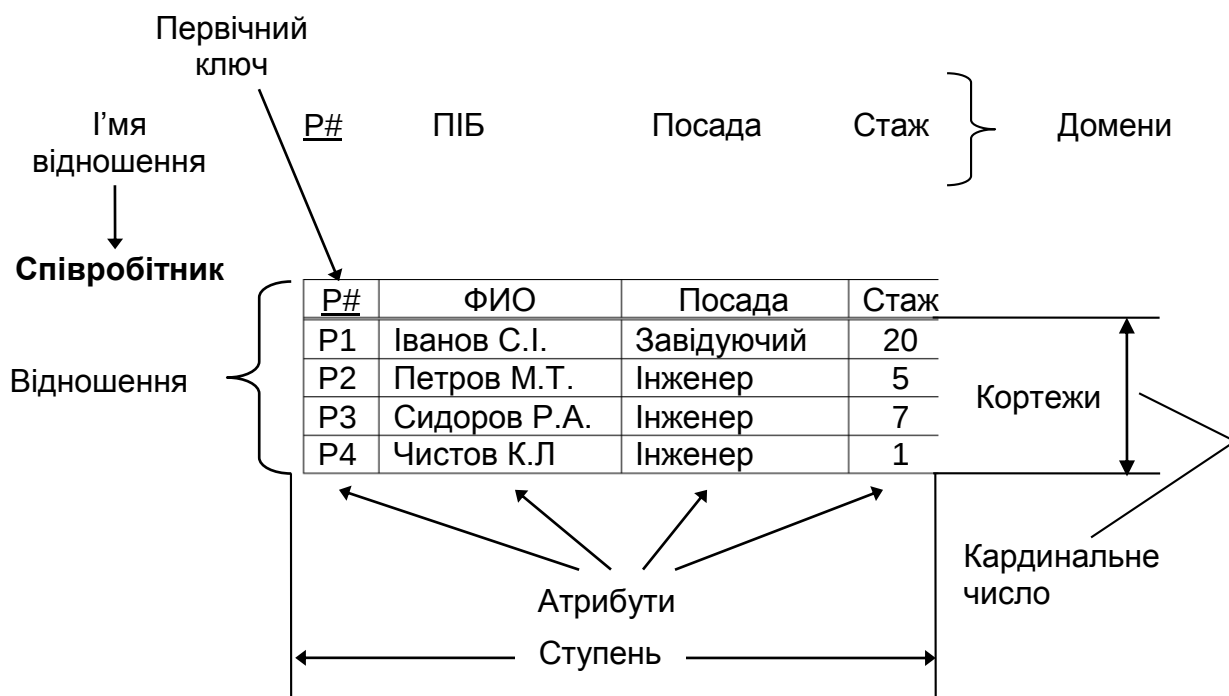


Рис. 1 Відношення СПІВРОБІТНИК

Однієї з головних характеристик відношення є його ступінь. Ступінь відношення визначається кількістю атрибутів, що у ньому є присутнім. Ця характеристика відношення має ще назви: ранг й арність. Відношення з одним атрибутом називається унарним, із двома атрибутами - бінарним, із трьома - тернарним, з п атрибутами п-арним. Визначення ступеня відношення здійснюється по заголовку відношення.

Всі названі характеристики відношення позначені на рисунку 1, де наведене відношення СПІВРОБІТНИК. У даному відношенні є рядок заголовків стовпців і тіло відношення. У рядку заголовків визначені чотири атрибути, отже, ступінь відношення дорівнює 4. Атрибути мають свої унікальні імена: R#, ФІО, Посада, Стаж. Кожен атрибут визначений на своєму домені. Тіло відношення містить ряд рядків - кортежів, де кожен кортеж представляє одного співробітника, а тому в даному відношенні всі рядки різні. Кількість кортежів визначає кардинальне число відношення. У кожному кортежі представлені значення атрибутів, що відповідають даному співробітнику. Значення атрибутів беруться з безлічі припустимих значень відповідного домену, на

якому визначений кожен атрибут. Елементи домену мають атомарні значення, кожне з яких передбачається використати єдиним елементом, не розбиваючи на частині. Це твердження, безумовно, ставиться й до значень атрибута ФІО. Таким чином, наведене відношення є нормалізованим або перебуває в першій нормальній формі, тій, що накладає на відношення самі мінімальні обмеження.

2.2 Властивості й види відношень.

Відношення за структурою подібно таблиці, але таблиці, що володіє певними властивостями. А саме:

- 1 Відношення має ім'я, що відрізняється від імен всіх інших відносин.
- 2 Відношення представляється у вигляді табличної структури. Ім'я таблиці відповідає імені відношення, імена стовпців - іменам атрибутів, а рядка таблиці - кортежам.
- 3 Кожен атрибут має унікальне ім'я, його значення беруться з того самого домену.
- 4 Кожен компонент кортежу є простим, атомарним значенням, що не складається із групи значень. Це не дозволяє замінити значення атрибута іншим відношенням, що привело б до мережного або ієрархічного відношення.
- 5 Упорядкування атрибутів теоретично несуттєво, однак воно може впливати на ефективність доступу до кортежів.
- 6 Всі рядки (кортежі) повинні бути різні.
- 7 Теоретично порядок проходження кортежів не має значення, але цей порядок впливає на ефективність доступу до кортежів.

Більша частина властивостей відношення виникає з того факту, що воно розглядається як безліч. Звідси й неістотність упорядкування атрибутів і кортежів, і відсутність дублікатів кортежів. Особливе місце в ряді властивостей займає вимогу того, щоб кожен атрибут мав тільки прості, атомарні значення. Іншими словами, необхідно будувати такі відношення, у яких у кожному кортежі кожен атрибут може мати тільки єдине значення. Відношення, що задовольняє цій умові, називається нормалізованим. Таким чином, з погляду реляційної моделі відношення завжди нормалізоване. Цим

воно відрізняється від математичного відношення, що зовсім не зобов'язано бути нормалізованим.

Даний факт визначає більше строге обмеження в реляційній моделі в цій сфері, чим у мережній моделі даних, де дозволене використання повторюваних груп. Кодд свідомо пішов на таке обмеження з метою спрощення реляційної моделі даних, але це спрощення є одним з вузьких її місць. Прагнення усунути його поряд з іншими факторами викликало до життя пост реляційні СУБД.

У реляційній теорії зустрічається кілька видів відносин, але не всі вони підтримуються реальними системами. Розрізняють:

- іменоване відношення - це змінна відношення, певна в СУБД за допомогою спеціальних операторів;
- базове відношення - це іменоване відношення, що є частиною бази даних;
- похідне відношення - це відношення, певне за допомогою реляційного виразу через базові відношення;
- подання - це іменоване віртуальне похідне відношення, представлене в системі винятково через визначення в термінах інших іменованих відносин;
- знімки - це відношення, подібні до подань, але вони зберігаються, доступні для читання й періодично оновлюються;
- результат запиту - це неіменоване похідне відношення, одержуване в результаті запиту, що для збереження необхідно перетворити в іменоване відношення;
- збережене відношення - це відношення, що підтримується у фізичній пам'яті.

2.3 Реляційні ключі.

У відношенні можуть існувати трохи одиночних або складених атрибутів, які однозначно ідентифікують кортеж відношення. Це – супер ключі.

В особливу категорію необхідно виділити потенційні ключі. Говорять, що безліч атрибутів $DO = (A_i, A_j, \dots, A_k)$ відношення R є потенційним ключем R тоді й тільки тоді, коли задовольняються два незалежних від часу умови:

- унікальність: у довільний заданий момент часу ніякі два різних кортежі R не мають того самого значення для $A_i, A_j, \dots, A_k$;
- мінімальність: жоден з атрибутів $A_i, A_j, \dots, A_k$ не може бути виключений з K без порушення унікальності.

Відношення може мати кілька потенційних ключів. Ключ, що містить два й більше атрибути, називається складеним ключем. Кожне відношення володіє хоча б одним можливим ключем, оскільки у відношенні не може бути однакових кортежів, а це значить, що, щонайменше, комбінація всіх його атрибутів задовольняє умові унікальності. Потенційні ключі, дозволяючи гарантовано виділити точно один кортеж, забезпечують основний механізм адресації на рівні кортежів реляційної моделі, а отже, мають фундаментальне значення для її успішної роботи. Один з можливих ключів (обраний довільним образом) приймається за його первинний ключ. Звичайно первинним ключем призначається той можливий ключ, яким найпростіше користуватися при повсякденній роботі. Інші можливі ключі, якщо вони є, називаються альтернативними ключами. Для індикації зв'язку між відносинами використовуються зовнішні ключі. Зовнішній ключ - це набір атрибутів одного відношення, що є потенційним ключем іншого відношення. Причому атрибути зовнішнього ключа не обов'язково повинні мати ті ж імена, що й атрибути ключа, яким вони відповідають. Часто в теорії реляційних БД розглядається більше тверда позиція, що вимагає, щоб зовнішні ключі в точності відповідали первинним ключам, а не просто потенційним ключам, що справедливо в більшості випадків. Завдяки наявності зв'язувань між потенційними й зовнішніми ключами забезпечується взаємозв'язок кортежів певних відносин, що тим самим сприяє втриманню бази даних у такому стані, що її можна розглядати як - єдине ціле. Відношення, що містить зовнішній ключ, називається дочірнім або відношенням, що посиляється. А відношення, що містить пов'язаний із зовнішнім ключем потенційний ключ, - батьківським або цільовим відношенням. Зовнішній і відповідний йому потенційний ключі повинні бути визначені на одному домені. Для зовнішнього ключа зовсім не потрібно, щоб він був компонентом якогось потенційного ключа в

утримуючому його відношенні. У дійсності зовнішнім ключем може бути будь-який атрибут або яке-небудь сполучення атрибутів відношення. Далі, по визначенню: кожне значення даного зовнішнього ключа повинне бути значенням відповідні йому потенційного ключа. Зворотне ж зовсім не потрібно. Потенційний ключ батьківського відношення може мати такі значення, які ще не є значеннями зовнішнього ключа в дочірнім відношенні. Наприклад, викладач зарахований у штат деякої кафедри, а виходить, у відношенні ВИКЛАДАЧ з'явився відповідний йому кортеж, але йому ще не доручене ведення жодної дисципліни. Тоді у відношенні ДИСЦИПЛІНА будуть відсутні кортежі, що мають у зовнішньому ключі значення, що відповідають потенційному ключу даного викладача у відношенні ВИКЛАДАЧ. Пояснити структуру бази даних можна шляхом побудови посилальних діаграм. Елементами такої діаграми є імена батьківського й дочірнього відношень і поміщена між ними стрілка, вістря якої спрямоване на ім'я батьківського відношення. Наприклад: ДИСЦИПЛІНА ==> ВИКЛАДАЧ. Те саме відношення може бути в одному зв'язку батьківським, а в іншій - дочірнім. Зручно використати так званий посилальний шлях, що представляє собою ланцюжка зв'язаних стрілками відносин. Наприклад: ДИСЦИПЛІНА ==> ВИКЛАДАЧ ==> КАФЕДРА ==> ФАКУЛЬТЕТ. Більше складні варіанти посилань можуть утворювати посилальні цикли, де в посилальній діаграмі на початку й наприкінці посилального шляху перебуває ім'я того самого відношення. Відношення не можуть розглядатися як статичні об'єкти, тому що вони призначені для відбиття деякої частини реального миру, а ця частина реального миру може змінюватися в часі. Тому й відношення змінюються в часі: кортежі можуть додаватися, віддалятися або модифікуватися. Проте, передбачається, що сама схема відношення інваріантна в часі. Відношення повинне сприйматися як безліч можливих станів, які може приймати відношення.

3 Відновлення відношень.

Для відновлення відносин необхідно мати можливість виконувати наступні операції:

- *додавання кортежу*; Операція додавання для відношення r ($A_1, A_2, \dots, A_n$) має вигляд:

ADD (r ; $A_1=d_1, A_2=d_2, \dots, A_n=d_n$).

Якщо порядок атрибутів фіксований, можливий більше короткий запис:

ADD (r ; $d_1, d_2, \dots, d_n$).

Виконання цієї операції може стати неможливим у ряді випадків:

- додає кортеж, що, не відповідає схемі певного відношення;
- деякі значення кортежів не належать відповідної доменам;
- описаний кортеж збігається по ключі з кортежем, що вже перебуває у відношенні.

видалення кортежу; Операція видалення призначена для видалення кортежів.

Вона може бути записана в такий спосіб:

DEL (r ; $A_1=d_1, A_2=d_2, \dots, A_n=d_n$),

або для впорядкованих атрибутів:

DEL (r ; $d_1, d_2, \dots, d_n$).

Для видалення деякого кортежу часто досить указати значення деякого ключа:

DEL (r ; ключ).

Якщо зазначений кортеж у відношенні відсутній, то відношення залишається незмінним.

зміна кортежу; Операція зміни призначена для модифікації частини кортежу.

Для відношення r її можна при $\{z_1, c_2, \dots \text{порівн}\} \{A_1, A_2, \dots, A_n\}$ визначити так:

CH (r ; $A_1=d_1, A_2=d_2, \dots, A_n=d_n$; $c_1=e_1, c_2=e_2, \dots, c_p=e_p$).

Якщо $ДО = \{B_1, B_2, \dots, B_m\}$ є ключем, то запис даної операції може бути скорочена:

CH (r ; $B_1=d_1, B_2=d_2, B_m=d_m$; $c_1=e, c_2=e_2, \dots, c_p=e_p$).

Можливі помилки в цьому випадку ті ж, що й у попередніх операцій:

- зазначений кортеж не існує;
- зміни мають неправильний формат;
- використовувані значення не належать відповідної доменам.

4 Цілісність бази даних.

Розглянемо питання, пов'язані з підтримкою дані бази в цілісному несуперечливому стані. Саме такий стан бази даних гарантує коректність даних. Підтримка цілісності бази даних реалізується за допомогою ряду обмежень, що накладають на дані.

Перший тип обмежень виникає з того факту, що кожен атрибут визначається на своєму домені або навпаки: домен атрибута задає безліч значень, які може приймати атрибут. Зазначене обмеження називається обмеженням домену. Якщо значення атрибута в даний момент невідомо або неприйнятно для даного кортежу, то в цьому випадку в реляційній теорії Кодда рекомендується використати поняття NULL, що він розглядав як невід'ємну його частину. Варто пам'ятати, що NULL - це не значення атрибута. Це поняття покликане позначати відсутність якого-небудь значення атрибута. Наприклад, фірма випускає виріб, але вартість його ще уточнюється, викладач оформлений на роботу, але склад його навантаження ще не визначений. У системі для атрибута може бути дозволене або заборонено мати значення NULL. У тому випадку, коли атрибуту забороняється мати такі значення, система повинна відкидати будь-які спроби введення значення NULL. Оскільки використання даного поняття значно ускладнює проблеми реалізації моделі, тому що вимагає використання логіки більше високого порядку, чим двозначна, підтримка з визначником NULL забезпечується не кожної реляційною системою.

Другий тип обмежень - категорна цілісність. Категорна цілісність обмежує набір значень первинних ключів базових відносин. Такого роду цілісність полягає в наступному: кортеж не може записуватися в БД доти, поки значення його ключових атрибутів не будуть повністю визначені. Іншими словами: ніякий ключовий атрибут будь-якого кортежу відношення не може містити відсутнього значення, позначуваного визначником NULL. Суть цього обмеження досить легко усвідомити, якщо згадати те, що первинний ключ К повинен мати мінімальність: жоден з його атрибутів $A_i, A_j, \dots, A_k$ не може бути виключений з До без порушення його унікальності. У тім же випадку, коли один з його атрибутів не визначений, саме й відбувається порушення його унікальності - він перестає бути ідентифікуючим атрибутом, а весь об'єкт

стає не ідентифікованим. Оскільки в базу даних не повинна записуватися інформація про щось, що неможливо ідентифікувати, то подібний об'єкт не має права бути включеним у базу даних. Для не базових відносин у реляційній моделі допускається присутність визначника NULL у його первинному ключі.

Третій тип обмежень - цілісність на рівні посилань. При побудові відносин для зв'язування рядків однієї таблиці з рядками іншої таблиці використовуються зовнішні ключі. База даних, у якій всі непусті зовнішні ключі посилаються на поточні значення ключів іншого відношення, має цілісність на рівні посилань. Із цього твердження випливає й інше: зовнішні ключі також не повинні містити NULL значень. Суть питання можна пояснити на прикладі. Нехай є двоє відносин:

STUD (Ном_зач_кн. ФІО, Адреса);

SES (Ном_зач_кн. Оцінка 1, Оцінка2, Оцінка3).

При видаленні запису, що містить відомості про студента з відношення STUD, у відношенні SES з'являться записи, не зв'язані з жодним студентом - відбувається порушення цілісності бази даних. Підтримка посилальної цілісності також вимагає, щоб система запобігала будь-яким спробам створити кортеж у відношенні SES, що містить відомості про здачу сесії студентом з атрибутом Ном_зач_кн доти , поки у відношенні STUD не буде створений кортеж, що містить відомості про цього студента. Відношення ж STUD може містити кортежі з відомостями про студентів, інформація про здачу сесії якими ще не внесена у відношення SES. Не всі СУБД своїми засобами автоматично підтримують цілісність БД. У цьому випадку користувач повинен сам піклуватися про забезпечення цілісності БД при видаленні й модифікації даних. Перевірка цілісності даних може здійснюватися програмними засобами. Однак більше правильним є визначення умови цілісності даних на рівні бази даних, тому що в цьому випадку жодне додаток не може порушити цілісність даних.

Виходячи зі сказаного, для забезпечення цілісності даних необхідно виконати наступні дії:

- 1 При видаленні з відношення запису потрібно перевірити відсутність посилань на видаляє запис, що, у всіх пов'язаних з ним відносинах. Причому насамперед необхідно видалити у зв'язаних відносинах всі записи, які посилаються на видаляє запис, що, і тільки після цього - сам запис.

2 Необхідно уважно ставитися до зміни значень сполучних полів. Наприклад, при видачі нової залікової книжки замість загубленої.

3 При додаванні нового запису у відношення, у якому здійснюється посилання на значення в іншому відношенні, необхідно перевірити наявність сполучних полів у відношенні, на якому здійснюється посилання.

Як правило, визначення умови цілісності даних здійснюється при встановленні взаємозв'язку між батьківським і дочірнім відношеннями; при цьому користувачеві часто надається можливість самому вирішити, яким шляхом зберігати цілісність бази даних і наскільки жорстко повинне дотримуватися умова цілісності при різних операціях зміни даних.

Наприклад, при зміні значення первинного ключа в батьківському відношенні часто дозволені наступні варіанти дій, з яких два перших - забезпечують знаходження бази даних у цілісному несуперечливому стані:

1 При зміні значень полів первинного ключа в батьківському відношенні автоматично здійснюється каскадна зміна всіх відповідних значень у дочірнім відношенні.

2 Не дозволяється змінювати значення полів первинного ключа в батьківському відношенні, якщо в дочірнім відношенні є хоча б один запис, що містить посилання на змінюваний запис.

3 Дозволяється змінювати значення полів первинного ключа в батьківському відношенні, незалежно від існування зв'язаних записів у дочірньому відношенні. Цілісність даних при цьому не підтримується.

При видаленні запису в батьківському відношенні також можливі кілька варіантів дій:

1 При видаленні запису в батьківському відношенні автоматично здійснюється каскадне видалення всіх записів з дочірнього відношення, пов'язаних з записом, що видаляється.

2 Не дозволяється видаляти записи в батьківському відношенні, якщо в дочірнім відношенні є хоча б один запис, що містить посилання на запис, що видаляється. При спробі видалення запису виникає помилка, яку можна обробити програмно.

3 Дозволяється видаляти записи в батьківському відношенні незалежно від існування зв'язаних записів у дочірнім відношенні. Очевидно, що цілісність даних при цьому не підтримується.

При додаванні нового запису в дочірнє відношення або редагуванні в ньому існуючого запису можливо вибрати один з варіантів:

1 Не дозволяється вводити запис, якщо значення індексного вираження дочірнього відношення не відповідає однієї із записів у батьківському відношенні;

2 При уведенні даних у дочірнє відношення не аналізується значення індексного вираження. Цілісність даних при цьому не підтримується.

Ще один вид обмежень, що накладає на інформацію, що втримується у відносинах, пов'язаний з тими бізнесами-правилами, які існують у даній організації або в предметної області. Це так називані додаткові правила підтримки цілісності даних, обумовлені користувачем або з даних, які називаються корпоративними обмеженнями цілісності.

Наприклад:

- студентам не можна планувати заняття тривалістю більше восьми годин у день;
- студенти весь день повинні займатися в одному корпусі вузу, щоб не був потрібно додатковий час на переходи з одного будинку в інше.
-

5 Підсумки

1 Реляційна база даних є сукупністю нормалізованих відносин, що змінюються в часі, різних ступенів арності, які можуть бути зв'язані один з одним через загальні домени.

2 Ступінь відношення - число його атрибутів. Відношення ступеня один називають унарним, ступеня два - бінарним, ступеня три - тернарним, а ступеня п - п-арним.

3 Кардинальне число або потужність відношення - це число його кортежів. Кардинальне число відношення змінюється в часі на відміну від його ступеня.

4 Якщо атрибут не застосуємо або його значення невідомо, те для відбиття цього факту необхідно використати визначник NULL.

5 Реляційна схема бази даних містить імена реляційних таблиць, імена атрибутів, ключові атрибути, зовнішні ключі.

6 У реляційній базі даних повинна забезпечуватися підтримка обмежень доменної, категорної, посилальної й корпоративної цілісності, яким повинні задовольняти дані.

Контрольні питання

1 Освітите історичні аспекти появи реляційного підходу створення баз даних і його зміст.

2 Поясните своїми словами зміст термінів:

- реляційна модель даних;
- реляційна схема бази даних.

3 Охарактеризуйте реляційну модель даних.

4 Дайте розгорнуте пояснення структурної частини реляційної моделі даних.

5 Поясните терміни:

- відношення;
- кортеж;
- атрибут і ступінь відношення;
- первинний і зовнішній ключі;
- домен;
- кардинальне число.

6 Поясните властивості й види відносин.

7 Поясните зміст використання потенційних і первинних ключів відносин.

Як використати ключі для зв'язку відносин?

8 Поясните операції відновлення відносин.

9 Яким образом реалізується підтримка цілісності бази даних?

10 Дайте визначення двох основних правил цілісності бази даних.

11 Які існують шляхи збереження цілісності бази даних при різних операціях зміни даних?

Лекція № 7

з навчальної дисципліни Організація баз даних

Тема Нормалізація відношень реляційної бази даних.

Мета Ввести поняття надмірності даних та дати характеристику аномалій модифікації даних. Дати визначення нормальних форм реляційної схеми БД. Пояснити на прикладах методи приведення відношень до відповідної нормальної форми.

Час – 2 години

План проведення лекції

1. Послідовна нормалізація
 - 1.1. Надмірність даних. Аномалії модифікації даних
 - 1.2. Нормальні форми відношень

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BVH, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Базы даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посіб-ник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1. Послідовна нормалізація

Для одержання найбільш ефективної структури даних необхідно, насамперед, рознести важливу для майбутньої системи інформацію з різних відносин. Варіантів розміщення інформації може бути багато й серед них потрібно вибрати один, який найбільшою мірою відповідає зазначеній меті:

- забезпечення швидкого доступу до даних;
- виключення непотрібного повторення даних, що може з'явитися причиною помилок при уведенні й нераціональному використанні дискового простору комп'ютера;
- забезпечення цілісності даних таким чином, щоб при зміні одних об'єктів автоматично відбувалася відповідна зміна пов'язаних з ними об'єктів.

У реляційних базах даних схема містить як структурну, так і семантичну інформацію. Структурна інформація пов'язана з оголошенням відносин. Семантична інформація виражається безліччю відомих функціональних залежностей між атрибутами відносин, оголошеними в схемі. Присутність відносно функціональних залежностей порозумівається різними причинами. Одні функціональні залежності відбивають взаємозв'язку в досліджуваній предметній області, джерело інших криється в структурі сформованих відносин. При неправильно згрупованих відносинах деякі функціональні залежності можуть виявитися небажаними через побічні ефекти або аномалії, які вони викликають при відновленні БД. У зв'язку із цим виникає питання про коректність представленої схеми. Коректною вважається схема, у якій відсутні небажані функціональні залежності між атрибутами. Для приведення реляційної бази даних до коректного стану пропонується використати розроблений Д. Коддом процес нормалізації, що являє собою метод створення набору відносин із заданими властивостями на основі вимог до даних, установленим у деякій предметній області. Процес нормалізації часто виконується у вигляді послідовності тестів для деякого відношення з метою перевірки його відповідності певним вимогам деякої нормальної форми. У випадку відсутності такої відповідності доводиться прибгати до процедури,

називаною декомпозицією (розкладанням), при якій дана множина відносин замінюється іншою множиною відносин (при цьому число їх зростає), яка є проекцією перших. Ціль цієї процедури - усунути небажані функціональні залежності (а, отже, і аномалії), що становить суть процесу нормалізації. Інакше кажучи, нормалізація - це покроковий, оборотний процес заміни даної схеми (або сукупності відносин) іншою схемою, у якій відносини мають більше просту й регулярну структуру.

1.1. Надмірність даних. Аномалії модифікації даних

Надмірність даних у БД відноситься до небажаних явищ, оскільки веде до збільшення обсягу пам'яті, необхідного для фізичного зберігання відносин. Надмірність викликається, насамперед, дублюванням даних. Не можна сказати, що всі випадки дублювання даних потрібно виключати. Якщо спробувати реалізувати подібну ситуацію, то замість єдиної бази даних зі зв'язаними між собою об'єктами одержимо розрізнені відносини. Тому надмірність повністю усувати не потрібно, потрібно лише її мінімізувати так, щоб були усунуті деякі її види, а залишилися тільки частка надмірності, необхідна для втримання бази даних. Характерний приклад відносини, що містять небажану надмірність наведено у таблиці 1:

Таблиця 1. Відношення СТУДЕНТ

Ном_зал_кн	ПІБ_студента	Код групи	ПІБ_старости	Куратор
20-Т-201	Іванов С.И.	20-Т-11	Рябов В.С	Доц. Фок И.И.
20-Т-215	Петров Я.Р.	20-Т-12	Сизов М.М.	Доц. Докин С.С.
20-Т-217	Рябов В.С	20-Т-11	Рябов В.С	Доц. Фок И.И.
20-Т-211	Сенова А.Л.	20-Т-11	Рябов В.С	Доц. Фок И.И.

У даному відношенні з первинним ключем Ном_зал_кн у кожному кортежі про кожного студента з однієї й тієї ж групи повторюються відомості про код групи, прізвища старости й куратора. Крім того, що для їхнього зберігання будуть потрібні додаткові ресурси пам'яті, при дублюванні інформації легко припуститися помилки при введенні значень атрибутів, у результаті чого база даних перейде в неузгоджений

стан. При роботі з відносинами, що містять надлишкові дані, можуть виникнути проблеми, які називаються аномаліями відновлення.

Аномалії відновлення в базі даних. Процес нормалізації дозволяє проектувальникові бази даних позбутися від присутності в ній різного роду аномалій. Розрізняють три види аномалій у базі даних:

Аномалії включення. У наведеному (див. табл. 1) відношенні можуть виникнути два види аномалій включення. Перший вид цієї аномалії пов'язаний із присутністю надлишкових даних і може виникнути при вставці відомостей про нового студента деякої групи. Оскільки кортеж, що містить відомості про цього студента, повинен включати також інформацію про старосту групи й про її куратора, тобто відповідати подібним уже наявним у базі даних відомостям, то маємо в наявності дублювання даних, при якому будь-яка неточність уведення даних про нового студента переведе базу даних у неузгоджений стан. Другий вид аномалії включення в такій ситуації виникає при спробі створити нову групу й увести її у відношення при тій умові, що в неї ще не зарахований жоден студент. Уведення такої інформації в подібній ситуації вимагає присвоєння значення NULL всім атрибутам опису студента, у тому числі й атрибуту **Ном_зач_кн**, що є первинним ключем даного відношення. Але реалізація такої спроби приведе до порушення категорної цілісності і виходить, система її зобов'язана відхилити. Результатом аналізу є висновок про те, що у відношенні табл. 1 присутні аномалії включення, а отже, це відношення повинне бути перетворене таким чином, щоб від них позбутися.

Таблиця 2. Відношення СТУДЕНТ

Ном_зал_кн	ПІБ_студента	Код групи
20-Т-201	Іванов С.И.	20-Т-11
20-Т-215	Петров Я.Р.	20-Т-12
20-Т-217	Рябов В.С	20-Т-11
20-Т-211	Сенова А.Л.	20-Т-11

Таблиця 3. Відношення ГРУПА

Код групи	ПІБ_старости	Куратор
20-T-11	Рябов В.С	Доц. Фок І.І.
20-T-12	Сизов М.М.	Доц. Докин С.С.
20-T-13	NULL	NULL

Структура відносин, що містить ту ж інформацію, що й відношення СТУДЕНТ, але позбавлена аномалій включення, представлена в табл. 2 й 3. Для того щоб домогтися зазначеної мети, треба було замість одного відношення сформуванати двоє відносин з меншими ступенями, чим вихідне, але ці відносини вже позбавлені аномалій включення. У відношенні СТУДЕНТ для кожного студента крім його атрибутів, які його описують, є при сутнім тільки атрибут, що відноситься до певної групи і єднає обоє відносини, а відношення ГРУПА містить перелік всіх груп, куди дуже просто внести додаткову групу, позначивши невизначені значення описових атрибутів NULL-значеннями.

Аномалії видалення. Повернемося до аналізу відношення, представленого в табл. 1. Нагадаємо, що це відношення містить відомості про студентів, і кожен його кортеж включає значення атрибутів, що ставляться до нього й до групи, у якій він навчається. Причому при видаленні із цього відношення кортежу

20-T-215 Петров Я.Р. 20-T-12 Сизов М.М. Доц. Докин С.С.

з бази даних будуть вилучені весь відомості про групу 20-T-12. Така ситуація представляє собою аномалію видалення. Для виключення з бази дані аномалії видалення це відношення повинне бути перетворене. Причому виявляється, що перетворення повинні бути проведені точно такі ж, які були проведені для виключення аномалії включення. Розглянемо відношення, представлені в табл. 2 й 3. При видаленні кортежу

20-T -215 Петров Я.Р. 20-T-12

з відношення СТУДЕНТ відомості про групу 20-T-12 не зникнуть із бази даних, оскільки вони втримуються у відношенні ГРУПА і їхня присутність не залежить від того факту, чи є в відношенні СТУДЕНТ кортежі, що відносяться до цієї групи, чи ні.

Аномалії модифікації. У вихідному відношенні, представленому в табл. 1, крім аномалій включення й видалення, присутня ще одна аномалія, що називається аномалією модифікації. Така аномалія виникає при спробі змінити що-небудь, що стосується відомостей про групу навчання студента. Допустимо, що в групі 20-Т-11 вирішили призначити нового старосту, наприклад Сенову А. Л. У такій ситуації необхідно переглянути всі кортежі відносини й у кожному кортежі значення атрибута ПІБ_старости замінити Рябов В. С. на Сенова А. Л. При наявності великої кількості кортежів у розглянутому відношенні дуже просто в процесі заміни пропустити який-небудь кортеж або припуститися помилки при введенні нової інформації. Крім більших часових витрат, необхідних для виконання цієї роботи, результатом таких промахів буде переведення бази даних у суперечливий стан. Природно, не можна сподіватися на те, що подібна база завжди буде видавати коректну інформацію. Появу аномалії модифікації можна заблокувати, якщо знову ж удатися до перетворення відношення з табл. 1. І не випадково виявляється, що ці перетворення точно такі ж, які були використані для виключення аномалій включення й видалення. Дійсно: зміна старости групи вимагає зміни значення атрибута ПІБ_старости тільки в одному кортежі відношення табл. 1, що зовсім неважко зробити.

Підведемо підсумки: завдання угруповання відносин може бути вирішено багатьма способами, але потрібно знайти такий склад відношень, що задовольняв би, принаймні, наступним вимогам:

- множина відношень повинна забезпечувати мінімізацію надмірності подання інформації;
- призначені для відношення ключі повинні бути мінімальними;
- при виконанні операцій видалення, включення, модифікації в БД не повинне бути аномалій.
- перебудова набору відношень при зміні структури якого-небудь відношення (додавання атрибутів у відношення, видалення, зміну наявних) повинна бути мінімальною;
- розкид часу виконання запитів у БД повинен бути невеликим.

Наприклад, відношення ПОСТАВКА містить наступні атрибути: ПОСТАВКА (Дата поставки, Назва постачальника, Товар, Адреса постачальника, Кількість товару, Ціна од. товару) Таке відношення має наступні небажані властивості:

- надмірність інформації (Адреса постачальника);
- первинний ключ не мінімальний (Дата поставки + Назва постачальника);
- аномалія модифікації: при зміні Адреса постачальника він не змінюється в інших записах, у результаті виникає порушення цілісності БД;
- аномалія включення: якщо з постачальником укладений договір, але товар не поставлений, його не можна включити в БД;
- аномалія видалення: якщо видалили всі поставки деякого постачальника, то губляться всі відомості про нього.

Проблема зворотності. Щоб виключити різного роду аномалії з відношення, його піддають процесу декомпозиції. При рішенні завдання декомпозиції виникають дві проблеми.

Перша проблема - це проблема зворотності, що полягає в можливості відновлення вихідної схеми, а саме - відновлення будь-якого кортежу вихідного відношення, використовуючи кортежі отриманих у результаті декомпозиції відносин. Декомпозиція, що задовольняє цій вимозі, називається декомпозицією, що гарантує відсутність втрат.

Друга проблема пов'язана зі збереженням залежностей. Варто нагадати, що проектування бази даних містить у собі й визначення обмежень, що накладають на її відносини. У процесі декомпозиції виходять нові відношення й обмеження, які приписуються їм, повинні бути такими, що зберігають вихідні обмеження.

Все сказане означає, що проведена декомпозиція повинна зберігати еквівалентність схем при заміні однієї схеми на іншу, тобто потрібна така декомпозиція, що гарантує відсутність втрат і збереження залежностей. При цьому в результуючому відношенні не повинні з'являтися раніше відсутні кортежі, що є наслідком помилкового з'єднання. Така декомпозиція забезпечує зворотність, тобто одержання вихідної множини відносин шляхом застосування природних з'єднань над їхніми проекціями, а раз це так, то відпадає необхідність виконання з'єднань результуючих відносин для перевірки того, чи не порушуються вихідні обмеження.

1.2. Нормальні форми відношень

Для усунення розглянутих вище недоліків і застосовується процес нормалізація відношень. Даний процес - це формальний метод аналізу відношень на основі їх первинних або потенційних ключів й існуючих функціональних залежностей, що є одним з найбільш строгих способів поліпшення характеристик БД. Він включає ряд формальних правил, використовуваних для перевірки всіх відносин бази даних. Розрізняють:

1НФ - першу нормальну форму; НФБК - нормальну форму Бойса - Кодда;
2НФ - другу нормальну форму; 4НФ - четверту нормальну форму;
3НФ - третю нормальну форму; 5НФ - п'яту нормальну форму.

Кожна нормальна форма накладає певні обмеження на дані. Ці обмеження вводяться в кожному конкретному відношенні, і дотримання цих обмежень у відношенні зв'язано вже з наявністю нормальної форми.

- 1НФ, 2НФ, 3НФ - обмежують залежність не первинних атрибутів від ключів.
- НФБК - обмежує залежність первинних атрибутів.
- 4НФ - формулює обмеження на види багатозначних залежностей.
- 5НФ - уводить інші типи залежностей: залежності з'єднань.

Кожна нормальна форма більше високого рівня припускає, що аналізоване відношення вже перебуває в нормальній формі на рівень нижче розглянутої. Чим вище рівень нормальної форми, тим жорсткіше обмеження накладається на відношення. У ході нормалізації схема бази даних стає усе більше строгою, а її відношення усе менш піддані різного роду аномаліям. Сам процес переходу від нормальної форми більше низького рівня до нормальної форми більше високого рівня й називається нормалізацією відношень (НВ). Для реляційних баз даних необхідно, щоб всі відношення бази даних обов'язково перебували в 1НФ. Нормальні форми більше високого порядку можуть використатися розроблювачами за своїм розсудом. Однак грамотний фахівець прагне до того, щоб довести рівень нормалізації бази даних хоча б до 3НФ, тим самим виключивши з бази даних надмірність даних й аномалії відновлення. У процесі нормалізації на кожному його етапі проводиться аналіз первісних відносин, у результаті

якого виявляються залежності між даними й відповідно до них здійснюється їхнє перегрупування. Процес нормалізації не використовує концепції ER-моделювання й починається з деяких табличних структур, які проектувальник має намір використати в якості вихідних. Які це будуть відношення, залежить від знань, досвіду й інтуїції проектувальника. Вдалий вибір первісних відношень може істотно скоротити обсяг робіт процесу нормалізації й звести його тільки до доказу правильності вибору.

Функціональні залежності й ключі. Функціональна залежність визначається для атрибутів відношення, що перебуває в 1НФ. Функціональна залежність є семантичною властивістю атрибутів відношення. Часто розглядаються функціональні залежності між двома атрибутами. Нехай дана схема відносини $R(a_1, a_2, \dots, a_n)$. Атрибут a_2 функціонально залежить від атрибута a_1 , якщо кожному значенню a_1 відповідає єдине значення a_2 (тобто кожен кортеж, що має одне й те ж саме значення a_1 , повинен мати одне й те ж значення a_2). Позначається подібна ситуація так: $a_1 \rightarrow a_2$. Ліва частина функціональної залежності називається детермінантом, а права частина - залежною частиною. Відсутність функціональної залежності позначається $a \not\rightarrow b$. Створення баз даних переслідує дві основні цілі: понизити надмірність даних і підвищити їхню надійність.

Перша нормальна форма. Відношення перебуває в першій нормальній формі, якщо всі його атрибути мають прості (атомарні) значення. Інакше кажучи, значення в домені кожного атрибута відношення не є ні списками, ні множиною простих або складних значень. Одним словом, у відношенні не повинне бути повторюваних груп. У протилежному випадку відношення вважається ненормалізованим і йому відповідає багаторівнева таблиця (ієрархія) на відміну від однорідної табличної структури нормалізованого відношення. Визначити поняття атомарності важко. Значення, атомарне в одному додатку, може бути неатомарним в іншому. Можна керуватися загальним принципом, що значення не атомарно, якщо в додатку воно використовується вроздріб. Розглянемо приклад відносини, представленого в табл. 4.

Таблиця 4. Відношення НАРОДЖЕННЯ

І'мя	Дата_народження
Ганна	5 березня 1976
Олександр	25 січня 1977
Ольга	1 листопада 1977
Федор	14 вересня 1976

Якщо значення атрибута Дата_ народження передбачається використати цілком, то в цьому ВИПАДКУ дане відношення перебуває в 1НФ. Якби б треба було виділити й окремо використовувати, скажімо, рік, число, місяць, то це відношення не перебувало б в 1НФ, тому що потрібні дані є тільки частинами значення атрибута Дата_ народження. Щоб перевести таке відношення в 1НФ, атрибут Дата_ народження повинен бути розбитий на частини так, як показано в табл. 5.

Таблиця 5. Відношення НАРОДЖЕННЯ

Імя	День_рождения	Місяць_рождения	Год_рождения
Ганна	5	Березень	1976
Олександр	25	Січень	1977
Ольга	1	Листопад	1977
Федор	14	Вересень	1976

Таким чином у відношенні дані можна представляти з тим ступенем деталізації, що потрібно в інформаційній системі. Припустимо, що потрібно додати атрибут Знак, що є знаком зодіаку людини, у перше відношення НАРОДЖЕННЯ (табл. 4). Відомо, що знак зодіаку визначається по місяцю й дню народження людини, а від року народження не залежить. Використовуючи табл. 4, у цій ситуації можна тільки встановити залежність Дата народження $\rightarrow$ Знак, яка дозволить двом особам, що народилися в той самий день, але в різні роки, мати різні знаки зодіаку, що не відповідає дійсності. У другому відношенні НАРОДЖЕННЯ (табл. 5) таких труднощів не виникає, тому що можна записати залежність: (День_народження, Місяць_народження) $\rightarrow$ Знак.

Або, наприклад, табл. 6 є ненормалізованою, і вона не перебуває в 1НФ тому, що включає величини, що є сукупністю атомарних значень. Щоб одержати відношення

СТАТЬ, що перебуває в 1НФ, необхідно його представити так, як це зроблено в табл. 7.

Таблиця 6. Відношення СТАТЬ

Імя	Стать
(Олександр, Федор)	Чоловіча
(Ганна, Ольга)	Жіноча

Таблиця 7. Відношення СТАТЬ

Імя	Стать
Олександр	Чоловіча
Федор	Чоловіча
Ганна	Жіноча
Ольга	Жіноча

Друга нормальна форма. Друга і третя нормальні форми виникли в результаті прагнення уникнути аномалій відновлення даних при роботі із БД і позбутися від інформаційної надмірності у відношення. Аномалії відновлення є небажаним побічним ефектом, обумовленим зміною відношення. Друга нормальна форма застосовується до відношень зі складеними ключами, тобто до таких відносин, первинний ключ яких складається із двох або більше атрибутів. Відношення, у якого первинний ключ включає тільки один атрибут, завжди перебуває в 2НФ. Тобто схема відношення R перебуває в 2НФ, якщо вона перебуває в 1НФ, і кожний не первинний атрибут функціонально повно залежить від первинного ключа. І навпаки, якщо є не ключові атрибути, що залежать від частини первинного ключа, то відношення не перебуває в 2НФ. Таке відношення може страждати від аномалій відновлення.

Приклад 1. Нехай є відношення: ПОСТАВКИ (Номер постачальника, Товар, Ціна). Припустимо, що постачальник може поставляти різні товари, а той самий товар можуть поставляти різні постачальники. Ключ відношення визначене як (Номер постачальника, Товар). За умови, що ціна будь-якого товару зафіксована, семантика відношення включає наступні залежності: (Номер постачальника, Товар $\rightarrow$ Ціна) (по ви-

значенню ключа); (Товар → Ціна). Наявність у цьому випадку неповної функціональної залежності атрибута Ціна від ключа приводить до ряду аномалій. Аномалія включення - якщо в постачальника з'являється новий товар, інформація про товар і його ціну не зможе зберігатися в базі даних доти, поки постачальник не почне поставляти його. Аномалія видалення - якщо поставки деякого товару припиняються, з бази даних прийдеється видалити відомості про товар і його ціну, навіть якщо він ще є в наявності в постачальників. Аномалія модифікації - при зміні ціни товару необхідний повний перегляд відношення з метою знайти всі поставки товару, щоб зміна ціни була відображена для всіх постачальників. Таким чином, зміни значень атрибута одного об'єкта тягне необхідність змін у декількох кортежах відношення: в протилежному випадку база даних виявиться неузгодженою. Причиною цих аномалій є неповна функціональна залежність атрибута Ціна від ключа, що обумовлено об'єднанням в відношенні ПОСТАВКИ двох семантичних фактів. Розкладання вихідного відношення на два відношення: ПОСТАВКИ (Номер Постачальника, Товар): ЦІНА_ ТОВАР (Товар, Ціна) усуває зазначену неповну функціональну залежність. Ціну товару конкретної поставки можна визначити шляхом з'єднання двох відношень по атрибуті Товар. В отриманій схемі реляційної бази даних, що складається із двох відношень, зміна ціни товару викличе модифікацію одного лише кортежу другого відношення, тому що в даних відношеннях аномалія відновлення вже відсутня. Щоб змінити ціну товару, досить змінити його в одному кортежі. Зміни в перше відношення вносити не прийдеється, оскільки в ньому ціна товару відсутня. Така розбивка вихідного відношення дозволило попутно позбутися також і від деякої надмірності даних. Відбулося все це тому, що схема відношення була приведена до другої нормальної форми.

Приклад 2. Нехай є відношення зі схемою: (Номер лікаря, Номер пацієнта, Дата, ПІБ_ пацієнта, Адреса_ пацієнта, ПІБ_ лікаря, Лікування, Ліки, Побічний_ ефект). У даному відношенні визначено первинний складений ключ: (Номер_ лікаря, Номер_ пацієнта, Дата). Дослідження цього відношення на приналежність до 2НФ приводить до наступних результатів.

- Не первинний атрибут ПІБ_ лікаря не залежить функціонально повно від усього первинного ключа, а залежить тільки від його частини Номер_ лікаря.

- Не первинні атрибути ПІБ_ пацієнта й Адреса_ пацієнта не залежать функціонально повно від усього первинного ключа, а залежать тільки від його частини Номер_ пацієнта.

Результати дослідження показують, що дане відношення не перебуває в 2НФ, а отже, може бути піддано всім негативним наслідкам такого стану. Для приведення вихідного відношення до 2НФ необхідно його розбити на три відношення, що перебувають в 2НФ:

ЛІКАР (Номер лікаря, ПІБ_ лікаря);

ПАЦІЄНТ (Номер пацієнта, ПІБ_ пацієнта, Адреса_ пацієнта);

ЛІКУВАННЯ (Номер лікаря, Номер пацієнта, Дата, Лікування, Ліки, Побічний_ ефект).

При аналізі відношень цього приклада, якщо розглядати інформаційну цінність використовуваних атрибутів, може виникнути питання про надмірність атрибутів Номер_ лікаря, Номер_ пацієнта. Дійсно, як ключі у відносинах можуть бути використані атрибути ПІБ_ лікаря й ПІБ_ пацієнта(у межах однієї лікарні чи навряд можна повне їхнє дублювання), але на практиці через їхній великий обсяг (довжини) для зв'язку відношень вигідніше ввести чисельні поля меншої довжини. Це приводить до деякого зменшення обсягу даних і до більшої швидкодії при зв'язуванні.

Приклад 3. Розглянемо відношення КОНСУЛЬТАЦІЇ ДИПЛОМНИКІВ зі схемою: (Табельний номер викладача, Номер залікової книжки, Дата, ПІБ_ викладача, Посада, ПІБ_ студента, Тема_ диплома, Час, Аудиторія, Місткість). Це відношення містить інформацію про те, який викладач консультує певного дипломника, а також дату, час консультації й аудиторію з її місткістю, де вона повинна проводитися. Визначимо основні залежності цього відношення.

- Первинний ключ даного відношення є складовим і по визначенню однозначно ідентифікує кожен кортеж відносини: (Табельний номер викладача, Номер залікової книжки, Дата) → (ПІБ_ викладача, Посада, ПІБ_ студента, Тема_ диплома, Час, Аудиторія, Місткість).

- Описові атрибути викладача ПІБ_ викладача, Посада залежать тільки від частини первинного ключа. Дану ситуацію визначає залежність: Табельний номер викладача → (ПІБ_ викладача, Посада).

- Описові атрибути студента ПІБ_ студента, Тема_ диплома також залежать тільки від частини первинного ключа й не залежать від інших атрибутів ключа, тобто є залежність виду: Номер_ залікової_ книжки $\rightarrow$ (ПІБ_ студента, Тема_ диплома).

Відсутність повної функціональної залежності кожного не первинного атрибута відношення від первинного ключа є джерелом аномалій відновлення й вносить свою частку надмірності в базу даних. Усунення даних негативних явищ здійснюється шляхом декомпозиції вихідного відношення на три з наступними схемами:

ВИКЛАДАЧ (Табельний Номер викладача, ПІБ_ викладача, Посада);

СТУДЕНТ (Номер залікової книжки, ПІБ_ студента, Тема_ диплома);

КОНСУЛЬТАЦІЇ (Табельний Номер викладача, Номер залікової книжки, Дата. Час, Аудиторія, Місткість).

Отримана схема бази даних містить три відношення: перше містить відомості про всіх викладачів, друге містить відомості про всіх дипломників, а останнє відношення КОНСУЛЬТАЦІЇ зв'язує воєдино базу даних, тому що в нього включені первинні атрибути перших двох відносин. Оскільки в базі даних тепер відсутні повторювані дані, то відсутня й можливість привести її в неузгоджений стан.

Третя нормальна форма Розглянемо транзитивну залежність наступного типу: якщо $A \rightarrow B$, $B \not\rightarrow A$ (B не є ключем), $B \rightarrow C$, то $A \rightarrow C$. У цьому випадку вважається, що C транзитивно залежить від A . Проаналізуємо БД приклада 2:

ЛІКАР (Номер лікаря, ПІБ_ лікаря);

ПАЦІЄНТ (Номер пацієнта, ПІБ_ пацієнта, Адреса_ пацієнта);

ЛІКУВАННЯ (Номер лікаря, Номер пацієнта, Дата, Лікування, Ліки, Побічний_ ефект),

а в ній останнє відношення ЛІКУВАННЯ. У зазначеному відношенні існують залежності: (Номер лікаря, Номер пацієнта, Дата) $\rightarrow$ Ліки; Ліки $\rightarrow$ Побічний_ ефект. Остання залежність визначена між описовими атрибутами даного відношення: атрибут Побічний_ ефект залежить від атрибута Ліки, що у свою чергу залежать від первинного ключа, тобто атрибут Побічний_ ефект транзитивно залежить від первинного ключа відношення. Наявність даної залежності в одному відношенні приводить до ряду негативних наслідків:

- не можна включити в БД відомості про ліки, не призначеному жодному хворому;
- при зміні відомостей про побічний ефект ліків потрібно переглянути всю БД для коректної зміни даних.

Транзитивна залежність викликана наявністю у відношенні двох семантичних різних типів. Перетворення відношення в 3НФ усуває розглянуті аномалії. Схема відношення перебуває в третій нормальній формі, якщо вона перебуває в 1НФ і жоден з не первинних атрибутів в R не є транзитивно залежним від ключа для R. Нормалізована БД розглянутого приклада має вигляд:

ЛІКАР (Номер лікаря, ПІБ_лікаря);

ПАЦІЄНТ (Номер пацієнта, ПІБ_пацієнта, Адреса_пацієнта);

ЛІКУВАННЯ (Номер лікаря, Номер пацієнта, Дата, Лікування, Ліки);

ЛІКИ (Ліки, Побічний_ефект).

Розглянемо схему бази даних приклада 3:

ВИКЛАДАЧ (Табельний Номер викладача, ПІБ_викладача, Посада);

СТУДЕНТ (Номер залікової книжки, ПІБ_студента, Тема_диплома);

КОНСУЛЬТАЦІЇ (Табельний Номер викладача, Номер залікової книжки, Дата, Час, Аудиторія, Місткість).

Останнє відношення містить транзитивну залежність: (Табельний Номер викладача, Номер залікової книжки, Дата) $\rightarrow$ Аудиторія $\rightarrow$ Місткість. Отже це відношення не перебуває в 3НФ із усіма наслідками, і насамперед наступними:

- якщо аудиторія виключається з розкладу консультацій, то про неї взагалі губляться відомості;
- якщо аудиторія перебудована й у результаті змінилася її місткість, то прийде продивитися всі кортежі й провести модифікацію значень атрибута.

Для усунення транзитивної залежності необхідно провести декомпозицію останнього відношення, видаливши з нього транзитивно-залежний атрибут і помістивши його в нове відношення разом з копією того атрибута, від якого він залежить. Таким чином, база даних цього приклада, позбавлена транзитивних залежностей, в 3НФ буде виглядати так:

ВИКЛАДАЧ (Табельний Номер викладача, ПІБ_викладача, Посада);

СТУДЕНТ (Номер залікової книжки, ПІБ_ студента, Тема_ диплома);

КОНСУЛЬТАЦІЇ (Табельний Номер викладача, Номер залікової книжки, Дата, Час, Аудиторія);

АУДИТОРІЯ (Аудиторія, Місткість).

Висновки

- Нормалізація являє собою формальний процес удосконалення структури бази даних.
- Перша нормальна форма (1НФ) вимагає, щоб значення стовпців або атрибутів були атомарними.
- Друга нормальна форма (2НФ) вимагає, щоб не ключові атрибути функціонально повно залежали від усього ключа. Якщо первинний ключ відношення складається з одного атрибута, то відношення знаходиться в 2НФ.
- Третя нормальна форма (3НФ) вимагає позбутися від транзитивних залежностей атрибутів.

Питання та завдання до контролю знань студентів

- 1 Освітите історичні аспекти появи реляційного підходу створення баз даних і його зміст.
- 2 Поясніть своїми словами зміст термінів:
 - реляційна модель даних;
 - реляційна схема бази даних.
- 3 Охарактеризуйте реляційну модель даних.
- 4 Дайте розгорнуте пояснення структурної частини реляційної моделі даних.
- 5 Поясніть терміни:
 - відношення;
 - кортеж;
 - атрибут і ступінь відносини;
 - первинний і зовнішній ключі;
 - домен;

- кардинальне число.
- 6 Поясніть властивості й види відношень.
- 7 Поясніть зміст використання потенційних і первинних ключів відносин.

Як використовувати ключі для зв'язку відношень?

- 8 Поясніть операції відновлення відношень.
- 9 Яким образом реалізується підтримка цілісності бази даних?
- 10 Дайте визначення двох основних правил цілісності бази даних.
- 11 Які існують шляхи збереження цілісності бази даних при різних операціях

модифікації даних?

Для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Поясніть своїми словами зміст термінів:

- надмірність даних;
- аномалії включення, відновлення, видалення;
- сторонній атрибут;
- нормалізація відносин;
- перша нормальна форма;
- функціональні залежності;
- друга нормальна форма;
- транзитивні залежності;
- третя нормальна форма;

2 Приведіть приклади відносин, що не перебувають у першій нормальній формі.

3 Розгляньте відношення з наведеною функціональною залежністю. Продемонструйте алгоритм виявлення функціональної залежності.

4 Що таке транзитивна залежність? До яких негативних наслідків приводить наявність транзитивної залежності? Які міри допомагають позбутися від цих наслідків?

- 5 Якщо таблиця перебуває в другій нормальній формі, то ця таблиця

- а) перебуває в першій нормальній формі;
- б) перебуває в третій нормальній формі;

- в) не містить транзитивних залежностей;
- г) містить атрибути, які не є повністю функціонально залежними від ключа,

6 Якщо таблиця перебуває в другій нормальній формі, то ця таблиця

- а) перебуває в нормальній формі Бойса-Кодда;
- б) містить неатомарні атрибути;
- в) не містить транзитивних залежностей;
- г) містить атрибути, які не є повністю функціонально залежними від ключа.

7 Трьома видами аномалій є аномалії

- а) вставки, вибору, видалення;
- б) вставки, модифікації, видалення;
- в) вибору, модифікації, видалення.

Лекція № 8

з навчальної дисципліни Організація баз даних

Тема	Методика перетворення концептуальних структур даних в реляційні структури.
Мета	Охарактеризувати фази проектування реляційної БД. Дати опис та показати приклад спрощення концептуальної моделі даних.

Час – 2 години

План проведення лекції

- 1 Фази проектування БД
- 2 Логічне проектування реляційної бази даних
 - 2.1 Спрощення концептуальної моделі даних

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BVH, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посіб-ник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Фази проектування БД

Розглянутий підхід декомпозиції до проектування схеми реляційної бази даних шляхом послідовної нормалізації первісних відношень, запропонований Коддом, є цілком придатним за умови невеликого числа задіяних атрибутів. Коли число атрибутів перевищує 20, метод, заснований на декомпозиції, стає зайво громіздким. Під час обговорення життєвого циклу бази даних було показано, що одним з найважливіших його етапів є етап проектування бази даних, що починається після створення плану розробки БД, збору й аналізу вимог користувачів і містить у собі три фази проектування:

- *концептуальне проектування*, у результаті якого створюється семантична модель предметної області, що не залежить від яких-небудь фізичних умов реалізації;
- *логічне проектування*, у результаті якого концептуальна модель предметної області перетерплює змін, викликаних видом обраної моделі даних;
- *фізичне проектування*, що розглядає питання фізичної реалізації отриманої логічної моделі за допомогою обраної СКБД.

Перша фаза проектування - створення концептуальної моделі інформаційної системи (ІС), що забезпечує точне й наочне подання складних аспектів прикладної проблеми. Основою методики побудови концептуальної моделі ІС служить процес створення ER-моделі. Оскільки процес побудови концептуальної моделі ІС був розглянутий досить докладно, те будемо виходити з того факту, що фаза концептуального проектування завершена, і в її ході для даної предметної області визначені й документовані наступні основні моменти моделі:

- типи об'єктів;
- типи зв'язків;
- атрибути об'єктів;
- домени атрибутів;
- потенційні й первинні ключі об'єктів;
- ER-діаграма предметної області.

Друга фаза процесу проектування БД - логічне проектування бази даних, для якої вже визначена модель даних - реляційна модель даних. Іншими словами, друга фаза - це фаза проектування реляційної бази даних, де концептуальна модель використовується як проміжна модель, що згодом за допомогою застосування деякої методології перетвориться в реляційну. Такий підхід, крім усього іншого, відрізняється від методу декомпозиції ще й тим, що функціональні залежності залучаються не на початковому, а на кінцевому етапі проектування. Тому важливою характеристикою зазначеного процесу є той факт, що всі отримані в результаті відношення будуть мати четверту нормальну форму, а отже, подальша нормалізація не буде потрібна.

Третя фаза проектування - фізичне проектування БД за допомогою конкретно ви-лайливої СУБД.

2 Логічне проектування реляційної бази даних

Концептуальна модель даних складається з ряду компонентів: об'єктів (простих або складових), зв'язків, атрибутів, спеціалізацій або генералізацій. При переході до реляційної схеми бази даних кожний із цих компонентів повинен бути проаналізований й, якщо це виявиться необхідним, то навіть і перетворений. Зміни, внесені в процесі перетворення, повинні бути такими, щоб їх результат повністю відповідав вимогам, висунутим реляційною моделлю даних. Створена логічна модель БД далі повинна пройти перевірку з використанням процедури послідовної нормалізації, а також повинна пройти контроль на можливість виконання всіх необхідних користувачеві транзакцій. Таким чином, дана фаза логічного проектування припускає наступні дії:

- перетворення концептуальної моделі даних у логічну модель, у результаті якого буде визначена схема реляційної моделі даних;
- перевірка моделі за допомогою концепцій послідовної нормалізації;
- перевірка моделі відносно транзакцій користувачів;
- перевірка підтримки цілісності даних.

Розглянемо послідовно кожен дію.

2.1 Спрощення концептуальної моделі даних

Перетворення концептуальної моделі даних у логічну модель, у результаті якого буде визначена схема реляційної моделі даних, може мати два підходи. Один з підходів полягає в тому, що проектувальник працює з концептуальною моделлю прямо, не прибігаючи до її попереднього перетворення. У цьому випадку йому доведеться зіштовхнутися з необхідністю перетворення різноманітних структур даних, причому на шляху перетворення деяких з них він може зустріти ряд труднощів. Другий же підхід полягає в тому, що проектувальник, перш ніж приступитися до процесу переходу від концептуальної моделі до логічної моделі, прагне спочатку даний перехід максимально спростити, провівши попередні перетворення концептуальної моделі, перетворення деяких її, що не підходять для реляційних СУБД, структур даних. До таких структур даних ставляться:

- зв'язку типу "багато - до - багатьох";
- складні зв'язки;
- рекурсивні зв'язки;
- зв'язки з атрибутами;
- множинні атрибути;
- надлишкові зв'язки.

Точка зору другого підходу на процес перетворення концептуальної моделі така: якщо в моделі присутні перераховані небажані структури, то вони повинні бути виключені шляхом тотожної їхньої заміни на структури даних, припустимі для логічної моделі. Розглянемо спочатку другий підхід, що припускає наявність проміжного етапу на шляху одержання логічної моделі БД, етапу одержання спрощеної концептуальної моделі.

Виключення зв'язку типу "багато - до - багатьох" Перетворення зв'язку типу "багато - до - багатьох" здійснюється шляхом введення деякого проміжного об'єкта із заміною одного зв'язку двома зв'язками типу "один - до - багатьох" зі знову створеним об'єктом. При організації нових зв'язків необхідно стежити за тим, щоб максимальна потужність зв'язку "один" була спрямована до вихідного об'єкта, а максимальна по-

тужність зв'язку "багато" - до знову створеного об'єкта. Наприклад, допустимо ситуацію (див. рис. 2.1), коли один викладач може читати кілька курсів, і один курс може викладатися декількома викладачами.



Рис. 2.1. Діаграма зв'язку типу "багато - до - багатьох"

Створимо додатковий об'єкт ЧИТАННЯ й визначимо нові два зв'язки (див. рис. 2.2):

ВИКЛАДАЧ_ЧИТАЄ типу 1:N між об'єктами ВИКЛАДАЧ і ЧИТАННЯ;
 КУРС_ЧИТАЄТЬСЯ типу 1:M між об'єктами КУРС і ЧИТАННЯ.



Рис. 2.2. . Перетворена діаграма зв'язку типу "багато - до - багатьох"

Виключення складних зв'язків. Хоча за допомогою бінарних зв'язків можуть бути описані багато ситуацій реального миру, проте неминуче виникнення й таких ситуацій, у яких побудова розумної моделі організації не може бути зведена до реалізації тільки таких зв'язків. Наприклад, виникає потреба моделювання тристоронніх зв'язків. Зв'язок, у якому кількість учасників перевищує два, тобто тернарний зв'язок й зв'язок більше високого порядку, n-арний зв'язок, називається складним. Виключення такого зв'язку з концептуальної моделі відбувається по наступному сценарію:

- у модель уводиться новий об'єкт;
- складний зв'язок замінюється бінарними зв'язками типу "один - до - багатьох" вихідних об'єктів зі знову створеним об'єктом, причому кількість бінарних зв'язків дорівнює ступеню складного зв'язку.

Наприклад, нехай заняття за деяким курсом ведуться деяким викладачем у деякій лабораторії. Концептуальна модель у цій ситуації може містити структуру даних, зображену на рисунку 2.3. Даний тернарний зв'язок, що відбиває відносини між викладачем, заняттями й лабораторією, повинен бути заміненим необхідною кількістю бінарних зв'язків типу "один - до - багатьох". Уведемо додатковий об'єкт: ЛАБОРАТОРНІ_ЗАНЯТТЯ й визначимо для нього три бінарні зв'язки типу "один - до - багатьох" з кожним з вихідних об'єктів (див. рис. 2.4): зв'язок ВИКЛАДАЧ_ВЕДЕ для об'єктів ВИКЛАДАЧ, ЛАБОРАТОРНІ_ЗАНЯТТЯ; зв'язок В для об'єктів ЛАБОРАТОРНІ_ЗАНЯТТЯ, ЛАБОРАТОРНІ_ЗАНЯТТЯ; зв'язок ВЕДУТЬСЯ для об'єктів ЗАНЯТТЯ, ЛАБОРАТОРНІ_ЗАНЯТТЯ.

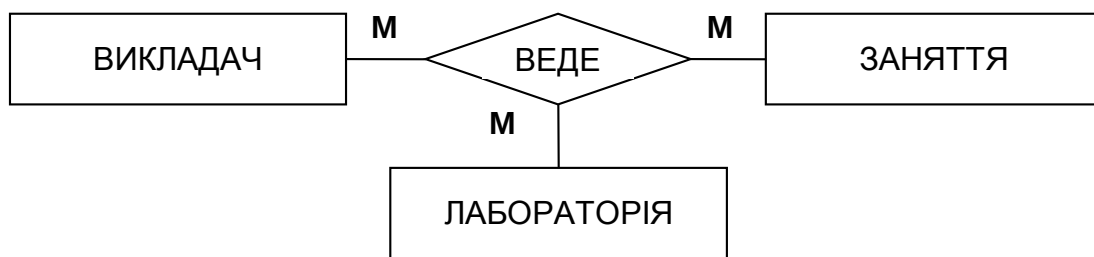


Рис. 2.3. Тернарний зв'язок ВИКЛАДАЧ - ЗАНЯТТЯ — ЛАБОРАТОРІЯ

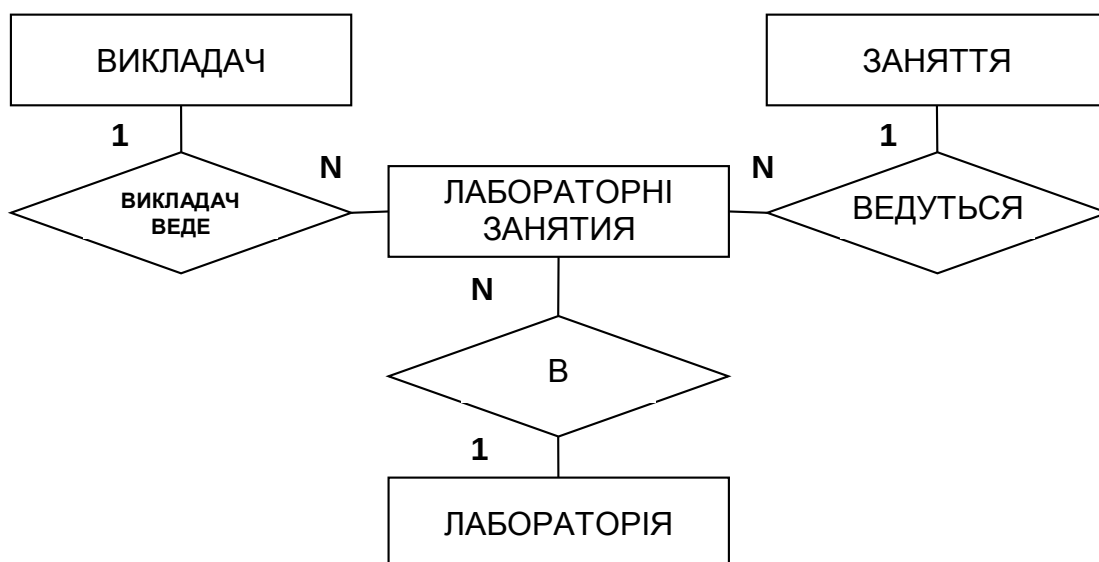


Рис. 2.4. Заміна тернарного зв'язку бінарними зв'язками

Отримана в результаті даного перетворення концептуальна модель уже не містить небажані для реляційної схеми бази дані структури.

Виключення рекурсивних зв'язків. Рекурсивні зв'язки з погляду їхньої реалізації

в реляційних схемах БД також ставляться до небажаних структур. Рекурсивний зв'язок - це особливий вид зв'язку, в якому одні й ті самі екземпляри об'єкта беруть участь кілька разів у різних ролях. Якщо концептуальна модель містить такі зв'язки, то перед переходом до реляційної схеми БД вони повинні бути виключені з моделі. Виключення здійснюється шляхом введення в модель додаткового об'єкта й визначення його зв'язків. Розглянемо ситуацію, коли один зі співробітників одночасно є й керівником підрозділу. Наявність у концептуальній моделі, отриманої з урахуванням вищеописаних факторів (див. рис. 2.5), рекурсивного зв'язку типу "один - до - багатьох" значно ускладнює положення, тому що подібні зв'язки при їхньому моделюванні вимагають до себе особливого підходу. Провівши перетворення вихідної моделі шляхом виключення даного небажаного зв'язку, можна спростити ситуацію. Для цього введемо додатковий об'єкт РОЛЬ_СПІВРОБІТНИКА й установимо наступні його зв'язки (див. рис. 2.6): ПРАЦЮЄ - зв'язок типу "один - до - багатьох"; КЕРУЄ - зв'язок типу "один - до - одного". Отримана в результаті даного перетворення концептуальна модель уже не містить небажаних для реляційної схеми бази дані структур.

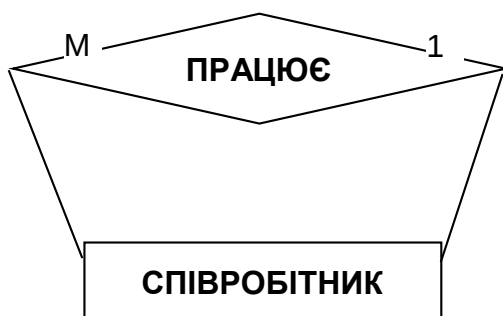


Рис. 2.5. Рекурсивний зв'язок

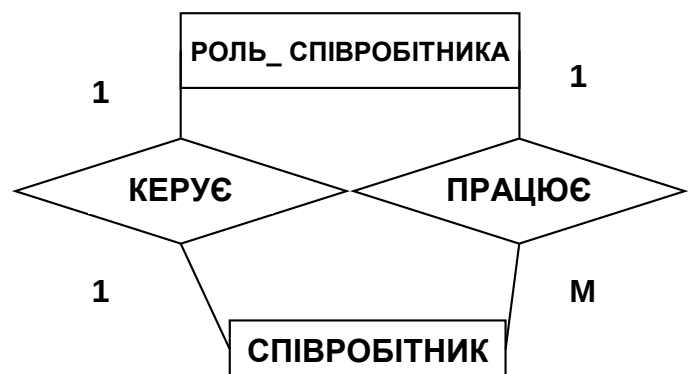


Рис. 2.6. Виключення рекурсивного зв'язку

Виключення зв'язків з атрибутами. Існування в моделі зв'язків з атрибутами також відноситься до тих факторів, які не бажано мати в логічній моделі даних. Виключення зв'язку з атрибутами може протікати по вже знайомому сценарію: додавання в модель нового об'єкта з визначенням його зв'язків. Розглянемо ситуацію, зображену на рисунку 2.7, де зв'язок ЧИТАЄ має свій атрибут Час.

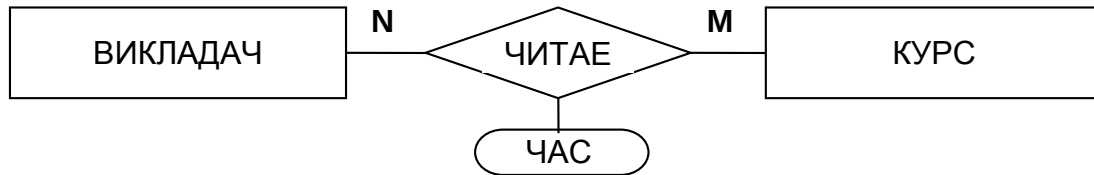


Рис. 2.7. Зв'язок з атрибутами

Для виключення з моделі такого зв'язку введемо додатковий об'єкт ЛЕКЦІЯ й припишемо йому атрибут Час (див. рис. 2.8). Між вихідними об'єктами й новим об'єктом установимо наступні зв'язки: ЧИТАЄ_В - зв'язок типу 1:N між об'єктами ВИКЛАДАЧ і ЛЕКЦІЯ; ЧИТАЄТЬСЯ - зв'язок типу 1:N між об'єктами КУРС і ЛЕКЦІЯ.

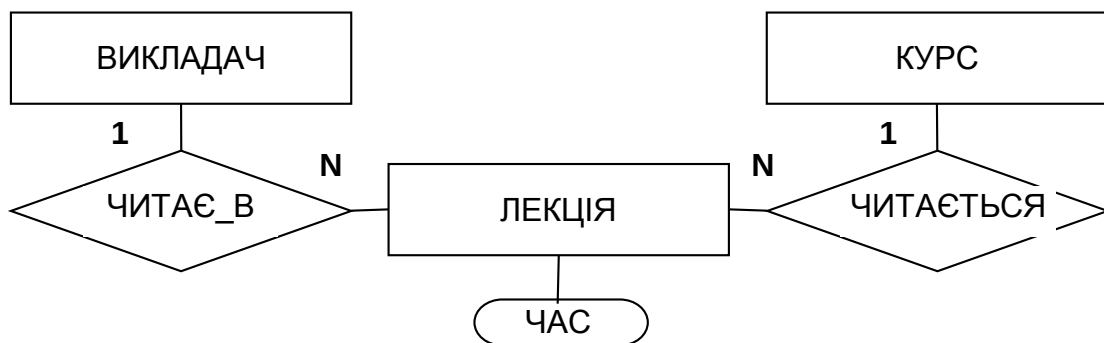


Рис. 2.8. Виключення зв'язку з атрибутами

Отримана в результаті даного перетворення модель має просту структуру й може бути без утруднень відбита на реляційну схему БД.

Виключення множинних атрибутів. Оскільки в якості логічної моделі даних ми розглядаємо реляційну модель даних, то необхідно не упускати з виду той факт, що 1НФ для реляційного відношення вимагає, щоб всі його атрибути мали прості атомарні значення. Тому, якщо який-небудь об'єкт концептуальної моделі має атрибут множинного типу, то ця модель повинна бути перетворена. Перетворення й у цьому випадку здійснюється введенням додаткового об'єкта. Допустимо, що в концептуальній моделі присутній об'єкт ВІДДІЛ організації, що має атрибут Номер_ телефону. У тому випадку, якщо в деякому відділі встановлений не один, а декілька контактних

телефонів, то даний атрибут має множинний тип. Для його виключення з моделі введемо додатковий об'єкт ТЕЛЕФОН й установимо його зв'язок з об'єктом ВІДДІЛ так, як показано на рисунку 2.9.

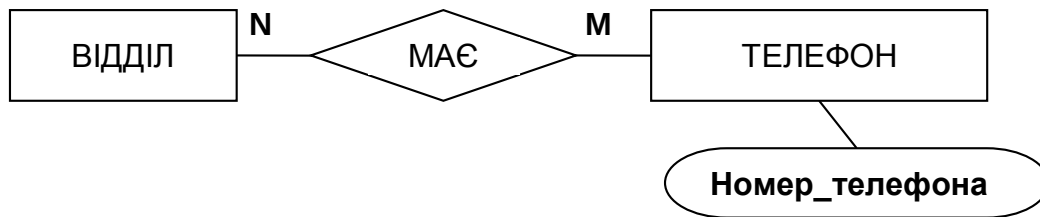


Рис. 2.9. Виключення множинного атрибуту

Виключення надлишкових зв'язків. Надмірність інформації взагалі в базах даних ставиться до небажаних факторів, які по можливості необхідно усувати. Наявність у концептуальній моделі надлишкового зв'язку, наприклад, характеризується тим, що та й сама інформація може бути отримана не тільки через цей зв'язок, але й за допомогою іншого зв'язку. Для виявлення надлишкових зв'язків модель піддається всебічному дослідженню із приводу того, якими шляхами може бути отримана різноманітна необхідна інформація. Якщо в моделі буде виявлений надлишковий зв'язок, то вона повинна бути виключена з її, оскільки в інформативному плані вона є зайвою і її присутність ускладнює модель.

У результаті виконання перерахованих вище дій по виключенню з моделі різних, небажаних структур даних буде отримана спрощена концептуальна модель предметної області, що, відповідно до певної методики, може легко бути перетворена в реляційну модель даних.

Питання та завдання до контролю знань студентів

- 1 Поясните своїми словами зміст термінів:
 - надмірність даних;
 - аномалії включення, відновлення, видалення;
 - сторонній атрибут;

- 2 Приведіть приклади відносин, що не перебувають у першій нормальній формі.
- 3 Розгляньте відношення з наведеною функціональною залежністю. Продемонструйте алгоритм виявлення функціональної залежності.
- 4 Що таке транзитивна залежність? До яких негативних наслідків приводить наявність транзитивної залежності? Які міри допомагають позбутися від цих наслідків?
- 5 Що служить альтернативою нормалізації за допомогою декомпозиції?
- 6 Які небажані функціональні залежності усуваються за допомогою приведення відносин до нормальної форми Бойса - Кодда?
- 7 Охарактеризуйте багатозначні залежності. Приведіть приклади відносин, де вони присутні.
- 8 Освітите процес проектування реляційної бази даних. Які дії припускає фаза логічного проектування реляційної БД?
- 9 Яким образом можна спростити концептуальну модель даних, щоб полегшити процес переходу від концептуальної моделі до логічної моделі?
- 10 Викладете правила методики перетворення концептуальних структур даних у реляційні структури.
- 11 Побудуйте концептуальну модель із Про Бібліотека й з її допомогою одержите реляційна модель відповідної бази даних.

Лекція № 9

з навчальної дисципліни Організація баз даних

Тема	Структура та типи даних мови SQL. Створення та видалення баз даних. Створення, видалення та модифікація таблиць. Вставка, видалення та оновлення даних.
Мета	Дати характеристику двох складових частин мови SQL необхідних для роботи з базами даних. Розглянути роботу операторів створення та видалення бази даних, створення таблиць та індексів, модифікації таблиць та індексів. Розглянути роботу операторів INSERT, REPLACE, DELETE, TRUNCATE й UPDATE
Час:	2 години.

План проведення лекції

- 1 Структура й типи даних мови SQL
- 2 Створення та видалення баз даних.
 - 2.1 Створення бази даних
 - 2.2 Створення таблиць
- 3 Вставка, видалення й оновлення даних
 - 3.1 Використання INSERT
 - 3.2 Використання REPLACE
 - 3.3 Використання DELETE
 - 3.4 Використання TRUNCATE й UPDATE

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посібник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Структура й типи даних мови SQL

Розгляд почнемо зі структури самої мови SQL і типів даних, на які можна опиратися при створенні різних конструкцій мови. На відміну від теоретичних мов, введених Коддом, реляційної алгебри й реляційного числення, призначених тільки для реалізації запитів до БД, SQL є повною мовою, у ньому присутні як складові обидві необхідні для роботи з базами дані частини: мова маніпулювання даними - DML і мова визначення даних - DDL. Крім того, даний мова містять оператори, призначені для керування БД. Коротко позначимо основні базисні елементи мови SQL.

У мові SQL1 визначений ряд типів даних, які представлені в таблиці 1.

Таблиця 1. Типи даних

Тип	Пояснення
CHARACTER (n) або CHAR(n)	Символьні рядки постійної довжини в n символів
NUMERIC [(n,m)]	Точні числа, тут n - загальна кількість цифр у числі, m- кількість цифр ліворуч від десяткової крапки
DECIMAL [(n,m)]	Точні числа, тут n - загальна кількість цифр у числі, m - кількість цифр ліворуч від десяткової крапки
DEC [(n,m)]	Те ж, що й DECIMAL [(n,m)]
INTEGER или INT	Цілі числа
SMALLINT	Цілі числа меншого діапазону
FLOAT [(n)]	Числа великої точності, збережені у формі із плаваючою крапкою. Тут n - число байтів, що виділяється під зберігання одного числа
REAL	Дійсний тип чисел, що відповідає числам із плаваючою крапкою, меншої точності, чим FLOAT
DOUBLE PRECISION	Специфікує тип даних з певної в реалізації точністю, більшої, ніж певна в реалізації точність для REAL

У стандарт SQL2 додано розширювальні можливості мови типів даних (таблиця 2).

Таблиця 2. Додаткові типи даних

Тип	Пояснення
VARCHAR(n)	Строки символів змінної довжини
NCHAR (n)	Строки локалізованих символів постійної довжини
NCHAR VARYING(n)	Строки локалізованих символів змінної довжини
BIT (n)	Строка бітів постійної довжини
BIT VARYING (n)	Строка битов змінної довжини
DATE	Календарна дата
TIMESTAMP(точність)	Дата і час
INTERVAL	Часовий інтервал

У стандарті визначені наступні константи:

- для числових типів даних визначені константи у вигляді послідовності цифр із обов'язковим завданням знака числа й десятковою крапкою;
- константи із плаваючою коми, як й у більшості мов програмування, обумовлені шляхом завдання мантиси й порядку, розділених символом E.

Наприклад: 2.9E-4; -134.235E7; 0.54267E18;

- строкові константи, які повинні бути укладені в одинарні лапки: 'Іванов І.І.';
- константи дати, часу й тимчасового інтервалу - у реляційних СУБД представляються у вигляді строкових констант;
- спеціальні системні константи.

Стовпці й типи даних в MySQL

В MySQL є три основних типи стовпців: числового, текстові або рядка, а також дати й часу. Ми розглянемо всі їх один по одному.

Числові типи

Числові типи стовпців використовуються для зберігання чисел. У нашому прикладі ми використали типи int (ціле число) і float (число із плаваючою коми). Вони представляють два підтипи числових типів: точні числові типи й наближені числові типи.

Числові типи можуть характеризуватися максимальною довжиною, а типи із плаваючою коми - числом десяткових розрядів. Ці значення вказуються відразу після оголошення типу, наприклад:

```
salary decimal(10, 2)
```

Тут зазначена довжина 10 і два знаки після десяткового роздільника.

Можна не використати ніяких параметрів взагалі й вказати тільки загальну довжину або ж вказати як довжину, так і число десяткових розрядів.

Оголошення числових типів можна також завершувати ключовими словами UNSIGNED й (або) ZEROFILL.

Ключове слово UNSIGNED вказує, що стовпець із тільки позитивні числа або нулі. Ключове слово ZEROFILL означає, що число з відображатися із провідними нулями.

NUMERI або DECIMAL

Ці типи ідентичні, а DECIMAL можна також скоротити до DEC. Ці типи використовуються для зберігання точних значень із плаваючою коми й звичайно використовуються для того, щоб запам'ятовувати грошові значення. Вони мають той же діапазон, що й числа із плаваючою коми подвійної з.

INTEGER і його варіації

INTEGER можна скоротити до INT. Це - стандартне ціле число, що займає 4 байти, з діапазоном з 232 можливих значень. Існує також кілька варіацій INT.

1 TINY INT займає 1 байт(28 можливих значень). Синонімами TINYINT є VI й BOOL.

2 SMALLINT займає 2 байти (216 можливих значень).

3 MEDIUMINT займає 3 байти (224 можливих значень).

4 BIGINT займає 8 байтів (264 можливих значень).

Нижче обговорюються наближені числові типи,

FLOAT

Це - числа із плаваючою коми зі звичайною точністю. Вони можуть представляти позитивні числа в діапазоні $1,18 \cdot 10^{-38}$ - $3,40 \cdot 10^{38}$ й аналогічний діапазон негативних чисел.

DOUBLE

Це - числа із плаваючою коми з подвійною точністю. Синонімами DOUBLE є REAL й DOUBLE PRECISION. Вони можуть представляти позитивні числа в діапазоні $2,23 \cdot 10^{-308}$ - $1,80 \cdot 10^{308}$ й аналогічний діапазон негативних чисел.

Текстові типи й рядки

MySQL підтримує різні текстові типи й рядки. Основними з них є CHAR, VARCHAR, TEXT, BLOB, ENUM й SET, Ми розглянемо кожний із цих типів по черзі.

CHAR

Тип CHAR використовується для зберігання рядків фіксованої довжини. Як й у розглянутому вище прикладі створення бази даних employee, після ключового слова CHAR звичайно вказується довжина рядка, наприклад CHAR (20), Якщо не вказати довжину, ви одержите CHAR(1). Максимальна довжина значення типу CHAR - 255 символів. При збереженні значень типу CHAR їм завжди буде виділятися довжина, зазначена вами в /декларації. Для цього місце, що залишилося в стовпці, буде заповнено пробілами. При витягу вмісту стовпця типу CHAR автоматично додані пробіли відкидаються.

Очевидно, що зберігання рядків типу CHAR зажадає більше місця на диску, чим зберігання таких же рядків змінної довжини. Перевага полягає в тому, що зі стовпців фіксованої довжини (зі стовпців типу CHAR, NUMERIC або DATE) дані витягаються швидше. Часто швидкість виявляється більше важливим параметром, чим дисковий простір, тому з погляду оптимізації буває вигідніше використати тип CHAR для текстових полів, не підданих більшим варіаціям.

Перед CHAR (як і перед VARCHAR) можна вказати ключове слово NATIONAL, щоб обмежити вміст стовпця відповідним стандартним набором символів. В MySQL воно використовується за замовчуванням, так що вам це ключове слово може знадобитися тільки для кросплатформеної сумісності.

Після CHAR й VARCHAR можна додати ключове слово BINARY, що означає необхідність обліку регістра символів при порівнянні рядків. За замовчуванням при порівнянні рядків регістр символів ігнорується.

VARCHAR

Тип VARCHAR призначений для зберігання рядків змінної довжини. Після вказівки типу в дужках вказується довжина, наприклад VARCHAR (10). Допускаються значення від 0 до 255.

TEXT, BLOB й їхньої варіації

Типи TEXT використовуються для зберігання більше довгих фрагментів тексту, чим допускається типами CHAR й VARCHAR. Аббревіатура BLOB означає Binary Large Object (великий двійковий об'єкт). Ці типи однакові, за винятком того, що тип BLOB призначений для зберігання двійкових даних, а не тексту. Порівняння значень типу BLOB залежать від регістра символів, а значень TEXT - немає. Обидва типи мають змінну довжину й обоє пропонуються в різних розмірах.

1 TINYTEXT й TINYBLOB можуть зберігати до 255 (це 2⁸ - 1) символів або байтів.

2 TEXT й BLOB можуть зберігати до 65535 (2¹⁶ - 1) символів або байтів (64 Кбайт).

3 MEDIUMTEXT й MEDIUMBLOB можуть зберігати до 16777215 (2²⁴-1) символів або байтів (16 Мбайт).

4 LONGTEXT й LONGBLOB можуть зберігати до 4294967295 (2³² - 1) символів або байтів (4GB).

ENUM

Цей тип дозволяє перелічити набір можливих значень. Кожен рядок може містити одне значення з перерахованого набору. Декларація типу ENUM виглядає в такий спосіб:

```
gender enum('m', 'f')
```

Даний тип допускає також значення NULL, так що можливими значеннями gender будуть m, f, NULL або error (помилка).

SET

Тип SET (набір) подібний до типу ENUM за винятком того, що в цьому випадку рядка можуть містити набір значень із безлічі перерахованих.

Типи дати й часу

MySQL підтримує різні типи дати й часу, які обговорюються нижче.

DATE

Тип DATE призначений для зберігання дат. MySQL очікує одержати дату в стандартній формі ISO (день^день-рік-місяць-день), а не в трансатлантичних її варіантах. Відображаються значення дати у вигляді ГГГГ-ММ-ДД.

TIME

Цей тип призначений для зберігання значень часу, відображуваних у вигляді ЧЧ:ММ:СС.

DATETIME

Це - комбінація попередніх двох типів. Формат її наступний:

ГГГГ-ММ-ДД ЧЧ:ММ:СС.

TIMESTAMP

Це - дуже корисний тип стовпця. Якщо у відповідному стовпці рядка ви не вкажете конкретне значення або NULL, там буде записане час, коли відповідний рядок був створений або востаннє змінена.

Значення, що витягає, TIMESTAMP буде відображатися у форматі DATETIME. Тут є істотне розходження між MySQL 4.0 й 4.1. Раніше ви могли встановити ширину значення при оголошенні стовпця типу TIMESTAMP.

YEAR

Цей тип призначений для зберігання значень року. При оголошенні стовпця цього типу можна оголосити його як YEAR (2) або YEAR (4), щоб указати число знаків. За замовчуванням використовується YEAR (4). YEAR (2) представляє значення з 1970 по 2069.

В операторах SQL можуть бути використані вираження, які будуються за стандартними правилами застосування знаків арифметичних операцій: додавання "+", вирахування "-", множення "*" і розподілу "/". У стандарт SQL2 включена можливість виконання операцій додавання й вирахування над датами. У більшості СУБД також визначена операція конкатенації над строковими даними. У стандарті SQL2 визначені стандартні убудовані функції (таблиця 3).

Таблиця 3. Стандартні убудовані функції

Функція	Результат
BIT_LENGTH (<рядок>)	Кількість бітів в рядку
CAST (<значення> AS <тип даних>)	Значення, перетворене в заданий тип даних
CHAR_LENGTH (<рядок>)	Довжина рядку символів
CONVERT (<рядок> USING <функція>)	Рядок, перетворений відповідно до зазначеної функції
CURRENT_DATE	Поточна дата
CURRENT_TIME (<точність>)	Поточний час із зазначеною точністю
CORRENT_TIMESTAMP (<точність>)	Поточна дата й час із зазначеною точністю
LOWER (<рядок>)	Рядок, перетворений до верхнього регістра
OCTED_LENGTH (<рядок>)	Число байтів у рядку символів
POSITION (<перший рядок> IN <другий рядок>)	Позиція, з якої починається вхідження першого рядка в другу
SUBSTRING (<рядок> FROM n FOR <довжина>)	Частина рядка, що починається з n-го символу й має зазначену довжину
TRANSLATE (<рядок> USING <функція>)	Рядок, перетворений з використанням зазначеної функції
TRIM (BOTH <символ> FROM <рядок>)	Рядок, у якій вилучені всі перші й останні символи
TRIM (LEADING <символ> FROM <рядок>)	Рядок, у якій вилучені всі перші зазначені символи
TRIM (TRAILING <символ> FROM <рядок>)	Рядок, у якій вилучені останні зазначені символи
UPPER (рядок)	Рядок, перетворений до верхнього регістра

2 Створення та видалення баз даних.

Для прикладу у цьому розділі ми будемо використовувати базу даних з інформацією про роботу організації по розробці програмного забезпечення. Дамо базі даних ім'я Organization й опишемо структуру сформованих таблиць.

empl (e_id, e_name, job, d_id)

dep (d_id, d_name)

skills(e_id, skill)

Client (cl_id, f_name, addr, tel)

projects (pr_id, pr_theme, e_id, cl_id, data_start, data_finish, price)

proj_ emp (pr_id, e_id, royalty_share)

Ідентифікатори в MySQL

Ідентифікатор - це ім'я псевдоніма, бази даних, таблиці, стовпця або індексу. З його допомогою ви унікальним образом ідентифікуєте відповідний об'єкт. Перш ніж створювати бази даних і таблиці, ми повинні з'ясувати, які ідентифікатори припустимі в MySQL,

Загалом кажучи, в ідентифікаторах допускається використати будь-які символи, з урахуванням наступних виключень.

Вони не повинні містити лапок, а також символів ACSII(O) і A5CII255).

Імена баз даних можуть містити будь-які символи, припустимі для імен каталогів, але, по очевидних причинах, не повинні містити символи, що мають для каталогів спеціальне значення (це /, \ й .) .

Імена таблиць можуть містити будь-які символи, припустимі для імен файлів, за винятком символів . й /.

Всі ідентифікатори, крім імен псевдонімів, можуть містити до 64 символів. Імена псевдонімів можуть містити до 255 символів.

Одне дивне правило про ідентифікатори в MySQL говорить, що ви можете використати як ідентифікатори зарезервовані слова, якщо укладете їх у лапки. Наприклад, можна створити таблицю з ім'ям TABLE. Звичайно, існування такої можливості не означає, що ви повинні її використати. Саме навпаки - такої практики краще уникати. Навіть якщо це не бентежить вас при роботі із системою, це може "збентежити" програму unysqldump, часто використовувану для резервування даних.

Є короткий список зарезервованих слів, які MySQL дозволяє використати як ідентифікатори без лапок. Це суперечить стандарту ANSI для SQL, але є досить розповсюдженою практикою. Приклади, з якими ви будете зіштовхуватися найчастіше, - це DATE (дата) і TIMESTAMP (мітка дати/часу), використовувані як імена стовпців.

2.1 Створення бази даних

Після розробки структури було б логічно повідомити MySQL про те, що ми хочемо створити нову базу даних. Це робиться за допомогою оператора SQL

```
CREATE DATABASE ім'я бази даних;  
create database organization;
```

Переконайтеся в тім, що цей оператор виконав своє завдання, можна за допомогою команди: show databases; Ви повинні побачити базу даних organization у списку баз даних, наявних у системі.

Отже, тепер ми маємо порожню базу даних, що очікує створення своїх таблиць.

Перш ніж одержати можливість створювати таблиці або робити щось інше з нашою новою базою даних, ми повинні повідомити MySQL, що хочемо працювати із цією базою даних. Для цього використовується оператор use:

```
use organization;
```

Тепер база даних organization обрана, і всі дії, які ми будемо вживати надалі, за замовчуванням будуть застосовуватися саме до цієї бази даних.

2.2 Створення таблиць

Для створення таблиць у базі даних organization використовується оператор SQL CREATE TABLE.

У своїй типовій формі цей оператор виглядає так:

```
create table ім'я_таблиці (визначення таблиці) [type=тип_таблиці];
```

Як бачите, необхідно почати зі слів create table (створити таблицю), за яких повинне впливати ім'я таблиці, а потім указати деякий набір визначень для стовпців. Наприкінці оператора можна вказати тип механізму зберігання даних, що ми воліємо

використати.

Розглянемо приклад створення таблиць `dep` й `empl` :

```
mysql> create table dep
-> (d_id int(2) not null primary key,
-> d_name varchar(30) not null);
Query OK, 0 rows affected (0.60 sec)

mysql> create table empl
-> (e_id int(3) not null primary key,
-> e_name varchar(15) not null,
-> job varchar(15) not null,
-> d_id int(2) not null references dep(d_id));
Query OK, 0 rows affected (0.09 sec)

mysql> _
```

Інформацію про структуру кожної таблиці можна одержати за допомогою команди `describe`, наприклад:

```
mysql> describe dep;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| d_id  | int(2)        | NO   | PRI | NULL    |       |
| d_name | varchar(30)   | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.05 sec)

mysql> describe empl;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| e_id  | int(3)        | NO   | PRI | NULL    |       |
| e_name | varchar(15)   | NO   |     | NULL    |       |
| job   | varchar(15)   | NO   |     | NULL    |       |
| d_id  | int(2)        | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.02 sec)
```

Тепер, після того як ми вивчили приклад, розглянемо повний синтаксис оператора `CREATE TABLE`. У посібнику з MySQL приводиться наступна загальна форма цього оператора:

```
create [temporary] table [if not exists] ім'я_таблиці
[(визначення_стовпця, ...)] [опції_таблиці] [оператор_select]
```

або

```
create [temporary] table [if not exists] ім'я_таблиці
```

```
like ім'я_старої_таблиці;
```

визначення_стовпця:

ім'я_стовця *тип* [not null | null] [default *значення*]
[auto_increment] [primary key] [*визначення_посилання*]

або primary key (*ім'я_стовця_індексу*, ...)
або key [*ім'я_індексу*] (*ім'я_стовця_індексу*, ...)
або index [*ім'я_індексу*] (*ім'я_стовця_індексу*, ... ,)
або unique [index] [*ім'я_індексу*] (*ім'я_стовця_індексу*, ...)
або fulltext [index] [*ім'я_індексу*] (*ім'я_стовця_індексу*, ...)
або [constraint символ, foreign key [*ім'я_індексу*]
(*ім'я_стовця_індексу*, ...) [*визначення_посилання*]
або CHECK (*вираження*)

Розглянемо опції, зазначені в цій загальній формі.

Ключове слово TEMPORARY використовується для створення таблиць, які будуть існувати тільки в поточному сеансі роботи з базою даних й які будуть автоматично видалені, коли сеанс завершиться.

При використанні вираження IF NOT EXISTS таблиця буде створена тільки в тому випадку, якщо ще немає таблиці із зазначеним ім'ям.

Вираження LIKE *ім'я_старої_таблиці* можна використати для того, щоб створити нову таблицю з такою ж схемою, як у таблиці *ім'я_старої_таблиці*.

У дужках оператора CREATE TABLE ми повідомляємо стовпці, указуємо їхні типи й іншу інформацію про структуру таблиці. Найпростіше визначення стовпця складається з його імені й типу. У наступному розділі ми розглянемо типи, припустимі для стовпців.

У визначення стовпця можна також додати наступні елементи.

1 Оголосити для будь-якого стовпця або NOT NULL, або NULL, і це значить, що стовпцю або не дозволено містити значення NULL (NOT NULL), або дозволено (NULL). За замовчуванням стовпцям дозволяється містити значення NULL.

2 Оголосити для стовпця значення за замовчуванням, використовуючи ключове слово DEFAULT, за яким повинне впливати значення, що ми хочемо використати за замовчуванням.

3 Використати ключове слово `AUTO_INCREMENT`, як у попередньому прикладі, щоб генерувати порядковий номер. Значення, що автоматично генерується, буде на одиницю більшим, ніж поточне найбільше значення в таблиці. Перша уведена в таблицю рядок буде мати порядковий номер 1. У кожній таблиці дозволяється мати не більше одного стовпця `AUTO_INCREMENT`, і він повинен індексуватися. Як ви побачите пізніше, у попередніх прикладах ми не створювали ніяких індексів вручну, однак вони були автоматично створені для нас. Індеси автоматично створюються для стовпців, оголошених як `PRIMARY KEY`.

4 Оголосити стовець первинним ключем таблиці за допомогою вираження `PRIMARY KEY`.

5 Оголосити стовець зовнішнім ключем, використовуючи, як у попередньому прикладі, вираження `REFERENCES`.

Крім імен стовпців й їхніх типів, у цій частині оператора `CREATE TABLE` можна оголосити й іншу інформацію про стовпці.

6 Щоб створити первинний ключ, що складається з декількох стовпців, як у прикладі, необхідно вказати вираження `PRIMARY KEY`, за яким у дужках, через кому, впливають імена стовпців, що формує цей ключ. Точно так само можна оголосити й первинний ключ, що складається з одного стовпця. Стовпець `PRIMARY KEY` є унікальним індексованим стовпцем, що не може містити значення `NULL`.

7 `INDEX` й `KEY` є синонімами й означають, що зазначені стовпці (стовпець) будуть індексовані. Зверніть увагу, ці стовпці не обов'язково повинні містити унікальні значення.

8 `UNIQUE` використовується для вказівки того, що відповідний стовець повинен містити унікальні значення. Стовпці `UNIQUE` теж будуть індексовані.

9 `FULLTEXT` використовується для створення повнотекстових індексів на основі стовпців типу `TEXT`, `CHAR` або `VARCHAR`. Повнотекстові індекси можна використати тільки з таблицями `MyISAM`.

10 Вираження `FOREIGN KEY` дозволяє повідомляти зовнішні ключі точно так само, як і первинні.

Після закриваючої дужки можна вказати деякі додаткові характеристики таблиці. Якщо не вказати тип, створювана таблиця за замовчуванням буде таблицею MyISAM.

1 Вставка, видалення й оновлення даних

У цій лекції ви дізнаєтеся, як вставляти й змінювати дані в базі даних MySQL за допомогою операторів INSERT, DELETE й UPDATE, Ми розглянемо наступні питання.

- 1 Використання оператора INSERT.
- 2 Використання оператора DELETE.
- 3 Використання оператора UPDATE.
- 4 Завантаження даних за допомогою команди LOAD DATA INFILE.
- 5 Розширення: REPLACE й TRUNCATE.

Ми переходимо до складової DML (мова маніпулювання даними) мови SQL. Після того як ви навчитеся вставляти дані в базу даних, ми перейдемо до вивчення численних способів витягу даних з бази даних.

1.1 Використання INSERT

Оператор SQL INSERT використовується для додавання рядків у таблиці. Його вивчення ми почнемо із приклада:

```
insert into dep values  
(11, 'Administrative department'),  
(12, 'Projecting department') ,  
(13, 'Realization department'),  
(14, 'Testing department');
```

У першому рядку ми вказуємо таблицю, у яку необхідно вставити дані, - у цьому випадку це таблиця dep. Тут у таблицю додаються чотири рядки. Ви, напевно, пам'ятаєте, що таблиця dep має два стовпці: d_id (номер відділу) і D_NAME (назва відділу). (Це можна перевірити, виконавши команду describe department.

Якщо ви подивитеся на наведені тут оператори INSERT, то побачите, що при додаванні даних типу рядки або дати ці дані містяться в одинарні лапки, наприклад 'Administrative department'. Якщо дані, що вставляють же, є числовими, лапки використати не слід.

Але якщо поміщати дані в лапки, те що робити, коли самі дані містять лапки? Тоді перед одинарними лапками необхідно помістити зворотну косу рису (\), наприклад 'O\'Leary'.

Очевидно, тут виникає нове питання: "Що робити, якщо необхідно помістити в дані зворотну косу рису, не надаючи їй якого-небудь спеціального значення?" Тоді необхідно точно так само розмістити зворотну косу рису перед цією зворотною косою рисою, тобто замінити об-ратну косу рису двома (\\).

Витягають дані з бази даних за допомогою оператора SELECT. Надрукувавши `select * from ім'я_таблиці`; ви одержите всі дані, які в даний момент зберігаються в таблиці.

```
mysql> select * from dep;
+-----+-----+
| d_id | d_name                |
+-----+-----+
| 11   | Administrative department |
| 12   | Projecting department    |
| 13   | Realisation department   |
| 14   | Testing department       |
+-----+-----+
4 rows in set (0.01 sec)

mysql>
```

Загальна форма оператора INSERT з посібника з MySQL виглядає так:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] ім'я_таблиці [ (ім'я_стовпця, ...)]
VALUES ( (вираження | DEFAULT), ...), (...), ...
[ON DUPLICATE KEY UPDATE ім'я_стовпця=вираження, ...]
```

або INSERT [LOW_PRIORITY | DELAYED] [IGNORE]

```
[INTO] ім'я_таблиці [(ім'я_стовбця, ... )]  
SELECT ...
```

```
або INSERT [LOW_PRIORITY | DELAYED] [IGNORE]  
[ INTO] ім'я_таблиці  
SET ім'я_стовбця=(вираження | DEFAULT), ...  
[ ON DUPLICATE KEY UPDATE ім'я_стовбця=вираження, ...]
```

У всіх розглянуті вище прикладах використалася перша із зазначених форм оператора INSERT. Ключове слово INTO у них є необов'язковим. Його можна було б пропустити, використавши запит `insert dep values`, але ми порахували це більше важким для розуміння.

У першій формі оператора необхідно вказувати список значень стовпців для кожного рядка в тім же порядку, у якому стовпці розміщуються в таблиці. Наприклад, варто вказати спочатку значення стовпця `d_id`, а потім - стовпця `d_name`, оскільки саме такий є структура таблиці `dep`. Як видно із із, ця форма оператора INSERT дозволяє додавати в таблицю кілька рядків відразу.

Друга форма оператора INSERT закінчується оператором SELECT. Вона дозволяє додавати рядка не вручну, а витягати їх з іншої таблиці (або таблиць) бази даних і зберігати в даній таблиці.

Третя форма дозволяє вказати, у який стовпець варто помістити дані. Прикладом такого використання INSERT є наступний оператор:

```
insert into dep  
set d_id=15;
```

Ця форма дозволяє ввести тільки один рядок за раз, але зате не потребує вказувати значення всіх стовпців. У цьому випадку ми вказуємо значення тільки для стовпця `d_id`. Всі інші стовпці з або значення за замовчуванням, якщо вони визначені, або значення NULL. У цьому випадку значення `d_name` буде встановлено рівним NULL.

Оператор INSERT допускає кілька додаткових опцій.

За допомогою ключових слів `LOW_PRIORITY` й `DELAYED` можна призначити низький пріоритет операторові INSERT або відкласти його виконання. Обое вираження змусять систему відкласти вставку даних, поки який-небудь клієнт не спробує

прочитати дані з таблиці. Різниця між зазначеними опціями полягає в тім, що LOW PRIORITY блокує клієнт вставки, а DELAYED - немає. Це значить, що при використанні вставки LOW PRIORITY вам, можливо, прийде небагато почекати перед тим, як ви зможете продовжити виконання запитів у своєму клієнті. При використанні DELAYED ви одержите у відповідь ОК і зможете продовжувати виконання запитів, але при цьому потрібно пам'ятати, що вставка однаково не буде виконана, поки таблиця не почне використовуватись.

Ключове слово IGNORE може виявитися корисним, в основному, при додаванні декількох рядків відразу. Звичайно якщо один з рядків, які ви намагаєтесь вставити, породжує конфлікт із уже існуючим в іншому рядку значенням PRIMARY KEY або UNIQUE, виникає помилка й операція вставки відкидається. Якщо ж указати IGNORE, помилка буде ігнорована, операція вставки буде продовжена й буде зроблена спроба вставити наступний рядок.

"Можна призначити для стовпця значення за замовчуванням, указавши таке значення за допомогою DEFAULT.

"Вираження ON DUPLICATE KEY UPDATE пропонує елегантне рішення зі значень PRIMARY KEY й UNIQUE. За цим вираженням повинне впливати вираження UPDATE, за допомогою якого можна змінити первинний ключ іди унікальне значення у вже існуючому рядку таблиці, щоб усунути конфлікт із новим значенням.

Наступний приклад демонструє типовий варіант використання вираження ON DUPLICATE KEY UPDATE:

```
create table warning
(
  E_id int primary key not null references empl (e_id) ,
  count int default 1
);
insert into warning (e_id)
values (6651)
on duplicate key update count=count+1;
```

Зазначене вираження зручно використати в тих випадках, коли необхідно запам'ятовувати не тільки унікальні події, але й виконувати якісь дії (наприклад, збільшувати лічильник) для не унікальних подій. Прикладом такого випадку може бути будь-який варіант реєстрації, але в контексті нашої бази даних можна розглянути запис у таблицю `warning` інформації про службовців, що одержали попередження.

Щоб записати службовцеві попередження, можна використати оператор `INSERT`, наведений вище. Оскільки лічильник `count` має значення за замовчуванням, рівне 1, і ми не вказали для нього значення в `INSERT`, це значення виявиться рівним 1 при першій вставці для кожного `e_id`. Наступні вставки для того ж `e_id` ініціюють роботу вираження `ON DUPLICATE KEY UPDATE`, результатом чого буде збільшення лічильника.

1.2 Використання REPLACE

Оператор `REPLACE` аналогічний операторові `INSERT`, за винятком того, що при виникненні конфлікту значень ключа новий рядок, що додається, замінить старий.

Загальна форма оператора `REPLACE` з посібника з MySQL наведена нижче.

```
REPLACE [LOW_PRIORITY | DELAYED]
```

```
[INTO] ім'я_таблиці [(ім'я_стовпця, ...)]
```

```
VALUES (вираження, ...), (...), ...
```

```
або REPLACE [LOW_PRIORITY | DELAYED]
```

```
[INTO] ім'я_таблиці [(ім'я_стовпця, ...)]
```

```
SELECT ...
```

```
або REPLACE [LOW_PRIORITY | DELAYED]
```

```
[INTO] ім'я_таблиці
```

```
SET ім'я_стовпця=вираження, ім'я_стовпця=вираження, ...
```

```
[ON DUPLICATE KEY UPDATE ім'я_стовпця=вираження, ...]
```

Аналогічність операторові `INSERT` повинна бути очевидної.

1.3 Використання DELETE

Оператор SQL DELETE дозволяє видаляти рядки з таблиць. Наприклад:

```
delete from dep;
```

У такій формі оператор DELETE видаляє всі рядки з таблиці dep. Можна обмежитися видаленням тільки певних рядків, якщо використовувати вираження WHERE. Наприклад:

```
delete from dep where d_name='Testing department';
```

У цьому випадку будуть вилучені рядки, що відповідають критерію, зазначеному у вираженні where. Тут будуть вилучені тільки рядки, у яких для назви відділу зазначене значення 'Testing department'.

Як правило, всі рядки відразу з таблиці не видаляють. Але через те, що зазначена сама коротка форма оператора delete є припустимою, її іноді трапляється ввести помилково, без вираження WHERE. Ви можете убезпечити себе й свою базу даних від таких ексцесів, включивши опцію -isafe-updates або -i-am-a-dummy з командного рядка клієнта mysql. Ці опції не дозволять видаляти (або оновляти) рядка без вказівки обмежень на ключ у вираженні WHERE; необхідно буде вказати, що ви хочете видалити тільки рядка, що містять певні значення ключа.

Загальна форма оператора DELETE з посібника з MySQL наведена нижче:

```
DELETE [LOW_PRIORITY] [QUICK] FROM ім'я_таблиці  
[WHERE визначення_where]  
[ORDER BY ...]  
[LIMIT рядка]
```

```
або DELETE [LOW_PRIORITY] [QUICK] ім'я_таблиці [ . * ]  
[ , ім'я_таблиці [ . * ] ... ]  
FROM таблиці-посилання  
[WHERE визначення_where]
```

```
або DELETE [LOW_PRIORITY] [QUICK]
```

```
FROM ім'я_таблиці[.*] [, ім'я_таблиці[.*] . . .]  
USING таблиці-посилання  
[WHERE визначення_where]
```

У наведені вище прикладах використалася перша форма цього оператора. Інші дві форми оператора DELETE призначені для видалення рядків з однієї або декількох таблиць із посиланнями на інші таблиці. Наприклад:

```
delete empl, skills  
from empl, skills, dep  
where empl. e_id = skills. e_id  
and empl. d_id = dep . d_id  
and dep. D_name='Testing department';
```

У цьому прикладі віддаляється інформація про всіх службовців, які працюють у відділі тестування, і стираються всі записи про їхній кваліфікації. Зверніть увагу на те, що тут будуть вилучені рядки з таблиць empl й skills (таблиці, які присутні у вихідному списку delete), але не з таблиці dep (яка зазначена тільки в списку from).

Рядки будуть віддалятися з таблиць, зазначених у вихідному списку вираження delete, тоді як таблиці зі списку вираження from використаються для пошуку даних, і рядка з них віддалятися не будуть, якщо відповідної таблиці в списку delete немає.

Загальна форма оператора DELETE допускає ще кілька опцій.

Ключове слово LOW_PRIORITY тут працює так само, як в операторі INSERT.

Вказівка QUICK може прискорити виконання оператора DELETE, дозволяючи MySQL не робити якусь роботу з упорядкування індексів вчасно видалення з таблиць;

Вираження ORDER BY дозволяє вказати порядок, у якому повинні віддалятися рядка. Воно особливо корисно в сполученні з вираженням LiMiT-можна, наприклад, видалити n самих старих рядків таблиці.

Вираження LiMi дозволяє встановити максимальне число рядків, які дозволяється видалити за допомогою оператора DELETE. Це вираження виявляється корисним або в парі з вираженням ORDER BY, або для того, щоб не допустити випадкове видалення занадто великого числа рядків.

1.4 Використання TRUNCATE й UPDATE

Оператор TRUNCATE дозволяє видалити всі рядки таблиці. Мер:

```
truncate table empl;
```

Цей запит видалить із всіх службовців з таблиці empl. Він працює швидше з DELETE, оскільки в цьому випадку стара таблиця знищується й замість її створюється порожня нова. Необхідно тільки пам'ятати, що оператор TRUNCATE не забезпечує безпеки транзакцій.

Оператор SQL UPDATE можна використати для зміни рядків, що вже зберігаються в базі даних. Припустимо, наприклад, що в одного зі співробітників помінялася посада:

```
update empl  
set job='Administrator DB'  
where e_id=132 ;
```

Цей оператор змінює значення стовпця job для службовця з номером 132.

Загальна форма оператора UPDATE з посібника з MySQL наведена нижче.

UPDATE [LOW_PRIORITY] [IGNORE] *ім'я_таблиці*

SET *ім'я_стовця1=вираження1* [,

ім'я_стовця2=вираження2 ...]

[WHERE *визначення_where*]

[ORDER BY ...]

[LIMIT *рядка*]

або UPDATE [LOW_PRIORITY] [IGNORE] *ім'я_таблиці* [,

ім'я_таблиці ...]

SET *ім'я_стовця1=вираження1* [,

ім'я_стовця2=вираження2, ...] [WHERE *визначення_where*]

Оператор UPDATE багато в чому аналогічний операторові DELETE.

Можна використати необов'язкове вираження WHERE, щоб оновити тільки деякі або всі рядки.

Друга з наведених вище версій оператора UPDATE призначена для відновлення декількох таблиць. Вона працює аналогічно багатотабличної версії оператора видалення, яку ми обговорювали вище. Зверніть увагу, що обновлятися будуть тільки ті стовпці, які ви перелічите в списку вираження SET.

Всі інші вираження оператора UPDATE ми вже розглядали. LOW_PRIORITY й IGNORE працюють так само, як в операторі INSERT, а ORDER BY й LIMIT - так само, як в операторі DELETE.

Контрольні питання

1 Які дві складові частини, необхідні для роботи з базами даних, присутні в мові SQL? Поясніть їхнє призначення.

2 Який ряд типів даних визначений у мові SQL1?

3 Які додаткові типи даних розширюють можливості мови?

4 Які константи визначені у стандарті мови?

5 Які стандартні вбудовані функції використовуються в запитах?

6 Який з наступних операторів успішно вставить рядок у таблицю employee?

a) insert into employee Values

set employeei=NULL, name='Laura Thomson1, job='Programmer',
departmenti=128;

б) insert employee values

(NULL, 'Laura Thomson1, 'Programmer1, 128) ;

b) insert into employee values

(NULL, Laura Thomson, Programmer, 128);

r) insert employee values

(NULL, 'Laura O'Leary', 'Programmer1, 128) ;

7 Оператор REPLACE

а) аналогічний операторові INSERT, за винятком того, що при конфлікті ключів він замінює старий рядок нової;

б) аналогічний операторові INSERT, за винятком того, що при конфлікті ключів він залишає старий рядок, а нову відкидає;

в) аналогічний операторові UPDATE, за винятком того, що при конфлікті ключів він замінює старий рядок нової;

г) аналогічний операторові UPDATE, за винятком того, що при конфлікті ключів він залишає старий рядок, а нову відкидає.

8 За замовчуванням значення полів у файлах даних розділяються

а) комами;

б) пробілами;

в) знаками табуляції;

г) каналами.

9 Опція LOCAL в операторі LOAD DATA INFILE повідомляє про те, що

а) клієнт і сервер виконуються на одній машині;

б) файл із даними перебуває на сервері;

в) файл із даними перебуває на машині клієнт

Лекція № 10

з навчальної дисципліни Організація баз даних

Тема Оператори мови SQL

Мета Дати характеристику основних груп операторів мови SQL: визначення даних, маніпулювання даними, операторів мови запитів, керування транзакціями та адміністрування даних

Час: 2 години.

План проведення лекції

Основна частина (викладення навчальних питань лекції)

1 Оператори мови SQL

1.1 Оператори визначення даних

1.2 Оператори маніпулювання даними

1.3 Основні оператори мови запитів

1.4 Оператори керування транзакціями та адміністрування даних

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посіб-ник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Оператори мови SQL

Мова SQL містить у собі оператори різних категорій. Любою SQL-оператор складається із зарезервованих слів і слів, обумовлених користувачем відповідно до встановлених синтаксичних правил. Як й у багатьох мовах програмування, більшість компонентів операторів мови не чутливі до регістра. Виключення із цього правила становлять символічні дані, при завданні яких необхідно пам'ятати про регістр і використати той, котрий необхідний для подання даних. Для запису операторів у мові прийнятий вільний формат, що дозволяє за допомогою відступів і вирівнювань додати SQL-програмі більше читабельний вид. При записі операторів рекомендується дотримуватися наступних правил:

- кожна фраза в операторі повинна починатися з нового рядка;
- початок кожної фрази повинне бути вирівняне з початком інших фраз оператора;
- кожна частина фрази повинна починатися з нового рядка з деяким відступом відносно початку всієї фрази, що дозволить виділити підлеглі частини.

Для запису операторів застосовуються деякі угоди:

- для запису зарезервованих слів використовуються прописні букви;
- для запису обумовлених користувачем слів використовуються малі літери;
- вертикальна риса "|" указує на необхідність вибору одного з декількох значень;
- фігурні дужки визначають обов'язковий елемент;
- квадратні дужки визначають необов'язковий елемент;
- "..." використовується для вказівки необов'язкової можливості повторення конструкції, від нуля до декількох разів.

1.1 Оператори визначення даних

Оператори визначення даних (див. таблиця 1) застосовуються для опису структур даних. До складу цієї категорії входять наступні оператори: CREATE TABLE, DROP TABLE, ALTER TABLE, CREATE VIEW, ALTER VIEW, DROP VIEW.

Таблиця 1. Оператори визначення даних

Оператор	Пояснення
CREATE TABLE	Створити таблицю
DROP TABLE	Видалити таблицю
ALTER TABLE	Змінити таблицю
CREATE VIEW	Створити представлення
ALTER VIEW	Змінити представлення
DROP VIEW	Видалити представлення

1.2 Оператори маніпулювання даними

Оператори маніпулювання даними, що утворюють наступну категорію операторів, призначені для заповнення таблиць даними й для відновлення завантаженої в них інформації. До них ставляться наступні оператори: DELETE, INSERT, UPDATE (див. таблиця 2).

Таблиця 2. Оператори маніпулювання даними

Оператор	Пояснення
DELETE	Видаляє одну або кілька рядків, що відповідають умовам фільтрації, з базової таблиці
INSERT	Вставляє один рядок у базову таблицю
UPDATE	Обновляє значення одного або декількох стовпців в одній або декількох рядках, що відповідають умовам фільтрації

1.3 Основні оператори мови запитів

Для вибірки інформації з бази даних призначена мова запитів, що у мові SQL представлена одним оператором SELECT (див. таблиця 3 16).

Таблиця 3. Мова запитів

Оператор	Пояснення
SELECT	Вибирає рядка; оператор, що дозволяє сформувати результуючу таблицю, що відповідає запиту

1.4 Оператори керування транзакціями та адміністрування даних

Крім зазначених категорій операторів, призначення яких не складно представити, прочитавши пояснення в таблицях, необхідно виділити ще два: оператори керування транзакціями (див. таблиця 4) і засобу адміністрування даних (див. таблиця 5).

Таблиця 4 Керування транзакціями

Оператор	Пояснення
COMMIT	Завершити транзакцію - завершити обробку інформації, об'єднану в транзакцію
ROLLBACK	Відкотити транзакцію - скасувати зміни, проведені в ході виконання транзакції
SAVEPOINT	Зберегти проміжну крапку виконання транзакції - зберегти проміжний стан БД, позначити його для того, щоб можна було надалі до нього повернутися

Таблиця 5. Адміністрування даних

Оператор	Пояснення
ALTER DATABASE	Змінити набір основних об'єктів у базі даних, обмежень, що стосується всієї бази даних
ALTER DBAREA	Змінити раніше створену область зберігання
ALTER PASSWORD	Змінити пароль для всієї бази даних
CREATE DATABASE	Створити нову базу даних
CREATE DBAREA	Створити нову область зберігання й зробити її доступною для розміщення даних
DROP DATABASE	Видалити існуючу базу даних
DROP DBAREA	Видалити існуючу область зберігання (якщо в ній на дійсний момент не розташовуються активні дані)
GRANT	Надати права доступу на ряд дій над деяким об'єктом БД
REVOKE	Позбавити прав доступу до деякого об'єкта або деяких дій над об'єктом

У комерційних СУБД набір основних операторів розширений. У більшість СУБД включені оператори визначення й видалення індексу запуску збережених процедур й оператори визначення тригерів.

Починати знайомство з даною мовою прийнято з розгляду можливостей мови запитів, що у мові SQL представлений одним оператором SELECT, тому що цей потужний оператор є самим складним. До того ж його можна використати разом з операторами маніпулювання даними.

Контрольні питання

- 1 Дайте характеристику різних категорій операторів, які містить у собі мова SQL:
 - оператори визначення даних;
 - оператори маніпулювання даними;
 - основні оператори мови запитів;
 - оператори керування транзакціями та адміністрування даних.

Лекція № 11

з навчальної дисципліни Організація баз даних

- Тема** Формування простих запитів MySQL. Робота з стовпцями та рядками таблиці БД при формування простих запитів
- Мета** Пояснити принцип формування простих запитів за допомогою оператора SE-LECN. Розглянути роботу опцій оператора SELECN, за допомогою яких можна робити відбір стовпців та рядків таблиці, видалення повторень в рядках таблиці, групування даних таблиці та обмеження результатів пошуку даних.

Час: 2 години.

План проведення лекції

1 Прості запити MySQL

- 1.1 Можливості оператора SELECT
- 1.2 Відбір стовпців
- 1.3 Відбір рядків за допомогою WHERE
- 1.4 Видалення повторень за допомогою DISTINCT
- 1.5 Використання GROUP BY
- 1.6 Обмеження результатів пошуку за допомогою LIMIT

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BVH, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посіб-ник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Прості запити MySQL

Ми вже розглянули питання розробки, створення й поповнення бази даних в MySQL. У цій лекції й двох наступних ви дізнаєтесь, як отримувати дані з бази даних, розглянемо деякі можливості оператора SELECT, що, напевно, можна назвати найважливішим оператором SQL. Саме цей оператор використається для витягу рядків з однієї або декількох таблиць бази даних.

Розглянувши питання:

- 1 Огляд можливостей оператора SELECT.
- 2 Прості запити.
- 3 Вибір стовпців.
- 4 Псевдоніми стовпців.
- 5 Вибір рядків за допомогою вираження WHERE,
- 6 Використання вираження GROUP BY.
- 7 Вибір груп за допомогою вираження HAVING.
- 8 Сортування результатів пошуку за допомогою вираження ORDER BY.
- 9 Обмеження результатів пошуку за допомогою вираження LIMIT.

ми зрозуміємо, як отримувати потрібні дані з однієї таблиці.

1.1 Можливості оператора SELECT

Загальна форма оператора SELECT виглядає так:

```
SELECT стовпці
FROM таблиці
[WHERE умови]
[GROUP BY група]
[HAVING групові_умови]
[ORDER BY сортування_стовпців]
[LIMIT межі];
```

Це - не вичерпний синтаксис оператора SELECT. Але наведена тут форма дає загальне подання про форму оператора. У цій лекції будуть розглянуті тільки зазначені опції.

Оператор SELECT має багато опцій. Їх можна використати або не використати, але вони повинні вказуватися саме в тому порядку, у якому вони наведені тут.

Найпростішої приклад оператора SELECT виглядає в такий спосіб:

```
select. * from dep;
```

Якщо виконати цей запит для даних, наявних у нашій базі даних organization, буде отриманий такий результат:

```
mysql> use organization;
Database changed
mysql> select * from dep;
+-----+-----+
| d_id | d_name                |
+-----+-----+
| 11   | Administrative department |
| 12   | Projecting department    |
| 13   | Realisation department   |
| 14   | Testing department       |
+-----+-----+
4 rows in set (0.32 sec)

mysql>
```

За допомогою цього запиту були обрані всі дані із зазначеної таблиці - у цьому випадку всі рядки й всі стовпці з таблиці dep.

Звичайно, користь реляційної бази даних полягає не стільки в можливості повертати всі розмішені в ній дані, а в можливості знаходити потрібну конкретну частину інформації.

1.2 Вибір стовпців

Першим обмеженням, що ми можемо використати, є вказівка набору стовпців. У попередньому запиті (select * from dep) символ * означає "всі стовпці таблиці". Замість * можна вказати список стовпців, значення яких ми хотіли б одержати. Це може бути як один стовпець, так і деяка підмножина або навіть повний набір стовпців таблиці в будь-якому порядку, що влаштовує нас. Імена стовпців повинні зазначені у вигляді списку імен стовпців, розділених комами.

Наприклад, наступний оператор вибирає тільки значення полів `e_id` й `f_name` з таблиці `empl`:

```
select f_name, e_id from empl;
```

Якщо виконати цей запит для нашої бази даних з інформацією про службовців, вийде приблизно такий результат:

```
mysql> select e_name, e_id from empl;;
+-----+-----+
| e_name | e_id |
+-----+-----+
| Drovko | 111  |
| Ivanov | 112  |
| Konov  | 113  |
| Kornienko | 114 |
| Koval  | 121  |
| Palkin | 122  |
| Petrov | 123  |
| Sidorov | 131 |
| Turov  | 132  |
| Vertko | 133  |
+-----+-----+
10 rows in set (0.02 sec)
```

Як бачите, запит повернув значення тільки двох зазначених нами стовпців, Зверніть увагу на те, що у виводі ці стовпці показані в тім порядку, у якому ми вказали їх у нашому запиті, а не в тім порядку, у якому вони існують у схемі бази даних.

Ще однією можливістю, який ми поки не користуємося, є можливість прямо вказати базу даних і таблицю, які ми маємо на увазі. Наприклад, можна представити посилання на стовпець `e_name` з таблиці `empl` у вигляді `empl . e_name`:

```
select empl . e_name from empl;
```

Так само можна уточнити й базу даних, про таблицю з якої говориться в нашому запиті:

```
select e_name
from organization .empl;
```

(Цей запит повинен видати точно такий же результат, як і попередній.)

Тут ми робимо явне посилання на таблицю `empl` з бази даних `organization`, використовуючи для цього нотацію виду *база_даних. таблиця*.

Якщо потрібно, разом з базою даних і таблицею можна вказати й приналежній таблиці стовпець, Той же приклад з використанням нотації *база_даних. таблиця . стовпець* можна записати так:

```
select organization. empl . e_name from empl;
```

Для зазначених тут простих запитів такі можливості синтаксису здаються за-
надтими, але згодом, при використанні більше складних запитів, це дозволить нам
уникнути двозначності при вказівці джерела з інформації.

Тепер ми повинні обговорити концепцію псевдонімів для імен стовпців і таб-
лиць.

В операторі SELECT можна перейменувати стовпці або призначити ім'я утри-
муючим їхнім вираженням, у результаті чого у висновку оператора буде показане
нове ім'я.

Наприклад, можна ввести наступний запит:

```
select e_name as employee_Name from empl;
```

У цьому випадку ми призначили стовпцю name нове ім'я employeeName тільки
для цього запиту. Результат виконання цього запиту для бази даних employee буде
наступним:

```
mysql> select e_name as employee_Name from empl;
+-----+
| employee_Name |
+-----+
| Drovko        |
| Ivanov        |
| Konov         |
| Kornienko     |
| Koval         |
| Palkin        |
| Petrov        |
| Sidorov       |
| Turov         |
| Vertko        |
+-----+
10 rows in set (0.00 sec)
```

Ідентифікатори типу employee_Name називають псевдонімами. Є певні правила
щодо того, що можна й чого не можна робити із псевдонімами, і ми ці правила розг-
лянемо.

У цьому конкретному прикладі користь від використання псевдоніма не занадто
велика. Ви зможете оцінити переваги використання псевдонімів тоді, коли ми поч-
немо створювати складні запити й запити, у яких застосовуються обчислення.

Можна також використати псевдоніми для таблиць:

```
select e.e_name from empl as e;
```

Результат виконання цього запиту буде аналогічний результату виконання запиту без псевдонімів. Відповідна нотація виявиться корисною, коли в наступній лекції ми почнемо створювати запити із вказівкою декількох таблиць.

В останніх двох прикладах ключове слово AS не є обов'язковим. Можна було б просто написати

```
select e_name employee_Name from empl;
```

та

```
select e.e_name  
from empl e;
```

Можна створювати запити в кожній із цих форм. Це - справа стилю. Індивідуальний стиль програмування в SQL варіює точно так само, як й в інших мовах програмування.

1.3 Відбір рядків за допомогою WHERE

Ми навчилися вибирати всі дані з таблиці або з конкретних її стовпців. Тепер ми з'ясуємо, як вибирати потрібні рядки. Це може пригодиться, оскільки часто потрібно вибрати тільки запису, що задовольняють певним критеріям пошуку. Це особливо важливо тоді, коли необхідно витягти всього кілька потрібних рядків з дуже великої таблиці.

Поставлене завдання можна вирішити за допомогою вираження WHERE в операторі SELECT. От простий приклад:

```
select e_id, e_name  
from empl  
where job='Programer' ;
```

(Згадаєте, між іншим, що запити можна розміщати на декількох рядках. Кожен запит закінчується крапкою з коми. Тут ми розмістили оператор SELECT у декількох рядках для того, щоб його було легше читати,)

Результат виконання цього запиту для бази даних employee буде наступним:

```
mysql> select e_id, e_name from empl
-> where job='Programer';
```

e_id	e_name
113	Konov
114	Kornienko
121	Koval
122	Palkin
123	Petrov
133	Vertko

```
6 rows in set (0.01 sec)
```

Ми використали умову у вираженні WHERE, щоб знайти тільки ті рядки в таблиці, які задовольняють зазначеним критеріям, - у цьому випадку це повинні бути службовці, що працюють програмістами.

Зверніть увагу на те, що крім цієї умови ми вказали також список необхідних стовпців, щоб одержати тільки ту інформацію, який ми цікавимось.

У цьому випадку ми використаємо у вираженні WHERE перевірку рівності. Зверніть увагу на те, що в SQL для перевірки з використовується =. Цим SQL відрізняється від багатьох інших мов програмування, де в таких випадках використовується == або eq.

Є величезне число функцій, які можна використати у вираженні WHERE, і ми докладно розглянемо їх пізніше. Тут же ми згадаємо тільки найбільше часто використовувані оператори.

- 1 Рівність, або ==, використання якого ми бачили вище.
- 2 Нерівність, позначувана знайомимми != або <>.
- 3 Варіанти > (більше), < (менше), >= (більше або дорівнює) і <= (менше або дорівнює).
- 4 IS NULL й IS NOT NULL які використовуються для того, щоб перевірити, чи є значення значенням NULL. Ви не зможете зробити це за допомогою перевірки типу значення=NULL.
- 5 Знаки арифметичних операцій, які звичайно використовуються з операторами. Наприклад, може знадобитися перевірка типу значення > інше_значення*10.
- 6 Стандартні логічні оператори AND (И), OR (АБО) і NOT (НЕ), які можна використати для угруповання умов перевірки. Пріоритет у них нижче, ніж в операторі

рів порівняння, так що, наприклад, `salary > 30000 AND salary < 50000` буде працювати так, як і варто очікувати.

У додавання до операторів у деяких прикладах ми будемо використати функцію `count ()`, що дає можливість порахувати число рядків, повернутих запитом. Наприклад:

```
select count(*) from empl;
```

Цей запит повідомить, скільки рядків є в таблиці `empl`. Нарешті, можна управляти пріоритетом операцій, групуючи вираження за допомогою дужок.

От приклад небагато більше складного запиту з використанням вираження **WHERE**:

```
select pr_theme, price from projects
where e_id=121 and price > 5500;
```



```
mysql> select pr_theme, price from projects
-> where e_id=121 and price>5500;
+-----+-----+
| pr_theme | price |
+-----+-----+
| 'Decans Office' | 5678.55 |
+-----+-----+
1 row in set (0.05 sec)
```

Цей запит повернув список тим всіх проектів, керівником яких є співробітник 121 і вартість яких більше 5500.

Отут варто зробити одне важливе зауваження: використати псевдоніми стовпців у вираженні **WHERE** не дозволено. Необхідно використати тільки оригінальне ім'я стовпця. Це - обмеження ANSI SQL. Причина полягає в тім, що в момент розгляду умови **WHERE** значення псевдоніма стовпця може бути невідомо.

1.4 Видалення повторень за допомогою **DISTINCT**

У запиті можна використати ключове слово **DISTINCT**, щоб результат не містив повторень уже наявних значень. Розглянемо, наприклад, запит

```
select job from empl;
```

який поверне наступні дані:

```
mysql> select job from empl;
+-----+
| job      |
+-----+
| AdministratorDB |
| System admin |
| Programmer  |
| Programmer  |
| Programmer  |
| Programmer  |
| Programmer  |
| AdministratorDB |
| System admin |
| Programmer  |
+-----+
10 rows in set (0.00 sec)
```

Як бачите, значення Programmer тут є присутнім шість разів. Причина в тім, що це значення втримується в шести рядках. Даний запит просто повернув повний список значень стовпця job зазначеної таблиці.

Тепер розглянемо запит

```
select distinct job from empl;
```

Він поверне наступні рядки:

```
mysql> select distinct job from empl;
+-----+
| job      |
+-----+
| AdministratorDB |
| System admin |
| Programmer  |
+-----+
3 rows in set (0.12 sec)
```

Тут повторення були вилучені.

У цьому конкретному прикладі різниця не здається занадто значною - другий результат лише небагато лаконічніше, але в принципі ненабагато краще. Різниця може виявитися більше помітної для більших таблиць із безліччю повторень, але результат однаково буде представляти всю потрібну інформацію.

З іншого боку, розглянемо наступний приклад:

```
select count (job) from empl;
```

```
mysql> select count(job) from empl;
+-----+
| count(job) |
+-----+
|          10 |
+-----+
1 row in set (0.05 sec)
```

Цей запит повідомляє, що в стовпці job є 10 значень. Результат почасти оманливий. Він виразно не говорить про те, що в стовпці job є 6 різних значень, оскільки попередня перевірка показала, що різних значень усього три,

Такий запит цілком імовірно ввести помилково, коли насправді ви маєте на увазі

```
select count(distinct job) from empl;
```

Результат цього запиту буде наступним:

```
mysql> select count(distinct job) from empl;
+-----+
| count(distinct job) |
+-----+
|                    3 |
+-----+
1 row in set (0.03 sec)
```

Тут повідомляється про те, скільки різних значень утримується в стовпці job, що звичайно дає більше важливу інформацію.

1.5 Використання GROUP BY

Наступної з розглянутих нами виражень є GROUP BY. Воно дозволяє розподілити витягають строки, що, по групах і виявляється особливо корисним при його використанні в комбінації з функціями, застосовуваними до груп рядків. Тут ми згадаємо тільки одну таку функцію - count ().

Розглянемо наступний запит:

```
select count(*), job from empl
group by job;
```

Цей запит підраховує число службовців по групах посад, тобто з'ясовує число

службовців, що займають та або інша посада. Виконавши цей запит, одержимо наступний результат:

```
mysql> select count(*), job from empl
-> group by job;
+-----+-----+
| count(*) | job          |
+-----+-----+
| 2        | AdministratorDB |
| 6        | Programmer     |
| 2        | System admin   |
+-----+-----+
3 rows in set (0.00 sec)
```

Тут слід зазначити, що в MySQL й ANSI SQL вираження GROUP BY працює по-різному. В ANSI SQL необхідно групувати по всіх стовпцях, зазначеним у вихідному вираженні SELECT. В MySQL у вираженні SELECT дозволяється вказувати додаткові стовпці, що не входять у вираження GROUP BY.

MySQL дозволяє також сортувати порядок груп, у якому вони повинні бути представлені в результаті. За замовчуванням заданий зростаючий порядок. Щоб у нашому останньому запиті результати були представлені в зворотному порядку, запит варто змінити в такий спосіб:

```
select count(*) job from empl
group by job desc;
```

Результат буде приблизно наступним:

```
mysql> select count(*), job from empl
-> group by job desc;
+-----+-----+
| count(*) | job          |
+-----+-----+
| 2        | System admin |
| 6        | Programmer   |
| 2        | AdministratorDB |
+-----+-----+
3 rows in set (0.00 sec)
```

Як бачите, назви посад тепер приводяться в порядку, зворотному алфавітному. Для вказівки зростаючого порядку можна використати ASC, але це значення використається за замовчуванням, тому вказувати його не потрібно.

Наступним вираженням в операторі SELECT є вираження HAVING. GROUP BY з вираженням HAVING подібно SELECT з вираженням WHERE. Наприклад:

```
select count(*), job from empl  
group by job having count(*)=2;
```

Такий запит вибере всі наявні в компанії посади, на яких працює рівно по двох службовців.

```
mysql> select count(*), job from empl  
-> group by job having count(*)=2;  
+-----+-----+  
| count(*) | job      |  
+-----+-----+  
| 2        | AdministratorDB |  
| 2        | System admin  |  
+-----+-----+  
2 rows in set (0.00 sec)
```

Досвід показує, що починаючи користувачі SQL часто плутають вираження WHERE й HAVING. Вираження WHERE використовується в запиті для перевірки умов, що стосуються окремих рядків, а умови HAVING застосовуються до груп.

1.6 Сортювання результатів пошуку за допомогою ORDER BY

Наступним вираженням в операторі SELECT є вираження ORDER BY. Воно дає можливість відсортувати рядка результату по одному або декількох стовпцях. Сортювати при цьому можна як по зростанню (позначається ASC), так і по убутанню (позначається DESC), Наприклад:

```
select * from empl  
order by job asc, e_name desc;
```

У результаті будуть обрані весь рядки з таблиці empl і відсортовані за значенням job за абеткою. Якщо при цьому два або трохи службовці будуть мати однакові посади, то відповідні рядки будуть відсортовані по імені у зворотному порядку. Результат буде наступним:

```
mysql> select * from empl
-> order by job asc, t_name desc;
ERROR 1054 (42S22): Unknown column 't_name' in 'order clause'
mysql> select * from empl
-> order by job asc, e_name desc;
```

e_id	e_name	job	d_id
131	Sidorov	AdministratorDB	12
111	Drovko	AdministratorDB	12
133	Vertko	Programer	13
123	Petrov	Programer	13
122	Palkin	Programer	14
121	Koval	Programer	13
114	Kornienko	Programer	14
113	Konov	Programer	14
132	Turov	System admin	11
112	Ivanov	System admin	11

```
10 rows in set (0.01 sec)
```

Якщо для стовпця в ORDER BY ви не вкажете ні ASC, ні DESC, за замовчуванням буде матися на увазі ASC. Зверніть увагу, під час відсутності вираження ORDER BY неможливо зробити ніяких припущень про те, у якому порядку будуть представлені у висновку повернуті рядки таблиці.

1.7 Обмеження результатів пошуку за допомогою LIMIT

Вираження LIMIT використовується для обмеження числа й діапазону рядків, що повертають запитом. Розглянемо, наприклад, що впливає запит:

```
select * from skills
limit 5;
```

Цей запит поверне тільки перші п'ять рядків, що задовольняють критеріям пошуку. У даному конкретному випадку це будуть просто перші п'ять рядків таблиці:

```
mysql> select * from skills
-> limit 5;
```

e_id	skill	license
111	MySQL	NULL
111	Linux	NULL
111	Java	NULL
111	UML	NULL
112	Windows	NULL

```
5 rows in set (0.10 sec)
```

Можна також указати деяка підмножина рядків, відмінне від перших п. Якщо, наприклад, необхідно одержати рядки 6-8, можна замість попереднього запиту використати такий:

```
select * from skills
```

```
limit 5, 3;
```

Коли у вираженні межі вказуються два параметри, перший з них визначає зміщення (початкову крапку), а другий - максимум для числа рядків, які необхідно повернути. При вказівці одного параметра, як у попередньому прикладі, цей параметр характеризує максимум для числа рядків, які необхідно повернути.

При визначенні зсуву нумерація рядків починається з нуля (саме тому в останньому прикладі для шостого рядка ми вказали зсув 5). Наш перший приклад використання LIMIT повертає рядка 0-4, а в другому вибиралися рядки 5-7. Якщо як другий параметр вказати -1, запит поверне рядка, починаючи з рядка, певної зсувом, до кінця таблиці.

Вираження LIMIT звичайно використовується разом з ORDER BY, щоб за допомогою порядку додати необхідний зміст повертає набору, що, рядків. Не забувайте про те, що без вираження ORDER BY повернуті записи не будуть підкорятися якому-небудь логічному порядку.

Це вираження виявляється особливо із при створенні Web- або GUI-додатків MySQL, оскільки забезпечує простий механізм посторінкового подання результатів.

Контрольні питання

1. Який з наступних запитів вибере всі рядки з таблиці client?

a) `select * from client`

`where cl_id=2;`

б) `select cl_id, f_name, addr, tel`
`from client;`

в) `select * from client`
`limit 1;`

г) `select all from client;`

2. Який з наступних запитів вибере всіх програмістів з таблиці empl?

a) `select *`

```
from empl
where job='Programer' ;
```

б)

```
select *
from empl
having job=' Programer';
```

в)

```
select *
from empl
where job=' Programer'
group by job
having job=' Programer' ;
```

г)

```
select job
from empl;
```

3. Який з наступних запитів не поверне загальне число службовців з таблиці empl?

- а)

```
select count(e_id) from empl;
```
- б)

```
select count(e_id) as total from empl;
```
- в)

```
select count(distinct e_id) from empl;
```
- г)

```
select count(e_id) from empl group by e_id;
```

4. Не допускається використати псевдоніми

- а) для стовпців;
- б) для таблиць;
- в) у вираженні WHERE;
- г) у вираженні SELECT.

5. Якщо необхідно за допомогою запиту повернути рядка 15-20, коректним вираженням LIMIT буде

- а)

```
LIMIT 15, 20
```
- б)

```
LIMIT 14, 19
```
- в)

```
LIMIT 14, 5
```
- г)

```
LIMIT 15, 5
```

Лекція № 12

з навчальної дисципліни Організація баз даних

Тема Складно-складові запити. Багатотабличні запити. Природні, внутрішні й перехресні об'єднання. Прямі, ліві та праві об'єднання.

Мета Охарактеризувати використання об'єднань у запитах до декількох таблиць, включаючи:

- 1 Природні, внутрішні й перехресні об'єднання.
- 2 Прямі об'єднання.
- 3 Ліві й праві об'єднання.

Час – 2 години

План проведення лекції

1 Складно-складові запити

1.1 Використання об'єднань для запитів до декількох таблиць

1.2 Об'єднання декількох таблиць

1.3 Самооб'єднання таблиць

2 Типи об'єднань

2.1 Основне об'єднання

2.2 Ліві й праві об'єднання

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BVH, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посіб-ник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Складно-складові запити

Лекція присвячена створенню більше складних запитів. Ми з'ясуємо, яким образом можна одночасно запросити дані з декількох таблиць, але для цього буде потрібно ознайомитися з поняттям об'єднання таблиць. Ми розглянемо наступні питання.

- 1 Використання об'єднань у запитах до декількох таблиць, включаючи.
- 2 Природні, внутрішні й перехресні об'єднання.
- 3 Прямі об'єднання.
- 4 Ліві й праві об'єднання.

1.1 Використання об'єднань для запитів до декількох таблиць

Запити, розглянуті на попередніх лекціях, повертали дані тільки з однієї таблиці. В умовах нормалізованої бази даних, коли інформація зберігається не в одній, а в цілому наборі таблиць, можливість ви-бора даних тільки з однієї таблиці, звичайно ж, є занадто більшим обмеженням. Добре спроектована реляційна база даних стає привабливим об'єктом через існуючі у ній відносини, тобто через зв'язки між таблицями. При виборі інформації з декількох таблиць такі зв'язки називають об'єднаннями.

А тепер розглянемо запити, у яких поєднуються дві таблиці вже відомої вам бази даних `organszation`.

Таблиця `dep`, що містить інформацію про відділи організації:

```
mysql> select * from dep;
+-----+-----+
| d_id | d_name |
+-----+-----+
|    11 | Administrative |
|    12 | Projecting |
|    13 | Realization |
|    14 | Testing |
+-----+-----+
4 rows in set (0.37 sec)
```

Таблиця empl, що містить інформацію про співробітників організації:

```
mysql> select * from empl;
```

e_id	e_name	job	d_id
111	Drovko	Director	11
112	Ivanov	Sysadmin	11
113	Konov	Sysadmin	11
121	Kornienko	AdministratorDB	12
122	Koval	Anaiyst	12
123	Palkin	Anaiyst	12
124	Gromov	Programer	12
131	Petrov	Programer	13
132	Sidorov	Programer	13
133	Tronov	Programer	13
134	Todorov	Coder	13
135	Abramov	Coder	13
141	Stogov	Programer	14
142	Vetrov	Coder	14

```
14 rows in set (0.07 sec)
```

Розглянемо наступний запит:

```
select empl.e_name, dep.d_name  
from empl, dep  
where empl.d_id = dep.d_id;
```

Тут у виразі FROM замість однієї таблиці зазначені дві. У цьому випадку ми хотіли б витягти з бази дані імена службовців разом з назвами відділів, у яких вони працюють. Результат буде наступним:

```
mysql> select empl.e_name, dep.d_name
-> from empl, dep
-> where empl.d_id=dep.d_id;
```

e_name	d_name
Drovko	Administrative
Ivanov	Administrative
Konov	Administrative
Kornienko	Projecting
Koval	Projecting
Palkin	Projecting
Gromov	Projecting
Petrov	Realization
Sidorov	Realization
Tronov	Realization
Todorov	Realization
Abramov	Realization
Stogov	Testing
Vetrov	Testing

```
14 rows in set (0.11 sec)
```

Як вийшов такий результат? Спочатку ми вибрали стовпці, що належать двом різним таблицям. (Як бачите, ми застосували нотацію, що використовує крапку-роздільник, щоб розрізнити поле `e_name` з таблиці `empl` і поле `d_name` з таблиці `dep`). Для цього треба було включити обидві таблиці у виразі `FROM`.

Найцікавішим у цьому запиті є вираз `WHERE`. Якщо виконати подібний запит без виразу: `WHERE, select empl.e_name, dep.d_name from empl, dep;` ми одержимо наступний результат:

```
mysql> select empl.e_name, dep.d_name
-> from empl, dep;
```

e_name	d_name
Drovko	Administrative
Drovko	Projecting
Drovko	Realization
Drovko	Testing
Ivanov	Administrative
Ivanov	Projecting
Ivanov	Realization
Ivanov	Testing
Konov	Administrative
Konov	Projecting
Konov	Realization
Konov	Testing
Kornienko	Administrative
Kornienko	Projecting
Kornienko	Realization
Kornienko	Testing
Koval	Administrative
Koval	Projecting
Koval	Realization
Koval	Testing
Palkin	Administrative
Palkin	Projecting
Palkin	Realization
Palkin	Testing
Gromov	Administrative
Gromov	Projecting
Gromov	Realization
Gromov	Testing
Petrov	Administrative
Petrov	Projecting
Petrov	Realization
Petrov	Testing
Sidorov	Administrative
Sidorov	Projecting
Sidorov	Realization
Sidorov	Testing
Tronov	Administrative
Tronov	Projecting
Tronov	Realization
Tronov	Testing
Todorov	Administrative
Todorov	Projecting
Todorov	Realization
Todorov	Testing
Abramov	Administrative
Abramov	Projecting
Abramov	Realization
Abramov	Testing
Stogov	Administrative
Stogov	Projecting
Stogov	Realization
Stogov	Testing
Vetrov	Administrative
Vetrov	Projecting
Vetrov	Realization
Vetrov	Testing

```
56 rows in set (0.04 sec)
```

У першому запиті, що використовує вираз WHERE, відобразилися імена службовців з відповідними назвами відділів, а в другому - всі можливі комбінації імен службовців і назв відділів. Однак це не дає ніякої можливості з'ясувати, які рядки несуть правильну інформацію, а які - помилкову. Результируюча множина, що містить всі можливі комбінації даних із двох таблиць, називається декартовим добутком цих таблиць. Ясно, що вираз WHERE важливий з погляду одержання потрібного нам результату.

Набір умов, використовуваних при створенні зв'язку для об'єднання таблиць, називають *умовою об'єднання*. У цьому випадку ми використали умову `empl.d_id = dep.d_id`, що зв'яже таблиці по зовнішніх ключах нашої оригінальної схеми.

В наступному запиті ми отримуємо прізвища тих співробітників, які працюють у відділі Реалізації:

```
mysql> select empl.e_name, dep.d_name
-> from empl, dep
-> where dep.d_name='Realization' and dep.d_id=empl.d_id;
```

e_name	d_name
Petrov	Realization
Sidorov	Realization
Tronov	Realization
Todorov	Realization
Abramov	Realization

5 rows in set (0.00 sec)

Коли потрібно витягти інформацію відразу з декількох таблиць, для знаходження цієї інформації доводиться використати наявні між таблицями зв'язки. Іноді це означає необхідність знайти шлях від уже наявної інформації до інформації, потрібної вам.

1.2 Об'єднання декількох таблиць

Об'єднання декількох таблиць аналогічно об'єднанню двох таблиць. Розглянемо випадок, коли потрібно з'ясувати, *яким службовцем і з яких відділів було доручено працювати над проектом 'College'*. Як знайти цю інформацію? Ми знаємо назву

проекту, і, знайшовши його в таблиці проектів (projects), можна з'ясувати його кодовий номер (pr_id). Це можна використати для того, щоб знайти відповідні рядки в таблиці proj_emp і побачити, які службовці працювали над даним проектом - ми одержимо кодові номери службовців (e_id), а по таблиці службовців (empl) одержати прізвища службовців і з'ясувати номери відділів, у яких ці службовці працюють. Із цією інформацією ми можемо звернутися до таблиці відділів (dep) і знайти назву відповідного відділу!

Маючи план у вигляді зазначеного шляху через чотири таблиці, ми повинні створити запит, що відбиває нашу логіку. Такий запит може виглядати в такий спосіб:

```
select dep .d_name, empl.e_name
from projects , proj_emp, empl, dep
where projects .pr_name='College'
and projects .pr_id = proj_emp .pr_id
and proj_emp .e_id = empl. e_id
and empl. d_id = dep .d_id;
```

Нижче наведений результат виконання цього запиту.

```
mysql> select empl.e_name, dep.d_name
-> from projects, proj_emp, empl, dep
-> where projects.pr_theme='College'
-> and projects.pr_id=proj_emp.pr_id
-> and proj_emp.e_id=empl.e_id
-> and empl.d_id=dep.d_id;
+-----+-----+
| e_name   | d_name   |
+-----+-----+
| Koval    | Projecting |
| Kornienko | Projecting |
| Petrov   | Realization |
| Sidorov  | Realization |
| Tronov   | Realization |
+-----+-----+
5 rows in set (0.06 sec)
```

Вивчаючи представлений тут запит, ви можете помітити, що нам довелося вказати спочатку всі таблиці спланованого нами шляхи, а потім умови об'єднань, що зв'

язують ці таблиці одну з іншою. Як бачите, для зв'язування чотирьох таблиць нам знадобилися три умови об'єднання.

Це спостереження можна використати для перевірки того, що в запиті присутні всі умови, необхідні для об'єднання. Якщо потрібно об'єднати n таблиць, те, як правило, кожна умова буде зв'язувати пари таблиць, тому потрібно вказати $n-1$ умова об'єднання.

1.3 Самооб'єднання таблиць

Аналогічно об'єднанню таблиці з іншою таблицею, можна об'єднати таблицю саму із собою. Але навіщо це робити? Іноді вас можуть цікавити зв'язку між рядками однієї й тієї ж таблиці. *Припустимо, що ви хочете з'ясувати імена службовців відділу, у якому працює Копов.* Для цього необхідно знайти в таблиці `empl` номер відділу (`d_id`), у якому працює Копов, а потім подивитися в таблиці `empl`, хто ще працює в тім же відділі.

```
mysql> select e2.e_name
-> from empl as e1, empl as e2
-> where e1.e_name='Koval'
-> and e1.d_id=e2.d_id;

+-----+
| e_name |
+-----+
| Kornienko |
| Koval      |
| Palkin     |
| Gromov     |
+-----+
4 rows in set (0.02 sec)
```

Це можна зробити так:

```
select e2.e_name
from empl e1, empl e2
where e1.e_name = 'Konov'
and e1.d_id = e2.d_id;
```

У цьому запиті для таблиці empl ми визначили два різних псевдоніми. По суті ми повідомили системі MySQL, що хотіли б мати дві окремих таблиці, e1 й e2, які повинні містити ті самі дані. Після цього ми поєднуємо їх так само, як будь-які інші таблиці. Тепер спочатку знаходимо рядок 'Konov' у таблиці e1 (e1.e_name='Konov'), а потім у таблиці e2 шукаємо рядка з тим же значенням номера відділу, що й в Konov (e1.d_id = e2.d_id). Спочатку це може здаватися незвичним, але в роботі з декількома таблицями в запитах ідея саме об'єднання таблиць не повинна викликати серйозних труднощів.

Нижче наведені результати попереднього запиту.

```
mysql> select e2.e_name
-> from empl e1, empl e2
-> where e1.e_name='Konov'
-> and e1.d_id=e2.d_id;
+-----+
| e_name |
+-----+
| Konov  |
| Kornienko |
| Palkin  |
+-----+
3 rows in set (0.00 sec)
```

Дуже просто додати умову, що виключить Konov з результуючого списку:

```
select e2.name
from empl e1, empl e2
where e1.e_name = 'Konov'
and e1.d_id = e2.d_id
and e2.e_name != 'Konov';
```

2 Типи об'єднань

Існують різні типи об'єднань, варіанти яких можуть використатися в MySQL.

2.1 Основне об'єднання

У попередньому розділі ми згадали поняття декартового добутку. У даному контексті його іноді називають повним або перехресним об'єднанням, але, незалежно

від термінології, таке об'єднання повертає повний набір комбінацій. При додаванні до об'єднання деякого умовного вираження (типу `empl.d_id = dep.d_id`) ми перетворюємо його в щось, називане об'єднанням по еквівалентності, що обмежує число рядків, що повертають запитом. Дотепер ми використали набір таблиць, перерахованих у вираженні FROM і розділених комами. Це визначало, як було показано вище, перехресне об'єднання, що перетворювалося в об'єднання по еквівалентності за допомогою вираження WHERE. MySQL пропонує різні форми синтаксису для об'єднань такого типу.

Розглянемо наш оригінальний запит:

```
select empl.e_name, dep.d_name  
from empl, dep  
where empl.d_id = dep.d_id;
```

Замість коми у вираженні FROM можна використати ключове слово JOIN:

```
select empl.e_name, dep.d_name  
from empl join dep  
where empl.d_id = dep.d_id;
```

Замість JOIN точно так само можна використати CROSS JOIN (перехресне об'єднання) або INNER JOIN (внутрішнє об'єднання).

При використанні цього типу об'єднання MySQL знаходить таблиці, що зв'язуються, і намагається об'єднати їх оптимальним образом, не обов'язково в тім порядку, у якому їх перелічили ви. Іноді така оптимізація запиту виявляється не зовсім правильною. Якщо ви хочете скасувати оптимізацію й змусити MySQL зв'язати таблиці в зазначеному вами порядку, замініте слово JOIN вираженням STRAIGHT JOIN (безпосереднє об'єднання).

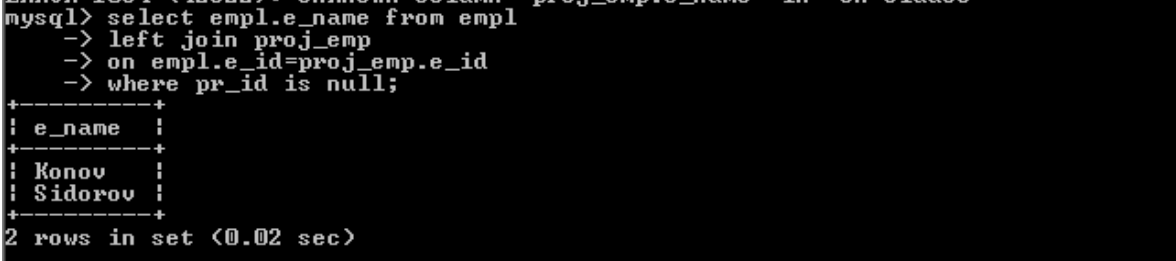
2.3 Ліві й праві об'єднання

При об'єднанні по еквівалентності в попередньому розділі ми використали JOIN, CROSS JOIN, INNER JOIN або, можливо, STRAIGHT JOIN і шукали рядки, що задовольняють критеріям пошуку для двох або декількох таблиць. Але що робити у випадках, коли потрібно знайти рядки однієї таблиці, які не мають відповідних рядків

в інший? Розглянемо, наприклад, ситуацію, коли потрібно з'ясувати імена службовців, що ще не одержували завдань по проектах, тобто службовців, коди яких (e_id) відсутні в таблиці (proj_emp). Вирішити таке завдання можна за допомогою LEFT JOIN, як показано нижче.

```
select empl.e_name
from empl left join proj_emp
on empl.e_id = proj_emp.e_id
where pr_id is null;
```

У результаті буде отримане наступне:



```
mysql> select empl.e_name from empl
-> left join proj_emp
-> on empl.e_id=proj_emp.e_id
-> where pr_id is null;
+-----+
| e_name |
+-----+
| Kovov  |
| Sidorov|
+-----+
2 rows in set (0.02 sec)
```

Легко підтвердити правильність отриманого результату за допомогою безпосередньої перевірки таблиць, але чому і як цей запит працює? Ліве об'єднання змушує взяти ліву таблицю в об'єднанні (у цьому випадку це таблиця empl) і спробувати знайти відповідності серед рядків правої таблиці (тут це таблиця proj_emp). Відповідні рядки розміщуються поруч із лівою таблицею. Для рядків лівої таблиці, що не мають відповідних рядків у правій таблиці, LEFT JOIN створює рядка, що складаються зі значень NULL. Рядка лівої таблиці, що не мають відповідних рядків у правій таблиці, можна знайти за допомогою пошуку по ключі NULL.

У даному прикладі ми використали LEFT JOIN, але точно так само можна використати RIGHT JOIN, що працює аналогічно, але використає в якості основний праву таблицю, а замість відсутніх рядків лівої таблиці створює рядка значень NULL.

Резюме

Об'єднання

1 Об'єднання - це зв'язування двох таблиць. Таблиці для об'єднання вказуються у вираженні FROM разом з типом об'єднання. Необхідно також указати вмови

об'єднання, що описують те, як повинні поєднуватися таблиці.

2 Поділяюча кома, JOIN, INNER JOIN й CROSS JOIN працюють однаково: вони поєднують таблиці так, щоб була можливість шукати в них дані.

3 Об'єднання типу LEFT й RIGHT дозволяють шукати рядка в таблиці, не всі рядки якої мають відповідні рядки в іншій.

Контрольні питання

1. Декартовий добуток

- а) представляє всі можливі комбінації рядків двох або декількох таблиць;
- б) представляє всі можливі комбінації співпадаючих рядків двох або декількох таблиць;
- в) представляє рядки однієї таблиці в парі з відповідними рядками іншої таблиці, а де це неможливо, замість рядка другої таблиці використовується рядок зі значень NULL;
- г) не описується жодним з попередніх варіантів.

2. Ліве об'єднання

- а) представляє всі можливі комбінації рядків двох або декількох таблиць;
- б) представляє всі можливі комбінації співпадаючих рядків двох або декількох таблиць;
- в) представляє рядки однієї таблиці в парі з відповідними рядками іншої таблиці, а де це неможливо, замість рядка другої таблиці використовується рядок зі значень NULL;
- г) не описується жодним з попередніх варіантів.

3. Об'єднання по еквівалентності

- а) представляє всі можливі комбінації рядків двох або декількох таблиць;
- б) представляє всі можливі комбінації співпадаючих рядків двох або декількох таблиць;

в) представляє рядки однієї таблиці в парі з відповідними рядками іншої таблиці, а де це неможливо, замість рядка другої таблиці використовується рядок зі значень NULL;

г) не описується жодним з попередніх варіантів.

4. Зв'язаний під запит називається так тому, що він:

а) зв'язує рядка таблиць;

б) зв'язує рядка в одній таблиці;

в) зв'язує два об'єднання;

г) зв'язує рядка зовнішнього запиту з рядками внутрішнього.

5. Різниця між наведеними нижче запитам 5.1 й 5.2 полягає в тім, що

а) ніякої різниці немає;

б) вони повертають різні дані;

в) вони повертають ті самі дані, але left JOINn (запит 5.1), швидше за все, буде виконуватися швидше;

г) вони повертають ті самі дані, але подзапрос (запит 5.2), швидше за все, буде виконуватися швидше.

Запит 5.1:

select employee.name

from employee left JOINn assignment

on employee.emp'loyeel = assignment.employeel

where clienti is null;

Запит 5.2:

select e.name, e.employeel from employee e where not exists

(select *

from assignment

where employeel = e.employeel);

Вправи

1. Створіть запит, що повертає ім'я службовця й всю інформацію про його (або неї) кваліфікації (skill).

2. Створіть запит, що використовує LEFT JOIN і повертає список клієнтів, для

яких не було виділено службовців, що працюють із ними.

3. Перепишіть свій запит із вправи 2, щоб тепер у ньому використалося ключове слово EXISTS.

Лекція № 13

з навчальної дисципліни Організація баз даних

Тема Вбудовані функції MySQL.

Мета Розглянути функції MySQL, які можна застосовувати в запитах, заснованих на використанні оператора SELECT.

Час – 2 години

План проведення лекції

- 1 Оператори
- 2 Керуючі функції
- 3 Функції рядків
- 4 Використання LIKE

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BVH, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посіб-ник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

ВСТУП

MySQL пропонує широкий вибір вбудованих операторів і функцій, які можуть виявитися корисними при створенні запитів. Більшість цих функцій призначено для використання у виразах SELECT й WHERE. Існують також деякі спеціальні функції угруповання для використання у вираженні GROUP BY. Ви вже бачили приклади використання основних операторів порівняння, а також функцій count () і max (). Число доступних для використання функцій дуже велико, тому ми розглянемо тільки найбільш корисні з них.

1 Оператори

В MySQL використовуються три головних типи операторів: знаки арифметичних операцій, оператори порівняння й логічних операторів.

Арифметичні операції

В MySQL використовуються звичайні арифметичні операції: додавання (+), вирахування (-), множення (*) і розподіл (/). Розподіл на нульове значення дає безпечний результат NULL.

Оператори порівняння

При роботі з операторами порівняння необхідно пам'ятати про те, що, за винятком декількох випадків, що виділяють особливо, порівняння чого-небудь зі значенням NULL дає в результаті NULL. Це стосується й порівняння значення NULL зі значенням NULL:

```
select NULL=NULL;
```

NULL = NULL
NULL

1 row in set (0.00 sec)

Порівняємо цей запит з наступним;

```
select NULL IS NULL;
```

```
I NULL IS NULL;
```

NULL IS NULL
1

1 row in set (0.00 sec)

Іншим моментом, про який також не слід забувати, є той факт, що в MySQL порівняння рядків у більшості випадків нечутливі до регістра символів. Якщо ви хочете, щоб рядки рівнялися з урахуванням регістра, укажіть перед кожною з них ключове слово BINARY. Наприклад, запит

```
select * from department where name='отдел маркетинга';
```

знайде слова 'отдел маркетинга' незалежно від регістра.

Якщо ж регістр символів важливий, варто вказати ключове слово binary:

```
select * from department where name = binary 'отдел маркетинга';
```

Озброївшись всією цією інформацією, розглянемо оператори порівняння. Найбільше часто використовувані з них наведені в таблиці 1.

Таблиця 1. Оператори порівняння

Оператор	Значення
=	Рівність
!= або <>	Нерівність
<	Менше
<=	Менше або дорівнює
>	Більше
>=	Більше або дорівнює
n BETWEEN min AND max	Перевірка діапазону
n IN (множина)	Приналежність до множини. Як множина може використатися список літеральних значень або виразів або подзапит. Прикладом множини є (яблуко, апельсин, груша)
<=>	Порівняння з урахуванням NULL: повертає 1 (істина) при порівнянні двох значень NULL
n IS NULL	Перевірка на наявність NULL у n
ISNULL(n)	Перевірка на наявність NULL у n

Логічні оператори

MySQL підтримує всі звичайні логічні операції, які можна використати у виразах об'єднання. Логічні вирази в MySQL можуть приймати значення 1 (істина), 0 (хибність) або NULL. Крім того, MySQL інтерпретує будь-яке ненульове значення, відмінне від NULL, як значення "істина".

Деякі з таблиць істинності, що містять значення NULL, небагато відрізняються від того, що можна було б очікувати. Логічні оператори наведені в таблиці 2.

Таблиця 2. Логічні оператори.

Оператор	Приклад	Значення
AND або &&	n && m	Логічне «І» істина && істина = істина хибність && хибність = хибність Всі інші вирази оцінюються як NULL
OR або	n m	Логічне «АБО» істина істина = істина NULL хибність = NULL NULL NULL = NULL хибність хибність = хибність
NOT або !	NOT n	Логічне «НІ» ! істина = хибність ! хибність = істина ! NULL = NULL
XOR	n XOR m	Логічне «виключаюче АБО» істина XOR істина = хибність істина XOR хибність = істина хибність XOR істина = істина NULL XOR n = NULL n XOR NULL = NULL

2 Керуючі функції

Першою множиною функцій, які ми розглянемо, будуть керуючі функції. Самими корисними з них є IF й CASE. Вони працюють так само, як оператори if й switch або case (відповідно) у більшості інших мов програмування.

Функція IF має прототип

IF (e1, e2, e3)

Якщо вираз e1 є істиною, то IF повертає e2, інакше повертається e3. Наприклад, використовуючи базу даних employee, можна виконати наступний запит:

```
select name, if ( job='Програміст', "фанат", "не фанат") from employee;
```

Функція CASE має наступні два прототипи:

CASE значення

WHEN [значення-для-порівняння] THEN результат

[WHEN [значення-для-порівняння] THEN результат ...]

[ELSE результат]

END

або

CASE

WHEN [умова] THEN результат

[WHEN [умова] THEN результат ...]

[ELSE результат]

END

Цю функцію можна використати для одержання одного або декількох значень.

Наприклад, розглянемо наступний запит:

```
select workdate, case
when workdate < 2000-01-01 then "архів"
when workdate < 2004-01-01 then "старі"
else "поточні"
end
from assignment;
```

У цьому запиті оцінюється значення workdate для кожного завдання з таблиці assignment. Завдання минулого століття характеризуються як здані в архів, ті завдання, що мають дату до початку цього року, - як старі, а всі інші - як поточні.

3 Функції рядків

Строкові функції MySQL можна розділити на дві категорії: функції обробки й функції порівняння рядків. Останні виявляються більше корисними, чим перші.

Функції обробки рядків

Список найбільш важливих функцій обробки рядків показаний у таблиці 3. У керівництві ви знайдете значно більше великий список.

Таблиця 3. Функції обробки рядків символів.

Оператор	Призначення
concat (<i>s1, s2, ...</i>)	Конкатенація рядків <i>s1, s2,...</i>
conv(<i>n, исх_базис, новый_базис</i>)	Конвертування числа <i>n</i> із системи числення з основою <i>исх_базис</i> у систему з основою <i>новый_базис</i> . (Може здатися дивним, що ця функція перебуває серед строкових, але деякі основи позначаються буквами, наприклад hexadecimal.)
length(<i>s</i>)	Повертає довжину рядка в символах
load_file(<i>имя_файла</i>)	Повертає вміст файлу <i>имя_файла</i> у вигляді рядка
locate (<i>игла, стог, старт</i>)	Повертає початкову позицію рядка <i>игла</i> в рядку <i>стог</i> . Пошук починається з позиції <i>старт</i>
lower(<i>s</i>) та upper(<i>s</i>)	Конвертує рядок <i>s</i> у нижній або верхній регістр
quote(<i>s</i>)	Видозмінює рядок <i>s</i> так, щоб віна підходив для введення в базу даних, тобто містить рядок в одиночні лапки й вставляє зворотну косу рису в необхідних випадках
replace(<i>исходная, поиск, замена</i>)	Повертає рядок, отриманий на основі рядка <i>исходная</i> , шляхом заміни всіх наявних у нього подстрок <i>поиск</i> на рядок <i>замена</i>
soundex(<i>s</i>)	Повертає рядок, схожу по вимові на <i>s</i> . Наприклад, іноді легше порівнювати схожі по вимові імена, чим самі імена
substrbg(<i>s,позиция,число</i>)	Повертає <i>число</i> символів рядка <i>s</i> , починаючи з позиції <i>позиция</i>
trim(<i>s</i>)	Видаляє початкові й кінцеві пробіли з рядка <i>s</i> . (Можна також використати ltrim (<i>s</i>) для видалення пробілів тільки на початку рядка й rtrim(<i>s</i>) для видалення їх тільки наприкінці.)

Функції порівняння рядків

На додаток до оператора рівності для порівняння рядків MySQL пропонує й інші функції порівняння.

- LIKE. Виконує порівняння з використанням групових символів.
- RLIKE. Виконує порівняння регулярних виразів.
- STRCMP. Виконує порівняння рядків, аналогічне strcmp О в С.
- MATCH. Виконує повнотекстовий пошук.

4 Використання LIKE

Використання LIKE для порівняння із груповими символами

Розглянемо приклад використання LIKE:

```
select *\nfrom department\nwhere name like '%проект%';
```

Замість того щоб шукати імена 'проект', ми тут шукаємо імена, що містять 'проект'. У результаті ми одержимо наступне:

departmentID	name
128	Отдел проектирования

1 row in set (0.04 sec)

Функція LIKE підтримує два види групових символів. Знак відсотка (%), як й у попередньому прикладі, відповідає будь-якому числу символів (включаючи нульове). Таким чином, вираження '%проект%' відповідає будь-якому рядку, що містить слово проект. Знову помітимо, що порівняння рядків, загалом кажучи, невідчутно до регістра символів, як уже згадувалося вище.

Другим груповим символом є знак підкреслення (_), що відповідає будь-якому одному символу. Наприклад, '_at' буде відповідати рядкам 'cat', 'mat', 'bat' і т.д.

Використання RLIKE для порівняння регулярних виразів.

Функцію RLIKE можна використати для порівняння на основі регулярних виражень.

Регулярний вираз - це шаблон, що описує загальну форму рядка. Існує спеціальна нотація для опису особливостей, які ми хотіли б бачити у відповідних рядків. Короткий опис такої нотації приводиться нижче/

Такому рядку відповідає будь-який строковий літерал (тобто строкова константа). Наприклад, за допомогою шаблону 'cat' буде знайдений рядок 'cat'. Але з його допомогою будуть також знайдені 'catacomb' й 'the cat sat on the mat'. Шаблону ' cat ' буде відповідати ' cat ' у будь-якому місці рядку, що перевіряється.

Якщо потрібно знайти тільки слово 'cat', шаблоном повинне бути l"cat\$1. Знак вставки (\$) позначає "показчик початку рядка", тобто в цьому випадку рядок повинна

починатися зі слова 'cat'. Знак долара (\$) позначає "покажчик кінця рядка", тобто в цьому випадку рядок повинна закінчуватися словом ' cat '. Тому шаблону ' "cat\$' буде відповідати тільки слово 'cat' і ніщо інше.

Як і вираження LIKE, регулярні вираження теж можуть містити групові символи, але в цьому випадку припустимий груповий символ іншої й тільки один - це крапка (.), що означає відповідність будь-якому одному символу. Так що тепер '.at' буде відповідати рядкам 'cat1', 'mat1', 'bat' і т.д.

Досить одного групового символу, оскільки є можливість указати, скільки разів символи (включаючи й групові) можуть з'явитися в рядку.

Наприклад, спеціальний знак * після символу означає, що символ може з'явитися або нуль, або більше раз. Так, 'п*' виявить ' ', 'п', 'пп', 'лпп' і т.д. Можна містити символи в дужки, і '(cat) *' виявить ' ', 'cat', 'catcat', 'catcatcat' і т.д. Можна також використати груповий символ, так що '.*' відповідає будь-якому числу будь-яких символів, тобто практично чому завгодно.

Точно так само знак "плюс" (+) означає, що символ або набір символів, перед яким він стоїть, повинен повторюватися один або більше раз, а знак питання (?) означає присутність нуль або один раз. Можна також указати конкретний діапазон для числа повторень, наприклад, '(cat) (2/4)' відповідає 'catcat', 'catcatcat1' й 'catcatcatcat'.

Нарешті, є цілий ряд класів символів, що представляють заздалегідь певні безлічі. Наприклад, '[: alnum :]' відповідає будь-якому буквено-цифровому символу.

Якщо ви використаєте мову програмування Perl, вам корисно знати, що MySQL використовує регулярні вираження POSIX, синтаксис яких відмінний від регулярних виражень в Perl.

Питання та завдання до контролю знань студентів

1 Який з наступних операторів не годиться для перевірки значення на рівність значенню NULL?

- а) ISNULL()
- б) <=>
- в) IS NULL

г) =

2 Виклик strcmp ('fred', 'Fred') поверне

а) -1

б) 0

в) 1

г) 2

3 Яку з наступних функцій варто використати для одержання назви місяця зі значення дати?

а) dayname ()

б) extract ()

в) subdate ()

г) now ()

4 Яку з наступних функцій використає MySQL для шифрування внутрішніх паролів своїх користувачів?

а) password()

б) encrypt ()

в) md5 ()

г) sha()

5 При використанні функцій, що групують, в операторі SELECT без виразу GROUP BY

а) буде отримане повідомлення про синтаксичну помилку;

б) вся таблиця буде розглядатися як єдина група;

в) вся результуюча безліч буде розглядатися" як єдина група;

г) кожен рядок буде вважатися окремою групою.

6 Створіть запит, що повертає імена службовців (name) і інформацію про їхній кваліфікації (job), але у випадку, коли кваліфікацією є 'Програміст', замініте її на 'Програміст/Аналітик'.

7 Створіть запит, що додає в базу даних новий відділ з назвою "Відділ керування майном". Потім додайте інформацію про нового службовця по імені Фред Смит, що буде працювати в цьому відділі на посаді адміністратора БД. При додаванні номера відділу використайте last_insert_id().

8 Запропонуйте запит, що повертає саме останнє завдання з таблиці assignment. (Підказка: використайте функцію, що групує, max ().)

Лекція № 14

з навчальної дисципліни Організація баз даних

Тема Створення підзапитів.

Мета Пояснити зміст поняття підзапиту. Показати на прикладах правила створення підзапитів похідних таблиць, підзапитів з одним значенням, підзапитів в логічних вираженнях.

Час – 2 години

План проведення лекції

1 Створення підзапитів

1.1 Підзапити похідних таблиць

1.2 Підзапити з одним значенням

1.3 Підзапити в логічних вираженнях

1.4 Використання агрегатних функцій

2 Опції оператора SELECT. Доповнення

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BVH, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посіб-ник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Створення підзапитів

Підзапит - це запит усередині іншого запиту, тобто запит, у якому використовуються результати іншого запиту. Іноді підзапити називають вкладеними запитами. Підзапити є новою можливістю в MySQL 4.1. Цієї можливості користувачі чекали давно. Підзапити не додають нових функціональних можливостей, але з ними запити часто виявляються більше зручними для читання, чим зі складними наборами умов об'єднання.

В MySQL були додані два основних види підзапитів:

- підзапити похідних таблиць;
- підзапити-вирази.

Підзапити-вирази застосовуються у вираженні WHERE оператора SELECT, Вони бувають двох типів:

- підзапит, що повертають одне значення або рядок;
- підзапит, використовуваний в логічному вираженні.

Ми розглянемо приклади підзапитів обох зазначених типів.

1.1 Підзапити похідних таблиць

Підзапити похідних таблиць дають можливість указати запит у вираженні FROM іншого запиту. По суті це дозволяє створити тимчасову таблицю й додати її в запит.

Використаємо наступні таблиці бази даних organization:

Таблицю emp1 з інформацією про співробітників організації:

```
mysql> select * from empl;
```

e_id	e_name	job	d_id
111	Drovko	Director	11
112	Ivanov	Sysadmin	11
113	Konov	Sysadmin	11
121	Kornienko	AdministratorDB	12
122	Koval	Analiyst	12
123	Palkin	Analiyst	12
124	Gromov	Programer	12
131	Petrov	Programer	13
132	Sidorov	Programer	13
133	Tronov	Programer	13
134	Todorov	Coder	13
135	Abramov	Coder	13
141	Stogov	Programer	14
142	Vetrov	Coder	14

```
14 rows in set (0.00 sec)
```

Та таблицю proj_empl, що містить кодові номери співробітників, кодові номери проектів, в яких задіяні ці співробітники та доля участі співробітника в проекті:

```
mysql> select * from proj_empl;
```

pr_id	e_id	royalty_share
234	122	45.00
234	121	30.15
234	131	10.00
234	132	10.00
234	133	4.75
134	123	30.00
134	112	25.00
134	124	25.00
134	133	20.00
235	123	25.00
235	113	25.00
235	124	15.00
235	135	15.00
235	142	10.00
235	141	10.00
456	123	40.00
456	112	10.00
456	121	10.00
456	132	20.00
456	135	20.00
156	122	40.00
156	124	10.00
156	134	10.00
156	141	10.00
156	142	15.00
156	133	15.00

```
26 rows in set (0.07 sec)
```

```
mysql> _
```

Розглянемо наступний простий запит:

```
select empl_id, e_name from empl where job='Programer';
```

Повинне бути очевидним, що цей запит поверне імена й кодові номери всіх програмістів. Можна використати цей запит у рамках іншого, щоб одержати додаткову інформацію:

```
select programmer.e_name
  from (select e_id, e_name from empl where job='Programer')
  as programmer,
  proj_emp
 where programmer.e_id = proj_emp.e_id;
```

```
mysql> select e_id, e_name from empl
-> where job='Programer';
+-----+-----+
| e_id | e_name |
+-----+-----+
| 124  | Gromov |
| 131  | Petrov |
| 132  | Sidorov|
| 133  | Tronov |
| 141  | Stogov |
+-----+-----+
5 rows in set (0.38 sec)
```

У цьому випадку ми використаємо підзапит (select e_id, e_name from empl where job='Programer') для створення похідної таблиці, що містить тільки e_id й e_name, що ми призначаємо псевдонім programmer. Після цього до створеної таблиці можна звернутися із запитом, як до будь-якої іншої. У цьому випадку ми використаємо цю таблицю для того, щоб з'ясувати, хто із програмістів працював над виконанням проектів, і одержуємо наступний результат:

```
mysql> select distinct programmer.e_name
-> from (select e_id,e_name from empl where job='Programer')
-> as programmer, proj_emp
-> where programmer.e_id=proj_emp.e_id;
+-----+
| e_name |
+-----+
| Petrov |
| Sidorov|
| Tronov |
| Gromov |
| Stogov |
+-----+
5 rows in set (0.03 sec)
```

Розглянемо запит1, який дозволить отримати наступну інформацію: *В яких проєктах приймав участь співробітник Tronov та з якою долею участі:*

```
mysql> select proj_emp1.pr_id, proj_emp1.royalty_share
-> from emp1, proj_emp1
-> where e_name='Tronov'
-> and emp1.e_id=proj_emp1.e_id;
+-----+-----+
| pr_id | royalty_share |
+-----+-----+
| 234   | 4.75          |
| 134   | 20.00         |
| 156   | 15.00         |
+-----+-----+
3 rows in set (0.00 sec)
```

Розглянемо запит2, який дозволить отримати наступну інформацію: *В скількох проєктах приймав участь співробітник Tronov та яке середнє значення долі його участі в цих проєктах:*

```
mysql> select count(emp.pr_id), avg(emp.royalty_share)
-> from (select proj_emp1.pr_id, proj_emp1.royalty_share
->       from emp1, proj_emp1
->       where e_name='Tronov'
->       and emp1.e_id=proj_emp1.e_id) as emp;
+-----+-----+
| count(emp.pr_id) | avg(emp.royalty_share) |
+-----+-----+
| 3                | 13.250000              |
+-----+-----+
1 row in set (0.04 sec)

mysql> _
```

1.2 Підзапити з одним значенням

Як й у попередньому розділі, ми починаємо із простого запиту
`select max (royalty_share) from proj_emp;`

```
mysql> select max(royalty_share)
-> from proj_emp1;
+-----+
| max(royalty_share) |
+-----+
| 45.00              |
+-----+
1 row in set (0.06 sec)
```

Він повертає одне значення, що представляє собою максимальну процентну ставку, що працівник одержить за виконання завдання в проєктах. Тут використається функція MySQL, що ми не ще згадували, - функція `max ( )`, що повертає максимальне значення відповідного стовпця. Використання результатів, повернутих такими функціями, являє типовий приклад застосування підзапитів з одним значенням.

Можна піти далі й використати даний запит у рамках іншого.

Підзапити з одним значенням повертають одне значення стовпця й звичайно використовуються для порівняння.

Ми намагаємося знайти той, кого можна назвати "ломовим конем" компанії:
Хто з службовців одержав більше всіх грошей за виконання завдання в проєктах?

```
mysql> select empl.e_name
-> from empl, proj_empl
-> where empl.e_id=proj_empl.e_id
-> and proj_empl.royalty_share=(select max(royalty_share) from proj_empl);
+-----+
| e_name |
+-----+
| Koval  |
+-----+
1 row in set (0.00 sec)
```

Можна також створити підзапит, що повертає не значення, а рядок, але користь від такого підзапита досить обмежена, тому відповідний приклад ми не приводимо.

1.3 Підзапити в логічних виразах

Підзапити в логічних виразах використовуються для перевірки істинності результату деяких спеціальних функцій, що повертають значення типу `BOOLEAN`. Такими спеціальними функціями є `IN`, `EXISTS`, а також `ALL`, `ANY` й `SOME`.

Ключове слово `IN` використовується для перевірки приналежності до деякої множини значень. Розглянемо наступний запит:

```
select e_name from empl
where e_id not in (select e_id from proj_emp);
```

Цей запит дасть той же результат, що й запит, розглянутий вище в прикладі застосування `LEFT JOIN`. Він дозволяє знайти службовців, які не мали завдань при

виконанні проектів. Ключове слово IN дозволяє провести пошук у деякій безлічі значень. Тут ми одержимо те ж, що й від вищенаведеного запиту:

```
mysql> select e_name, job from empl
-> where e_id not in (select e_id from proj_empl);
+-----+-----+
| e_name | job      |
+-----+-----+
| Drovko | Director |
+-----+-----+
1 row in set (0.00 sec)
```

Досить цікавим є наступна можливість використання ключового слова IN для перевірки приналежності до певного набору значень:

```
select e_name from empl
where d_id not in (13, 14);
```

```
mysql> select e_name, d_id from empl
-> where d_id not in (13,14);
+-----+-----+
| e_name | d_id |
+-----+-----+
| Drovko | 11    |
| Ivanov | 11    |
| Konov  | 11    |
| Kornienko | 12   |
| Koval  | 12    |
| Palkin | 12    |
| Gromov | 12    |
+-----+-----+
7 rows in set (0.03 sec)
```

Ключове слово EXISTS працює лише небагато по-іншому в порівнянні зі словом IN. При використанні EXISTS ми насправді використаємо в підзапиті дані деякого зовнішнього запиту. Такий запит іноді називають зв'язаним підзапитом.

Розглянемо наступний приклад: потрібно одержати список службовців, які ніколи не працювали над проектами.

```
mysql> select empl.e_name, empl.job
-> from empl
-> where not exists (select * from proj_empl
->                    where empl.e_id=proj_empl.e_id);
```

e_name	job
Drovko	Director

```
1 row in set (0.00 sec)
```

У підзапиті ми шукаємо рядки таблиці proj_empl, у яких значення e_id збігається зі значенням empl.e_id. Значення e.e_id отримано від зовнішнього запиту. Насправді MySQL тут робить наступне. Для кожного рядка з таблиці empl перевіряються результати підзапиту, і у випадку відсутності відповідності (WHERE NOT EXISTS) інформація про службовця додається в результуючу безліч.

Деякі користувачі знаходять такий синтаксис більше простим для розуміння, але той же результат можна одержати й за допомогою LEFT JOIN, як це було зроблено вище. Використання LEFT JOIN, крім того, буде більше ефективним, і, таким чином, що відповідає запит буде виконаний швидше.

Ключові слова ALL, ANY й SOME використовуються для порівняння з набором значень, повернутих підзапитом.

Отримаємо спочатку долі участі програмістів в розробці проектів:

```
mysql> select proj_empl.royalty_share
-> from proj_empl, empl
-> where proj_empl.e_id=empl.e_id and empl.job='Programer';
```

royalty_share
10.00
10.00
4.75
25.00
20.00
15.00
10.00
20.00
10.00
10.00
15.00

```
11 rows in set (0.00 sec)
```

А тепер отримаємо список тих співробітників, які працювали над проектами більше всіх програмістів:

```
mysql> select empl.e_name, proj_empl.royalty_share
-> from empl, proj_empl
-> where empl.e_id=proj_empl.e_id and proj_empl.royalty_share > all
-> (select proj_empl.royalty_share
-> from empl, proj_empl
-> where empl.e_id=proj_empl.e_id and empl.job='Programer');
+-----+-----+
| e_name | royalty_share |
+-----+-----+
| Koval | 45.00 |
| Kornienko | 30.15 |
| Palkin | 30.00 |
| Palkin | 40.00 |
| Koval | 40.00 |
+-----+-----+
5 rows in set (0.00 sec)
```

1.4 Використання агрегатних функцій

Дійсний розділ знайомить із функціями агрегатів, або агрегатними функціями, або функціями, аргументами яких є групи рядків, називані агрегатами, і на виході яких завжди одне-єдине значення, що резюмує. В один агрегат ви, за своїм розсудом, можете включити довільну кількість рядків, що може складатися з:

- всіх рядків довільної таблиці;
- тільки рядків, заданих якоюсь пропозицією WHERE;
- тільки рядків, створених якоюсь пропозицією GROUP BY.

Незалежно від того, скільки саме рядків ви включите у свій агрегат, агрегатна функція видасть для нього тільки одне значення, наприклад статистичне (сума, мінімум або середнє).

Функція	Результат
MIN(expr)	Мінімальне значення в expr
MAX(expr)	Максимальне значення в expr
SUM(expr)	Сума всіх значень в expr
AVG(expr)	Середнє всіх значень в expr
COUNT(expr)	Число всіх значень, що не є значеннями null, в expr
COUNT(*)	Число рядків у довільній таблиці або довільному наборі рядків

Можна також створити підзапит, що повертає не значення, а рядок, але користь від такого підзапита досить обмежена, тому відповідний приклад ми не приводимо.

2 Опції оператора SELECT. Доповнення

Знайомлячи з оператором SELECT, ми розглянули скорочену форму загального синтаксису цього оператора. Тепер розглянемо повний синтаксис оператора й з'ясуємо, що нам уже відомо.

Відповідно до керівництва по MySQL, оператор SELECT має наступну форму:

```
SELECT [STRAIGHT JOIN]
[SQL_SMALL_RESULT]    [SQL_BIG_RESULT]    [SQL_BUFFER_RESULT]
[SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
[HIGH PRIORITY] [DISTINCT | DISTINCTROW | ALL]
Вираження_select, . . .
[ INTO {OUTFILE | DUMPFILE} 'ім'я_файлу' опції_експорту]
[FROM посилання на таблиці]
[WHERE визначення_where]
[GROUP BY {ціле_без_знака | ім'я_стовпця | формула}
[ASC | DESC], . . .]
[HAVING групові_умови]
[ORDER BY {ціле_без_знака | ім'я_стовпця | формула}
[ASC | DESC], . . .]
[LIMIT [зсув,] рядка | рядка OFFSET зсув]
[PROCEDURE ім'я_процедури(список_аргументів)]
[FOR UPDATE й LOCK IN SHARE MODE]]
```

Більшість із зазначених виражень нам уже знайома. А тепер коротенько розглянемо ті з них, які ми ще не обговорювали.

1 Вираження STRAIGHT JOIN на початку оператора може використатися для того, щоб змусити оптимізатора запиту поєднувати таблиці так, як указали ви. Це буде мати той же ефект, що й вказівка STRAIGHT JOIN у вираженні WHERE, як було зроблено в розглянутому вище прикладі. Це - розширення ANSI SQL.

2 Опції SQL_SMALL_RESULT, SQL_BIG_RESULT й SQL_BUFFER_RESULT призначені для налаштування параметрів оптимізації. SQL_SMALL_RESULT й SQL_BIG_RESULT повідомляють MySQL, що ви очікуєте

побачити в результуючій безлічі або всього кілька рядків, або велике їхнє число. `SQL_BUFFER_RESULT` говорить MySQL про те, що результат варто помістити в тимчасову таблицю. Цим способом можна скористатися тоді, коли обробка даних забирає значний час, щоб відправити результати клієнтові, не допускаючи блокування відповідної таблиці в цей час. Ці опції теж є розширеннями MySQL у порівнянні з ANSI SQL.

3 `SQL_CACHE` й `SQL_NOCACHE` повідомляють MySQL, чи треба кешувати результати. (Ще одне розширення ANSI SQL.)

4 `SQL_CALC_FOUND_ROWS` передбачається використати з вираженням `LIMIT`. Ця опція змушує MySQL з'ясувати, скільки рядків повинне повернутися у випадку відсутності вираження `LIMIT`. Це значення тоді можна з'ясувати за допомогою `select found_rows ()` (ще одне розширення ANSI SQL). Опція має своєю метою мінімум дублювання дій. У версіях без цієї можливості звичайним рішенням виявляється виконання спочатку запиту з функцією `COUNT (*)`, а потім - оператора `SELECT` з вираженням `LIMIT`.

5 `HIGH PRIORITY` повідомляє MySQL про те, що запит повинен мати зазначений пріоритет у порівнянні з усіма операторами `UPDATE`, що очікують доступу до використовуваних таблиць.

6 Ми вже говорили про ключове слово `DISTINCT`, а `DISTINCTROW` є його синонімом. `ALL` є антонімом (повертає всі повторення) і використовується за замовчуванням.

7 Команда `SELECT INTO OUTFILE` протилежна команді `LOAD DATA INFILE`. Ця команда поміщає результат, що повертає оператором `SELECT`, у заданий файл. Параметри опції_експорту аналогічні опціям вираження `LOAD DATA INFILE`.

8 Команда `PROCEDURE` дозволяє вказати процедуру, яку варто застосувати до результуючої безлічі перед тим, як воно буде відправлено клієнтові. Ця процедура повинна бути написана мовою C++ й як така не може бути розглянута в контексті цієї лекції, але підходящий приклад можна знайти в посібнику з MySQL.

9 Вираження `FOR UPDATE` й `LOCK IN SHARE MODE` можуть знадобитися тільки тоді, коли механізм зберігання використовує блокування на рівні сторінок або рядків - на практиці це означає роботу з InnoDB й BDB. Якщо вказати `FOR`

UPDATE, установлюється блокування з монополізацією, а якщо вказати LOCK IN SHARE MODE, установлюється блокування із забезпеченням спільного доступу

Контрольні питання

- 1 У яких випадках у підсумковому запиті не використовується GROUP BY?
- 2 У чому відмінність використання WHERE від HAVING?
- 3 Які елементи можна використати в списку SELECT підсумкового запиту?
- 4 У яких випадках варто використати оператори IN або NOT IN у підзапитах? Приведіть приклади.
- 5 У яких випадках варто використати оператори EXISTS або NOT EXISTS у підзапитах? Приведіть приклади.
- 6 Чим SOME й ANY відрізняється від ALL при використанні у вкладених запитах? Приведіть приклади.

Лекція № 15

з навчальної дисципліни Організація баз даних

Тема Обробка транзакцій: транзакції, невдачі і відновлення, управління паралелізмом.

Мета Дати визначення поняття транзакції. Сформулювати принципи та правила виконання та використання транзакцій.

Час – 2 години

План проведення лекції

1 Поняття транзакції

1.1 Виконання транзакцій

1.2 Правила використання транзакцій

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група ВНУ, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посібник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Поняття транзакції

Транзакція (Transaction) - це послідовність із однієї або декількох команд SQL, які виконуються у вигляді єдиного логічного цілого. СУБД розглядає транзакцію як неподільну групу й виконує або всі її команди, або ні однієї.

Транзакція розглядається як деяка неподільна дія над базою даних, осмислена з погляду користувача. У той же час це логічна одиниця роботи системи. Розглянемо кілька прикладів. Що може бути названо транзакцією? Ким визначається, яка послідовність операцій над базою даних становить транзакцію? Звичайно, однозначно саме розроблювач визначає, яка послідовність операцій становить єдине ціле, тобто транзакцію. Розроблювач додатків або збережених процедур визначає це виходячи зі змісту обробки даних, саме семантика сукупності операцій над базою даних, що моделює з погляду розроблювача деяку одну нерозривну роботу, і становить транзакцію. Допустимо, виділимо роботу з уведення даних про книги, що надійшли, нових книгах, яких не було раніше в бібліотеці. Тоді цю операцію можна розбити на два послідовні: спочатку уведення даних про книгу - це новий рядок у таблиці BOOKS, а потім уведення даних про всі екземпляри нової книги - це уведення набору нових рядків у таблицю EXEMPLAR у кількості, рівній кількості екземплярів, що надійшли, книги. Якщо ця послідовність робіт буде перервана, то наша база даних не буде відповідати реальному об'єкту, тому бажано виконувати її як єдину роботу над базою даних.

Наступний приклад, що пов'язаний із прийняттям замовлення у фірмі на виготовлення комп'ютера. Комп'ютер складається з комплектуючих, які відразу резервуються за даним замовленням у момент його формування. Тоді транзакцією буде вся послідовність операцій, що включає наступні операції:

- уведення нового замовлення з усіма реквізитами замовника;
- зміни стану для всіх обраних комплектуючих на складі на "зайнято" із прив'язкою їх до певного замовлення;
- підрахунок вартості замовлення з формуванням платіжного документа типу рахунку, що виставляє, до оплати;

— включення нового замовлення у виробництво.

З погляду працівника, це єдина послідовність операцій, якщо вона буде перервана, то база даних втратить свій цілісний стан.

Ще один приклад, що досить характерний для навчальних закладів. При тривалій хворобі викладача або при його звільненні перед адміністрацією кафедри встає завдання перерозподілу всього навантаження, що веде викладач, по інших викладачах кафедри. Видів навантаження може бути трохи: читання лекцій і проведення занять за поточним розкладом, керівництво кваліфікаційними роботами бакалаврів, керівництво дипломними проектами фахівців, керівництво магістерськими дисертаціями, індивідуальна науково-дослідна робота зі студентами. І для кожного виду навантаження необхідно знайти виконавців і призначити їм додаткове навантаження.

Існують різні моделі транзакцій, які можуть бути класифіковані на підставі різних властивостей, що включають структуру транзакції, паралельність усередині транзакції, тривалість і т.д.

У даний момент виділяють наступні типи транзакцій: плоскі або класичні транзакції, ланцюгові транзакції й вкладені транзакції.

Плоскі, або традиційні, транзакції, характеризуються чотирма класичними властивостями: атомарності, погодженості, ізольованості, довговічності (міцності) - ACI (Atomicity, Consistency, Isolation, Durability). Іноді традиційні транзакції називають ACID-транзакціями. Згадані вище властивості означають наступне:

Властивість атомарності (Atomicity) виражається в тім, що транзакція повинна бути виконана в цілому або не виконана зовсім.

Властивість погодженості (Consistency) гарантує, що в міру виконання транзакцій дані переходять із одного погодженого стану в інше - транзакція не руйнує взаємної погодженості даних.

Властивість ізольованості (Isolation) означає, що конкуруючі за доступ до бази дані транзакції фізично обробляються послідовно, ізольовано друг від друга, але для користувачів це виглядає так, начебто вони виконуються паралельно.

Властивість довговічності (Durability) трактується в такий спосіб: якщо транзакція завершена успішно, те ті зміни в даних, які були нею зроблені, не можуть бути загублені ні за яких умов (навіть у випадку наступних помилок).

Можливі два варіанти завершення транзакції. Якщо всі оператори виконані успішно й у процесі виконання транзакції не відбулося ніяких збоїв програмного або апаратного забезпечення, транзакція фіксується.

Фіксація транзакції - це дія, що забезпечує запис на диск змін у базі даних, які були зроблені в процесі виконання транзакції.

Доти поки транзакція не зафіксована, припустиме анулювання цих змін, відновлення бази даних у той стан, у якому вона була на момент початку транзакції. Фіксація транзакції означає, що всі результати виконання транзакції стають постійними. Вони стануть видимими іншим транзакціям тільки після того, як поточна транзакція буде зафіксована. До цього моменту всі дані, що зачіпають транзакцією, будуть "видні" користувачеві в стані на початок поточної транзакції.

Якщо в процесі виконання транзакції трапилося щось таке, що унеможлиблює її нормальне завершення, база даних повинна бути повернута у вихідний стан. Відкіт транзакції - це дія, що забезпечує анулювання всіх змін даних, які були зроблені операторами SQL у тілі поточної незавершеної транзакції.

Кожен оператор у транзакції виконує свою частину роботи, але для успішного завершення всієї роботи в цілому потрібне безумовне завершення всіх їхніх операторів. Групування операторів у транзакції повідомляє СУБД, що вся ця група повинна бути виконана як єдине ціле, причому таке виконання повинне підтримуватися автоматично.

У стандарті ANSI/ISO SQL визначені модель транзакцій і функції операторів COMMIT й ROLLBACK. Стандарт визначає, що транзакція починається з першого SQL-оператора, що ініціюється користувачем або втримується в програмі, що змінює поточний стан бази даних. Всі наступні SQL-оператори становлять тіло транзакції. Транзакція завершується одним із чотирьох можливих шляхів (див. рис. 1.1):

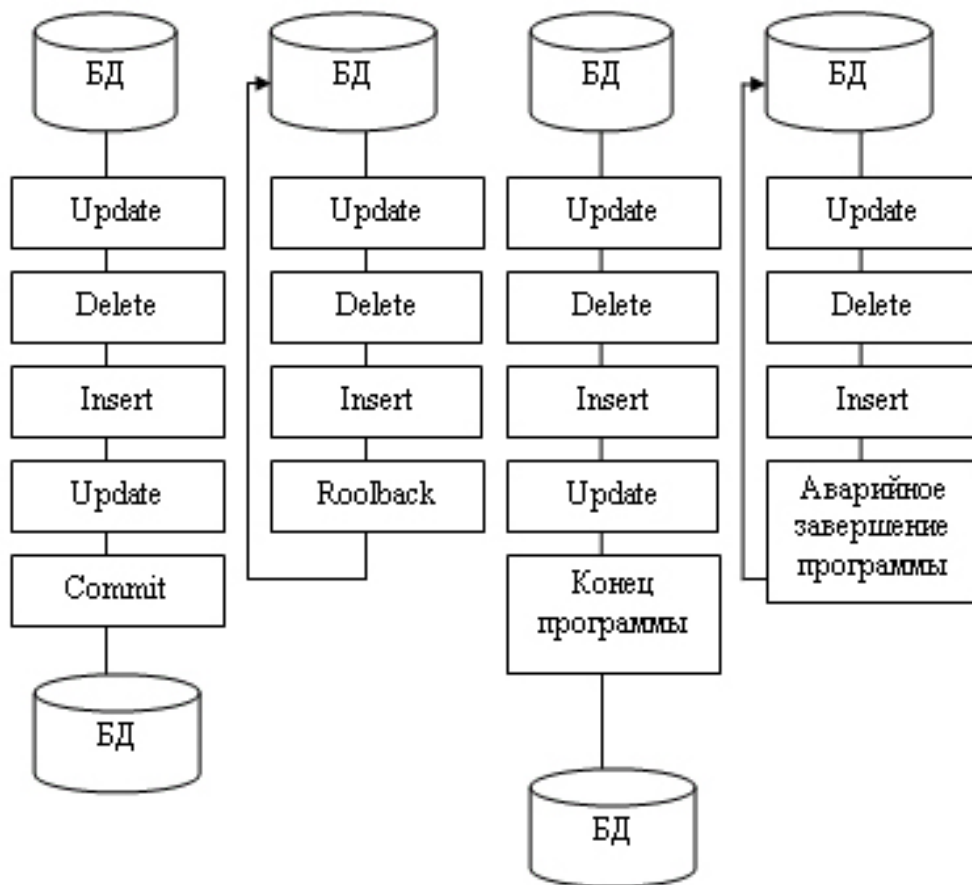


Рис. 1.1. Модель транзакцій ANSI/ISO

- оператор COMMIT означає успішне завершення транзакції; його використання робить постійними зміни, внесені в базу даних у рамках поточної транзакції;
- оператор ROLLBACK перериває транзакцію, скасовуючи зміни, зроблені в базі даних у рамках цієї транзакції; нова транзакція починається безпосередньо після використання ROLLBACK;
- успішне завершення програми, у якій була ініційована поточна транзакція, означає успішне завершення транзакції (начебто був використаний оператор COMMIT);
- помилкове завершення програми перериває транзакцію (начебто був використаний оператор ROLLBACK).

У цій моделі кожен оператор, що змінює стан БД, розглядається як транзакція, тому при успішному завершенні цього оператора БД переходить у новий стійкий стан.

У перших версіях комерційних СУБД була реалізована модель транзакцій ANSI/ISO. Надалі в СУБД SYBASE була реалізована розширена модель транзакцій,

що включає ще ряд додаткових операцій. У моделі SYBASE використовуються наступні чотири оператори:

Оператор `BEGIN TRANSACTION` повідомляє про початок транзакції. На відміну від моделі в стандарті ANSI/ISO, де початок транзакції неявно задається першим оператором модифікації даних, у моделі SYBASE початок транзакції задається явно за допомогою оператора початку транзакції.

Оператор `COMMIT TRANSACTION` повідомляє про успішне завершення транзакції. Він еквівалентний операторові `COMMIT` у моделі стандарту ANSI/ISO. Цей оператор, як й оператор `COMMIT`, фіксує всі зміни, які вироблялися в БД у процесі виконання транзакції.

Оператор `SAVE TRANSACTION` створює усередині транзакції крапку збереження, що відповідає проміжному стану БД, збереженому на момент виконання цього оператора. В операторі `SAVE TRANSACTION` може стояти ім'я крапки збереження. Тому в ході виконання транзакції може бути запам'ятовано кілька крапок збереження, що відповідають декільком проміжним станам.

Оператор `ROLLBACK` має дві модифікації. Якщо цей оператор використовується без додаткового параметра, то він інтерпретується як оператор відкату всієї транзакції, тобто в цьому випадку він еквівалентний операторові відкату `ROLLBACK` у моделі ANSI/ISO. Якщо ж оператор відкату має параметр і записаний у вигляді `ROLLBACK B`, то він інтерпретується як оператор часткового відкату транзакції в крапку збереження `B`.

Принципи виконання транзакцій у розширеній моделі транзакцій представлені на рисунку 1.2. На рисунку оператори позначені номерами, щоб нам зручніше було простежити хід виконання транзакції у всіх припустимих випадках.

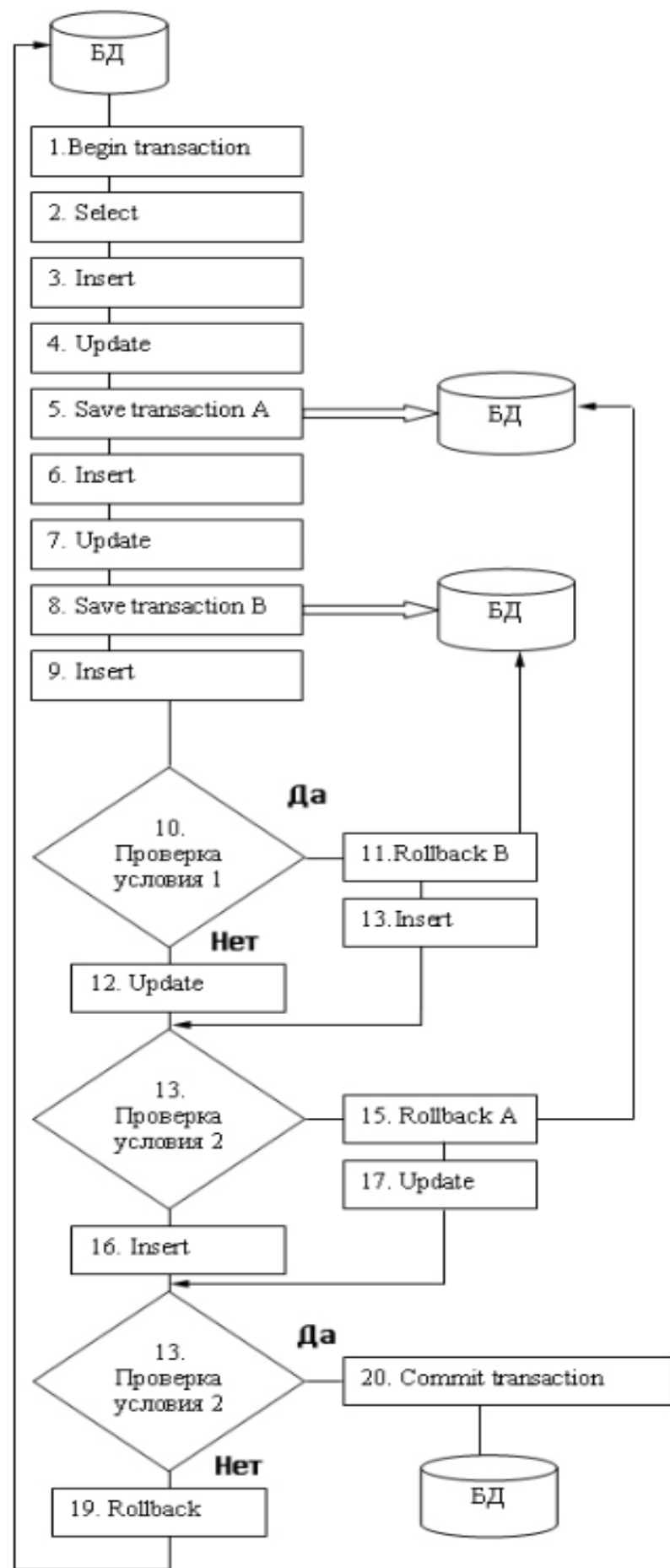


Рис. 1.2. Приклади виконання транзакцій у розширеній моделі

Транзакція починається явним оператором початку транзакції, що має в нашій схемі номер 1. Далі йде оператор 2, що є оператором пошуку й не міняє поточний стан БД, а наступні за ним оператори 3 й 4 переводять базу даних уже в новий стан. Оператор 5 зберігає цей новий проміжний стан БД і позначає його як проміжний стан у крапці А. Далі впливають оператори 6 й 7, які переводять базу даних у новий стан. А з 8 зберігає цей стан як проміжний стан у крапці В. Оператор 9 виконує введення нових даних, а оператор 10 проводить деяку перевірку умови 1; якщо умова 1 виконана, то виконується оператор 11, що проводить відкрит транзакції в проміжний стан В. Це означає, що наслідку дій оператора 9 як би стираються й база даних знову повертається в проміжний стан В, хоча після виконання оператора 9 вона вже перебувала в новому стані. И після відкоту транзакції замість оператора 9, що виконувався раніше зі стану В БД, виконується оператор 13 введення нових даних, і далі керування передається операторові 14. Оператор 14 знову перевіряє умову, але вже деяку нову умову 2; якщо умова виконана, то керування передається операторові 15, що виконує відкрит транзакції в проміжний стан А, тобто всі оператори, які змінювали БД, починаючи з 6 і закінчуючи 13, вважаються невиконаними, тобто результати їхнього виконання зникли й ми знову перебуваємо в стані А, як після виконання оператора 4. Далі з передається операторові 17, що обновляє вміст БД, після цього керування передається операторові 18, що пов'язаний з перевіркою умови 3. Перевірка з або передачею керування операторові 20, що фіксує транзакцію, і БД переходить у новий стійкий стан, і змінити його в рамках поточної транзакції неможливо. Або, якщо керування передане операторові 19, то транзакція відкочується до початку й БД повертається у свій початковий стан, а всі проміжні стани тут уже перевірені, і виконати операцію відкоту в ці проміжні стани після виконання оператора 19 неможливо.

Звичайно, розширена модель транзакції, запропонована фірмою SYBASE, підтримує набагато більше гнучкий механізм виконання транзакцій. Крапки збереження дозволяють установлювати маркери усередині транзакції таким чином, щоб була можливість скасування тільки частини роботи, проробленої в транзакції. Доцільно використати крапки збереження в довгих і складних транзакціях, щоб забезпечити можливість скасування зміни для певних операторів. Однак це обумовлює додаткові ви-

трати ресурсів системи - оператор виконує роботу, а зміни потім скасовуються; звичайно вдосконалення в логіку обробки можуть виявитися більше оптимальним рішенням.

2 Приклад формування та виконання транзакції

Давайте проілюструємо важливість транзакцій на прикладі. Припустимо, що клієнт переводить 500 доларів США зі свого ощадного рахунку на свій чековий рахунок. Ця операція складається із двох окремих дій, які виконуються послідовно:

1 Зменшити ощадний рахунок на 500 доларів США.

2 Збільшити чековий рахунок на 500 доларів США.

Для даної транзакції потрібно виконати дві команди SQL.

```
UPDATE saving_accounts
```

```
SET balance = balance - 500.00
```

```
WHERE account_number = 1009;
```

```
UPDATE checking_accounts
```

```
SET balance = balance + 500.00
```

```
WHERE account_number = 6482;
```

Представимо тепер, що СУБД відмовить (через збій оживлення, порушення в роботі системи, несправності апаратних засобів) після виконання першої команди, але до того, як буде виконана друга команда. Баланс на рахунках порушиться, а ви не будете об цьому знати. Це може привести до дуже серйозних і неприємних наслідків. Щоб уникнути цього, за допомогою транзакції варто гарантувати виконання двох команд SQL, що дозволить зберегти правильний баланс на рахунках. Якщо щось заважає виконанню однієї з команд транзакції, СУБД скасовує всі команди транзакції, виконані до цього. При відсутності помилки всі команди будуть виконані.

Щоб вивчити принципи виконання транзакцій, варто мати подання про деякі поняття:

- *виконання (Commit)*. Виконання транзакції реалізує в базі даних всі зміни, що відбулися з початку виконання транзакції. Після виконання операції всі зміни будуть видні іншим користувачам і залишаться в базі даних навіть при збої системи;

- *відкат (Roll back)*. Відкат транзакції скасовує всі зміни, які виникли в результаті виконання її команд. Після відкоту транзакції дані зберезуть свій початковий вид, начебто вона ніколи не була виконана;

- *журнал транзакцій*. Файл журналу транзакцій (або просто журнал) - це запис всіх змін, які були внесені в базу даних за допомогою транзакції. У журнал транзакцій записуються початок кожної транзакції, зміни, внесені в дані, а також інформація, за допомогою якої можна скасувати або повторити зміни (якщо це знадобиться згодом). При виконанні транзакцій у базі даних журнал постійно збільшується.

Хоча СУБД підтримує фізичну цілісність кожної транзакції, ви повинні починати й завершувати транзакції таким чином, щоб зберігати логічну цілісність даних, відповідно до вимог розв'язуваного завдання. Операція повинна складатися тільки з тих команд SQL, які необхідні для виконання однієї зміни (не більше й не менше).

Перед транзакцією й після її дані у всіх зв'язаних таблицях повинні бути в робочому стані.

При розробці й виконанні транзакцій необхідно враховувати наступні фактори:

- команди SQL, що працюють у транзакціях, змінюють дані, тому вам доведеться одержати дозвіл адміністратора бази даних, щоб запустити їх;

- обробка транзакцій ставиться тільки до команд INSERT, UPDATE й DELETE. Ви не зможете скасувати результат виконання команд SELECT, CREATE, ALTER й DROP, якщо включите їх у транзакцію;

- кожна команда INSERT, UPDATE й DELETE повинна виконуватися як частина транзакції;

- виконана транзакція стає постійною. Це значить, що всі зміни зберезуться навіть у випадку збою системи;

- система відновлення даних СУБД залежить від транзакції. Коли ви включаєте СУБД після збою, вона перевіряє журнал транзакцій, щоб з'ясувати, чи були вони виконані. Якщо СУБД знайде незавершені транзакції, вона скасує їх на підставі журналу. Вам належить повторити всі скасовані транзакції (хоча деякі СУБД автоматично повторюють незавершені транзакції);

- функція збереження/відновлення резервних копій СУБД залежить від транзакцій. Функція резервного копіювання регулярно зберігає копії бази даних разом з журналами транзакцій на резервному диску. Припустимо, що при збої системи робочий диск був ушкоджений, тому дані й журнал транзакцій стали що не читають. Ви можете скористатися функцією відновлення, що завантажить останню резервну копію бази даних і потім виконає всі виконані транзакції в журналі з моменту збереження резервної копії до останньої транзакції перед збоєм. При цьому база даних відновлюється в тім виді, у якому вона була до збою (вам знов-таки прийде повторити всі незакінчені транзакції);
- по очевидних причинах вам належить зберігати базу даних і журнал транзакцій на окремих фізичних дисках.

Для виконання транзакції її границі (початкова й кінцева крапки) повинні бути чітко позначені. Ці границі дозволяють СУБД виконати всі команди у вигляді єдиної операції. У відповідності зі стандартом SQL транзакція завжди починається з першої команди SQL і закінчується командою COMMIT або ROLLBACK.

Ви можете (або повинні) починати операцію за допомогою певної команди. СУБД звичайно використовує ключове слово BEGIN, щоб точно позначити початок транзакції. BEGIN не є частиною стандарту SQL, тому його використання відрізняється для різних СУБД (звернетеся до документації по вашій СУБД).

Команди SELECT у лістингу 1 показують, що транзакції UPDATE виконуються СУБД, а потім скасовуються командою ROLLBACK. Результат виконання лістингу показаний на рисунку 2.1.

Лістинг 1 Команди UPDATE (як і команди INSERT й DELETE) у групі команд транзакції ніколи не є кінцевими.

```
SELECT SUM(pages), AVG(price) FROM titles;  
BEGIN;  
UPDATE titles SET pages = 0;  
UPDATE titles SET price = price * 2;  
SELECT SUM(pages), AVG(price) FROM titles;  
ROLLBACK;  
SELECT SUM(pages), AVG(price) FROM titles;
```

SUM(pages)	AVG(price)
-----	-----
5107	18.3975

SUM(pages)	AVG(price)
-----	-----
0	36.7750

SUM(pages)	AVG(price)
-----	-----
5107	18.3975

Рис. 2.1. Результат виконання транзакції

Лістинг 2 демонструє більше практичне застосування транзакцій. Ми бажаємо видалити видавництво P04 з таблиці publishers без порушення посилальної цілісності. Оскільки деякі значення вторинного ключа в titles указують на видавництво P04, спочатку нам потрібно видалити зв'язані рядки з таблиць titles, titles_authors й royalties. Ми використаємо транзакцію, щоб переконатися, що всі команди DELETE будуть виконані. Якщо не всі команди були виконані успішно, то дані залишаться незміненими

Використайте транзакцію, щоб видалити видавництво P04 з таблиці publishers, а також пов'язані з нею рядка в інших таблицях. Спочатку зі зв'язаних, а потім з основної.

Лістинг 2

```
BEGIN;
DELETE FROM title_authors
WHERE title_id IN
(SELECT title_id FROM titles WHERE pub_id = 'P04');
DELETE FROM royalties
```

```
WHERE title_id IN  
(SELECT title_id FROM titles  
WHERE pub_id = 'P04');  
DELETE FROM titles  
WHERE pub_id = 'P04';  
DELETE FROM publishers  
WHERE pub_id = 'P04';  
COMMIT;
```

Завжди завершуйте транзакцію за допомогою команди COMMIT або ROLLBACK. Відсутність кінцевої команди може привести до виникнення більших транзакцій з непередбаченими результатами, до переривання виконання програми або до відновлення останньої незавершеної транзакції.

Ви можете розташовувати транзакції на декількох рівнях. Максимальна кількість рівнів залежить від конкретної СУБД. Більшість СУБД за замовчуванням працюють у режимі автоматичного виконання, якщо тільки ви не ввели ключове слово для транзакції. У режимі автоматичного виконання кожна команда виконується як окрема транзакція. Якщо команда була завершена успішно, СУБД виконує її; якщо СУБД виявляє помилку, команда буде скасована.

Працюючи з більшими транзакціями, ви можете задати проміжні маркери (крапки збереження), щоб розділити транзакцію на кілька частин. Крапки збереження дозволяють скасовувати зміни від поточної крапки в транзакції до попередньої крапки (за умови, що транзакція поки не була виконана). Уявіть собі сесію, у якій ви ввели кілька невиконаних команд INSERT, UPDATE й DELETE, а потім зрозуміли, що останні зміни невірні або не потрібні. За допомогою крапок збереження ви можете уникнути відновлення всіх команд. Microsoft SQL Server й Oracle підтримують крапки збереження.

Питання та завдання до контролю знань студентів

1. При виключеному режимі autocommit транзакції будуть фіксуватися
 - а) при виклику COMMIT;
 - б) при запиті блокування;
 - в) при виконанні умов пп. а) і б);
 - г) при жодному з вищевказаних умов.
2. Атомарність означає, що
 - а) або виконується весь зміст транзакції, або не виконується нічого;
 - б) операції перетворюють базу даних з одного погодженого стану в інше;
 - в) транзакції не впливають одна на іншу;
 - г) результати зафіксованої транзакції повинні бути перманентними.
3. Ізольованість означає, що
 - а) або виконується весь зміст транзакції, або не виконується нічого;
 - б) операції перетворюють базу даних з одного погодженого стану в інше;
 - в) транзакції не впливають одна на іншу;
 - г) результати зафіксованої транзакції повинні бути перманентними.
4. Стійкість означає, що
 - а) або виконується весь зміст транзакції, або не виконується нічого;
 - б) операції перетворюють базу даних з одного погодженого стану в інше;
 - в) транзакції не впливають одна на іншу;
 - г) результати зафіксованої транзакції повинні бути перманентними.
5. У режимі повторюваного читання
 - а) можливо "брудне" читання;
 - б) можливо неповторювальне читання;
 - в) можливо фантомне читання;

Лекція № 16

з навчальної дисципліни Організація баз даних

Тема Тригери: створення, використання, видалення.

Мета Ознайомитись з поняттям тригера, створенням тригера, прикладами використання тригерів.

Час – 2 години

Навчальні питання:

- 1 Створення тригера
- 2 Приклади використання тригерів
- 3 Видалення тригера
- 4 Перегляд існуючих тригерів

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BVH, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Базы даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посібник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

Тригер - це спеціальна збережена процедура, що виконується автоматично при настанні одного з подій: INSERT, UPDATE або DELETE, тобто при додаванні, зміні або видаленні рядків у таблиці. Тригер завжди є прив'язаним до конкретної таблиці.

1 Створення тригера

Тригер створюється за допомогою оператора CREATE TRIGGER. Синтаксис оператора:

CREATE TRIGGER *ім'я тригера*

Коли Спрацьовує Подія

ON *ім'я таблиці* FOR EACH ROW *Оператори тіла тригера*

Подія:

INSERT - тригер прив'язаний до події вставки нових записів у таблицю.

UPDATE - тригер прив'язаний до події відновлення записів у таблиці.

DELETE - тригер прив'язаний до події видалення записів у таблиці.

Коли Спрацьовує - може мати два значення:

BEFORE - до події

AFTER - після події.

Оператори тіла тригера :

BEGIN ... END;

У тілі тригера допускаються всі специфічні для збережених процедур і функцій конструкції:

- Інші складені оператори BEGIN... END.

- Оператори керування потоком виконання (IF, CASE, WHILE, REPEAT, LEAVE, ITERATE).

- Оголошення локальних змінних за допомогою оператора DECLARE і задання їм значень за допомогою оператора SET.

- Іменовані умови й оброблювачі помилок.

Крім зазначених операторів у тілі тригера можна використати так називані контекстні змінні, тобто змінні із префіксом NEW й OLD, які дозволяють одержувати доступ, відповідно, до нового й старого значення змінної. Синтаксис:

NEW. *Ім'я змінної* й OLD. *Ім'я змінної*.

Наприклад: NEW.Total - дозволяє одержати доступ до нового значення змінної Total.

OLD.Total - дозволяє одержати доступ до старого значення змінної Total.

NEW не використовується для події DELETE. OLD не використовується для події INSERT.

Для створення тригера потрібна спеціальний привілей
TRIGGER.

Зауваження. У СУБД MySQL тригер не можна прив'язати до каскадного відновлення або видалення записів з таблиці типу InnoDB по зв'язку первинний ключ/зовнішній ключ.

2 Приклади використання тригерів

1. Створимо тригер, що реалізує каскадне видалення для таблиць типу MyiSAM. Після кожного видалення рядка в таблиці Otdel виконується автоматичне видалення відповідних рядків з таблиці Sotr. Код тригера має такий вигляд:

```
DELIMITER //  
CREATE TRIGGER TRDelOtdel AFTER DELETE On Otdel  
FOR EACH ROW BEGIN  
DELETE FROM Sotr WHERE NOtd=OLD.NOtd; END;  
//  
DELIMITER ;
```

2. Нехай задані таблиці "Заповнення салонів" й "Квитки".

Таблиця "Квитки" має ім'я билет і містить інформацію про продані квитки на різні авіарейси. Таблиця має наступну структуру:

Nb - номер квитка, первинний ключ.

Nr - номер рейса.

Data - дата вильоту.

FIO – ПІБ пасажирів.

Comfort - рівень комфортності, може приймати одне із двох значень: або B - бізнес-клас, або E - клас-економ.

Таблиця створюється за допомогою наступного оператора.

```
CREATE TABLE 'bilet' (  
'Nb' smallint(6) NOT NULL AUTO_INCREMENT,  
'Nr' smallint(6) NOT NULL,  
'Data' DATE NOT NULL,  
'comfort' enum('B','E') NOT NULL,  
'FIO' varchar(30) NOT NULL, PRIMARY KEY ('Nb')  
) ENGINE=InnoDB AUTO_INCREMENT=1  
DEFAULT CHARSET=utf8 ;
```

Таблиця "Заповнення салонів" містить інформацію про те, скільки квитків кожного виду комфортності вже продано на даний рейс на дану дату. Таблиця називається Salon і має наступну структуру:

Nr - номер рейса;

Data - дата вильоту;

KolB - містить кількість проданих квитків бізнес-класу на цей рейс.

KolE - містить кількість проданих квитків класу економ на цей рейс.

Таблиця має складений первинний ключ: Nr, Data.

Таблиця Reis створюється за допомогою наступного оператора:

```
CREATE TABLE 'Salon' (  
'Nr' smallint(6) NOT NULL,  
'Data' date NOT NULL,  
'kol' smallint(6) DEFAULT NULL,  
'Kol' smallint(6) DEFAULT NULL, PRIMARY KEY ('Nr','Data')  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Потрібно написати тригер на додавання записів до таблиці "Квиток". Щораз при покупці квитка пасажиром відбувається додавання нового запису до таблиці "Квиток", що супроводжується автоматичним виконанням наступні:

- Перевіряється, є чи вже інформація про цей рейс у таблиці "Заповнення салонів". Якщо ні, то додається новий запис у таблицю "Заповнення салонів".

- До значень полів KOLB й KOLE таблиці "Заповнення салонів" додається кількість куплених квитків, відповідно, бізнесу-класу й класу економ.

Код тригера має такий вигляд:

```
DELIMITER //
CREATE TRIGGER INSBilet BEFORE INSERT ON Bilet FOR EACH ROW
BEGIN
DECLARE nomR SMALLINT;
DECLARE dataR DATE;
DECLARE kolE1 SMALLINT;
DECLARE kolB1 SMALLINT;
SELECT Nr, Data INTO nom, data FROM Salon
WHERE Nr=NEW.Nr AND data=NEW.data; SET kolB1=0;
SET kolE1=0;
IF (NEW.comfort='B') THEN
    SET kolB1=1;
    ELSE
    SET kolE1=1;
END IF;
IF (dataR IS NULL) THEN
    INSERT INTO Salon (Nr, data, kolB, kolE)
    VALUES (NEW.Nr, NEW.data, kolB1, kolE1);
ELSE
    UPDATE Salon SET kolB=kolB+kolB1, kolE+kolE1
    WHERE Nr=nomR AND data=dataR;
END IF;
END;
//
DELIMITER;
```

Тригер спрацьовує автоматично при додаванні запису до таблиці Квиток:

```
> INSERT INTO Bilet (Nr, data, FI, comfort)
```

```
VALUES (3337, "2008.11.20", "Іванов І.І", "В");
```

Для перевірки роботи тригера варто переглянути вміст таблиці Salon.

3 Видалення тригера

Для видалення тригера призначений оператор DROP TRIGGER, що має наступний синтаксис:

```
DROP TRIGGER ІМ'Я ТАБЛИЦІ . Ім'я тригера
```

Оператор DROP TRIGGER видаляє існуючий тригер для даної таблиці.

4 Перегляд існуючих тригерів

Одержати список існуючих тригерів можна за допомогою оператора SHOW TRIGGERS, що має наступний синтаксис:

```
SHOW TRIGGERS [FROM ІМЯБАЗЫДАНЫХ ] [LIKE Шаблон ]
```

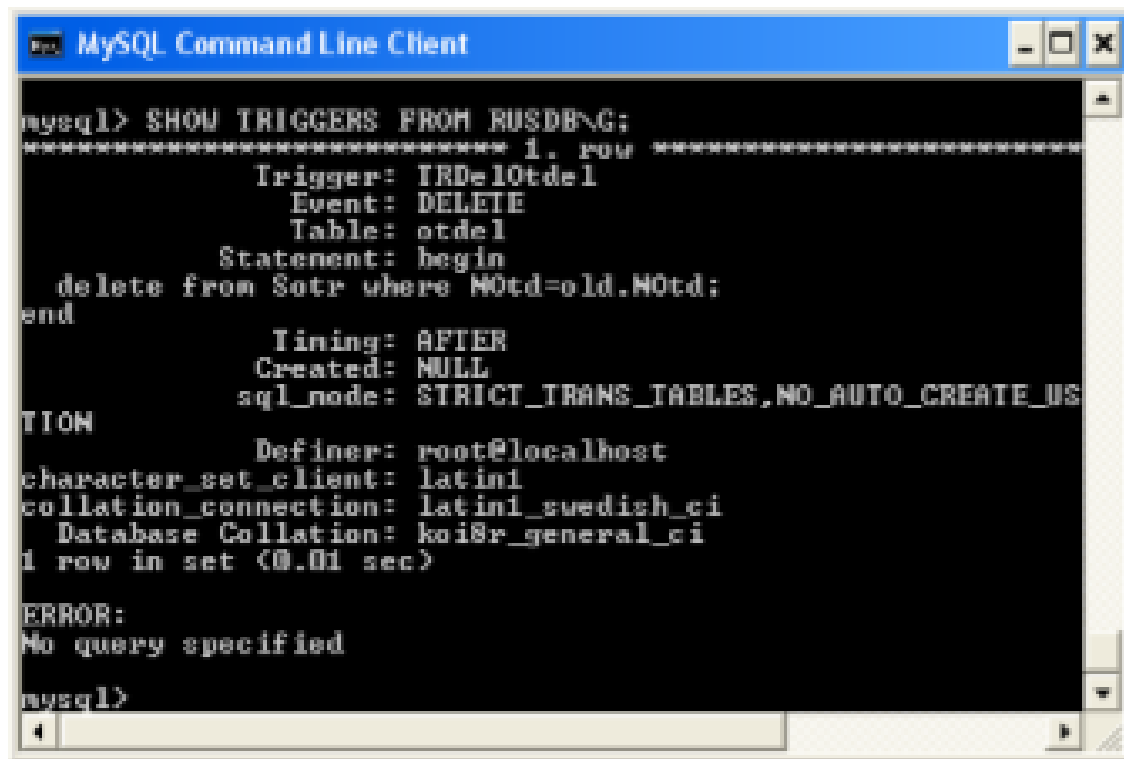
Оператор дозволяє витягти список і характеристики тригера із заданої бази даних. Якщо база даних має велику кількість тригерів, висновок можна обмежити за допомогою ключового слова LIKE, що має той же синтаксис, що й ключове слово LIKE в операторі SELECT. На малюнку 9.1 показане використання команди SHOW TRIGGERS для бази даних RUSDB. Поряд з ім'ям тригера, виводиться подія й таблиця, до якого прив'язаний тригер, а також тіло тригера:

```
BEGIN
```

```
DELETE FROM Sotr WHERE NOtd=OLD.NOtd;
```

```
END
```

На малюнку зазначений момент спрацьовування тригера: AFTER.



```
mysql> SHOW TRIGGERS FROM RUSDB\G;
+-----+-----+-----+-----+-----+-----+
Trigger: TRDel10tdel
Event: DELETE
Table: otel
Statement: begin
delete from Sotr where NOTd=old.NOTd;
end
Timing: AFTER
Created: NULL
sql_mode: STRICT_TRANS_TABLES,NO_AUTO_CREATE_US
TION
Definer: root@localhost
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: koi8r_general_ci
1 row in set (0.01 sec)

ERROR:
No query specified
mysql>
```

Питання та завдання для самоконтролю знань студентів

1. Що таке тригер?
2. Для чого використовуються контекстні змінні в тілі тригеру?
3. Чи можна в тілі тригеру використовувати локальні змінні?
4. Чи можна в тілі тригеру використовувати оператори розгалуження та циклічні оператори?
5. За допомогою яких команд можна створити тригер?
6. Як видалити тригер?
7. Як можна переглянути які тригери вже створені в базі даних?

Лекція № 17

з навчальної дисципліни Організація баз даних

Тема	Адміністрування бази даних засобами MySQL.
Мета	Огляд команд MySQL, що дозволяють отримати інформацію про сервер та бази даних
Час	– 2 години

План проведення лекції

- 1 Початок і припинення роботи сервера MySQL
- 2 Одержання інформації про сервер і бази даних
- 3 Установка значень змінних
- 4 Примусове завершення потоків
- 5 Очищення кешу
- 6 Вміст файлів журналу
- 7 Зведення опцій mysqladmin

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BVH, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посібник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Початок і припинення роботи сервера MySQL

Система конфігурується так, щоб сервер MySQL запускався автоматично. Однак іноді буває необхідно припинити роботу або запустити знову сервер вручну, наприклад, у випадку виникнення проблем.

Ми вже з'ясували, як запустити сервер MySQL. При цьому спосіб запуску залежить від операційної системи, заданих шляхів й інших параметрів установки. У середовищі Linux запустити сервер можна за допомогою команди

```
/etc/init.d/mysqld start
```

але тільки якщо виконувана копія `mysqld` розміщена по стандартній адресі в Red Hat. Якщо шлях інший, необхідно вказати саме його. Запустити сервер можна також за допомогою команди `safe_mysqld`

Зазначений сценарій намагається визначити правильні параметри запуску автоматично, а потім запускає MySQL з відповідними опціями. Знову ж, якщо шлях до сценарію не присутній у списку шляхів пошуку, потрібно вказати повний шлях, щоб система знайшла його.

Для завершення роботи сервера MySQL у середовищі Linux теж є пара можливостей. Можна використати

```
etc/init.d/mysqld stop
```

або,

```
mysqladmin -u root -p shutdown
```

Можна використати й інший обліковий запис адміністратора, але `root` виразно підійде. Облікові записи звичайних користувачів не повинні мати такого права.

У середовищі Windows, якщо `mysqld` був установлений у вигляді сервісу, найпростіше запустити його, відкривши Control Panel (Панель керування) і перейшовши до папки Administrative Tools (Засоби адміністрування), Services (Сервіси). Якщо вибрати сервіс MySQL, Windows запропонує вам варіанти Stop (Зупинка), Pause (Пауза) і Restart (Перезапуск).

2 Одержання інформації про сервер і бази даних

Програма `mysqlshow` і команда SQL `SHOW` дозволяють вам, як адміністраторові, одержати докладну інформацію про те, що відбувається з вашими базами даних і сервером.

Інформація про базу даних

Програма `mysqlshow` надає інформацію про бази даних. Якщо запустити її без параметрів, як

```
mysqlshow
```

вона видасть список баз даних, доступних вам, як користувачеві. Той же результат дасть команда

```
show databases;
```

якщо запустити її в рамках монітора `mysql` або іншого підходящого інтерфейсу користувача.

Як і для більшості інших сценаріїв командного рядка, можна додати до `mysqlshow` параметри `-u` з ім'ям користувача й `-p`, щоб указати відповідний пароль. Є й інші корисні опції. Повний список цих опцій можна одержати, виконавши команду

```
mysqlshow --help
```

Одна з опцій дозволяє вказати ім'я бази даних, щоб одержати більше докладну інформацію про цю базу даних. Наприклад, якщо вказати базу даних, як це зроблено нижче, ми одержимо список таблиць цієї бази даних:

```
mysqlshow -u ім'я користувача -p база даних
```

Можна одержати більше інформації про таблиці, якщо додати наприкінці рядка `--status`. Перевірте це самі для бази даних `employee`, як показано тут:

```
mysqlshow -u ім'я_користувача --status employee
```

Виведення в цьому випадку досить важко читати, оскільки виведені дані досить великі, але воно включає інформацію про механізм зберігання, про кожну таблицю, що використовується, про обсяг даних у кожній таблиці, про поточні значення всіх стовпців автоприросту й про набір символів, що використовується в кожній таблиці.

Щоб з'ясувати інформацію про базу даних і статусі сервера, у рамках клієнта MySQL можна також використати команду SQL `SHOW`.

Команди

```
show databases;
```

i

```
show tables;
```

дозволяють одержати інформацію про базу даних і про таблиці. Але оператор SHOW має величезну безліч інших опцій, які також можна використати. Можна використати команду

```
show columns from tablename;
```

щоб одержати інформацію, аналогічну тієї, котру надає оператор DESC. Точно так само можна використати

```
show table status
```

щоб одержати інформацію, що ми вже одержували від `mysqlshow --status`.

Статус сервера й значення змінних

Щоб одержати інформацію про сервер і параметри його роботи, можна з'ясувати статус сервера й значення його змінних.

Щоб побачити стан сервера MySQL, можна використати або команду

```
SHOW STATUS
```

у рамках MySQL, або

```
mysqladmin -u username -p -extended-status
```

з командного рядка.

Це дасть велику статистичну інформацію про те, що сервер робив з моменту його запуску. Корисними можуть виявитися значення з назвами `com_*` (наприклад, `com_select` повідомляє, скільки разів сервером виконувався оператор `select`).

Нижче наведені деякі інші значення, на які варто звернути увагу.

- `threads_connected`. Відповідає поточному числу з'єднань із сервером.
- `slow_queries`. Відповідає числу запитів, які були виконані сервером і зайняли більше часу, чим значення змінної сервера `long_query_time`. Такі запити реєструються в журналі повільних запитів.

- `uptime`. Відповідає часу в секундах, протягом якого виконується даний екземпляр сервера.

Щоб побачити значення змінних сервера, можете використати команду

show variables;
у рамках MySQL або
mysqladmin -u username -p variables
з командного рядка.

Значення більшості цих змінних можуть бути встановлені у файлі конфігурації, з командного рядка при запуску сервера або динамічно в рамках MySQL за допомогою команди SET.

Інформація про процеси

Можна з'ясувати, які процеси в даний момент виконуються вашим сервером, запустивши в рамках MySQL наступну команду:

```
show processlist;
```

У результаті ви, як мінімум, побачите інформацію об тільки що уведеному вами запиті (show processlist).

Ту ж інформацію можна одержати з командного рядка, використовуючи
mysqladmin -u ім'я_користувача -p showprocesslist

Інформація про надані привілеї

Можна з'ясувати, які привілеї надані конкретному користувачеві, якщо ввести
show grants for ім'я_користувача@хост;

Результат виражається в термінах оператора GRANT, який можна використати для одержання інформації про права, наданих користувачеві. Наприклад,

```
mysql> show grants for root@localhost;
```

у моїй системі повертає наступне:

Grants for root@localhost
GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION

Якщо вам необхідно згадати, які взагалі існують привілеї, введіть
show privileges;

У результаті ви одержите список привілеїв, доступних у вашій системі.

Довідкова інформація про таблиці

З'ясувати, які типи таблиць є й доступні в системі, можна за допомогою команди

`show table types;`

Можна побачити оператор `create`, за допомогою якого могла бути створена будь-яка конкретна таблиця в базі даних, якщо використати

`show create table ім'я_таблиці;`

Наприклад, для нашої бази даних `employee` ми можемо ввести

`show create table , department;`

й у результаті одержимо

```
CREATE TABLE 'department' (  
'departmentID' int(11) NOT NULL auto_increment,  
'name' varchar(30) default NULL,  
PRIMARY KEY ('departmentID')  
) TYPE=InnoDB CHARSET=latin1
```

(Зверніть увагу на те, що імена стовпців тут про всякий випадок укладені в лапки й зазначений використовуваний за замовчуванням набір символів, хоча ми його явно й не вказували.)

3 Установка значень змінних

Для установки значень змінних сервера застосовується оператор `SET`, а самі ці змінні можна побачити за допомогою оператора `show variables`, про що говорилося в попередньому розділі. Оператор `SET` використовує синтаксис

`set змінна=значення;`

Наприклад, можна ввести

`set sql_safe_updates=1;`

У результаті буде включений режим безпечного відновлення даних (з командного рядка цей режим включається за допомогою опції `--i-am-a-dummy`).

4 Примусове завершення потоків

Оператор `show processlist`, що обговорювався вище, дає змогу побачити, які потоки виконуються сервером. Ця команда також повідомляє `id` або унікальне ім'я кожного потоку. Якщо є проблемний потік (наприклад, запит, що виконується занадто довго, або проблемний користувач), можна примусово завершити виконання відповідного потоку за допомогою команди

```
kill id_процесу;
```

5 Очищення кешу

MySQL має безліч внутрішніх кешів. Виконати їхнє очищення можна за допомогою команд `FLUSH` й `RESET`. Наприклад, при відновленні привілеїв користувача шляхом зміни таблиць привілеїв вручну, щоб бути впевненими, що ці зміни сприйняті системою, варто виконати команду

```
flush privileges;
```

Іншим типовим прикладом використання `FLUSH` є очищення кеша запитів:

```
flush query cache;
```

У результаті кеш запитів буде дефрагментовано, а продуктивність системи збільшиться.

Використання оператора `RESET` аналогічно використанню оператора `FLUSH`. Наприклад, можна ввести

```
reset query cache;
```

й у результаті замість дефрагментації кеш запитів буде повністю очищений.

Повний список змінних, до яких застосовні оператори `FLUSH` й `RESET`, ви знайдете в посібнику з MySQL.

MySQL зберігає безліч файлів журналів (реєстрації подій), які можуть виявитися для вас корисними. По більшій частині ці файли за замовчуванням не створюються, тому, якщо ви хочете їх мати, необхідно активізувати запис файлів журналу. Включити запис кожного з файлів журналу можна за допомогою опцій командного рядка при запуску сервера або за допомогою команди `set`.

Нижче наведений список журналів, які можна створити.

Журнал реєстрації помилок. Відслідковує всі виникаючі помилки. Єдиний з журналів, запис у який за замовчуванням включена, розміщується у вашому каталозі даних. Файл називається `hostname.err` в Linux й `mysql.err` в Windows. Можна призначити йому будь-яке інше розміщення, використовуючи опцію `log-err=ім'я_файлу` у файлі конфігурації `my. ini` (або `my.cnf`).

Журнал реєстрації запитів. Реєструє всі запити, виконувані системою. Можна включити запис цього журналу й указати його розташування за допомогою опції `log=ім'я_файлу`.

Журнал двійкової реєстрації. Реєструє всі запити, що змінюють дані. Замінив журнал реєстрації відновлень, який все ще існує і буде існувати до версії MySQL 5.0. Можна включити запис цього журналу й указати його розташування за допомогою опції `log-bin=ім'я_файлу`.

Журнал реєстрації повільних запитів. Реєструє всі запити, час виконання яких виявився більше, ніж значення, збережене змінною `long_query_time`. Можна включити запис цього журналу й указати його розташування за допомогою опції `log-slow-queries=ім'я_файлу`.

Всі зазначені вище журнали, крім журналу двійкової реєстрації, є простими текстовими файлами. Журнал двійкової реєстрації Можна побачити за допомогою команди

```
mysqlbinlog logfile
```

Файли журналів реєстрації будуть згодом розростатися, тому бажано на регулярній основі міняти файли журналів. У середовищі Linux система MySQL пропонує сценарій `mysql-log-rotate`, що рятує від необхідності робити це вручну.

При використанні іншої операційної системи ви можете перемістити старі файли журналів в інше місце на диску вручну й дати вказівку MySQL почати використання нових файлів журналів за допомогою команди `mysqladmin flush-logs`

7 Зведення опцій `mysqladmin`

Програма `mysqladmin` має дуже багато опцій, користь і частота застосування яких досить різні.

Деякі завдання можуть бути виконані як за допомогою команд SQL, так і за допомогою `mysqladmin` - наприклад, створення й знищення баз даних:

```
mysqladmin create ім'яБД
```

```
mysqladmin drop ім'яБД
```

Типовим прикладом використання `mysqladmin` є одержання інформації про сервер і його поточний стан. Така інформація може бути як дуже простою ("чи доступний сервер?" - `ping`), так й істотно більше докладню, наприклад, що включає список всіх доступних змінних або процесів. Нижче приводиться кілька прикладів використання `mysqladmin`.

Щоб з'ясувати, чи включений сервер, використайте

```
mysqladmin ping
```

Щоб з'ясувати, яка версія програмного забезпечення сервера MySQL установлена на вашій машині, використайте

```
mysqladmin version
```

Щоб одержати коротке або довге повідомлення про статус сервера, використайте

```
mysqladmin status
```

```
mysqladmin extended-status
```

Щоб одержати список активних потоків у рамках даного сервера, використайте

```
mysqladmin processlist
```

Одержавши список процесів (потоків), ви можете примусово завершити виконання деяких з них по вашому розсуді, використовуючи команду

```
mysqladmin kill id1,id2,id3...
```

Щоб надрукувати значення змінних MySQL, використайте
`mysqladmin variables`

Питання та завдання до контролю знань студентів

- 1 Запис у які з журналів подій включена за замовчуванням?
 - а) у журнал реєстрації запитів;
 - б) у журнал реєстрації повільних запитів;
 - в) у журнал реєстрації помилок;
 - г) в усі зазначені вище журнали.
- 2 Команда SQL SHOW може використатися для одержання
 - а) списку доступних баз даних;
 - б) списку таблиць у базі даних;
 - в) списку стовпців у таблиці;
 - г) усього вищезгаданого.
- 3 Програма `mysqladmin` може використатися для
 - а) перезавантаження привілею, що дозволяє бути впевненим, що зміни прийняті системою;
 - б) перевірки поточного стану сервера;
 - в) запуску й припинення роботи сервера;
 - г) закриття й наступного відкриття файлів журналів;
 - д) усього вищевказаного.
- 4 Включите опції запису всіх чотирьох журналів. Після запуску декількох запитів розглянете їхній уміст. Якщо у вас немає доступу до досить великої бази даних, можуть виникнути труднощі з одержанням записів у журнал реєстрації повільних запитів. Мінімум для визначення часу повільного запиту дорівнює одній секунді.

Лекція № 18

з навчальної дисципліни Організація баз даних

Тема Рівні привілеїв. Створення облікових записів користувачів за допомогою GRANT й REVOKE.

Мета Розглянути створення облікових записів користувачів, різні доступні привілеї й те, як ці привілеї представлені в таблицях MySQL.

Час – 2 години

План проведення лекції

1 Створення облікових записів користувачів за допомогою GRANT й REVOKE.

2 Рівні привілеїв.

3 Таблиці привілеїв.

Література

1. Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BVH, 2006. – 384 с.: іл.
2. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. [Текст] – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
3. Трофименко О. Г. Організація баз даних :Навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. [Текст] – Одеса : Фенікс, 2019. – 246 с.
4. Харів Н. О. Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. [Текст] – Рівне : НУВГП, 2018. – 127 с.
5. Ярцев В.П. Організація баз даних та знань: навчальний посібник./ В.П. Ярцев [Текст] – К. ДУТ 2018.-214с.

Навчальні матеріали лекції

1 Створення облікових записів за допомогою GRANT й REVOKE

Користувальницькі привілеї встановлюються оператором GRANT і скасовуються оператором REVOKE. Це - стандартні оператори SQL, які можна виконувати точно так само, як будь-який інший оператор. Вся інформація про користувачів MySQL і користувальницьких правах в остаточному підсумку зберігається в базі даних MySQL, точно так само, як і ваші додатки.

Щоб запустити зазначені оператори, потрібно мати відповідний рівень привілеїв доступу. Якщо ви встановлювали MySQL самостійно, ви будете мати доступ до кореневого облікового запису й, таким чином, мати необхідні привілеї. Якщо ви використаєте MySQL на машині, контрольованій кимсь іншим (наприклад, на роботі або на машині постачальника послуг Internet), то можете не мати прав доступу, необхідних для виконання відповідних запитів. Якщо ви відповідних прав не маєте, то одержите повідомлення про помилку, подібне наступний:

```
ERROR 1045: Access denied for user: 'laura@127.0.0.1'
```

(Using password: YES)

Надання привілеїв

Почнемо з розгляду оператора GRANT, що використовується для створення облікових записів користувачів і надання їм прав доступу до баз даних, таблицям і функціям. Спочатку розглянемо наступний простий приклад:

```
grant usage  
on *  
to luke@localhost identified by 'пароль';
```

Цей оператор створює обліковий запис для користувача з ім'ям luke, коли він намагається увійти в систему з вузла localhost. Для нього встановлюється пароль (замість зазначеного тут пароля пароль, мабуть, варто використати що-небудь більше надійне!). Слово usage указує привілею, який ви наділяєте користувача luke. Воно означає, що з може зареєструватися (тобто увійти в систему) - і нічого іншого. Вира-

ження ON використовується для того, щоб указати, на які об'єкти поширюються надавані права. Через те, що тут ми надаємо тільки право з у системі, вираження ON у цьому випадку не дуже важливо.

Загальна форма оператора GRANT з посібника з MySQL виглядає в такий спосіб:

```
GRANT привілея [ (список_стовбців) ]  
[, привілея [ (список_стовбців) ] ...]  
ON { ім'я_таблиці | * | *.* | ім'я_БД.* }  
TO ім'я_користувача [ IDENTIFIED BY [PASSWORD] 'пароль']  
[, ім'я_користувача [IDENTIFIED BY 'пароль'] ...]  
[REQUIRE  
NONE |  
[{SSL | X509}]  
[CIPHER шифр [AND] ]  
[ISSUER видавець [ AND] ]  
[SUBJECT суб'єкт] ]  
[WITH [GRANT OPTION | MAX__QUERIES__PER_HOUR # |  
MAX_UPDATES_PER_HOUR # | MAX_CONNECTIONS_FER_HOUR # ] ]
```

Вираження GRANT містить список надаваних привілеїв. Одні привілеї є глобальними (тобто вони застосовуються до всіх баз даних), а інші ставляться тільки до певних об'єктів (базам даних, таблицям або стовпцям).

У вираженні ON вказуються об'єкти, на доступ до яких надаються права. Це може бути з таблиця або певна база даних з усіма її таблицями (ім'яБД . *). Можна також вказати * . *, що позначає всі бази дані й всі таблиці. Якщо вказати *, привілеї будуть надані на обрану в даний момент базу даних. Якщо ніякої бази даних не обрано, привілеї будуть надані так, начебто у вираженні зазначене * . *.

Вираження TO використовується для того, щоб вказати користувача, якому надаються привілеї. Якщо цей користувач уже має обліковий запис, нові привілеї будуть додані до неї. Якщо немає - обліковий запис користувача буде створена. У цьому вираженні можна вказати більше одного користувача. Можна також вказати імена хостів, з яких ці користувачі можуть зареєструватися, наприклад fred@localhost. Якщо у

вас виникають труднощі при реєстрації нового користувача, спробуйте додати в оператор GRANT ім'я хосту, з якого ви реєструєтесь. Ім'я користувача для входу в MySQL може не збігатися з ім'ям користувача в операційній системі. Імена користувачів з містити до 16 символів.

Вираження IDENTIFIED BY установлює пароль для нового користувача або переустановлює пароль, якщо користувач уже існує.

Користувачі можуть змінити свої паролі, виконавши команду

```
set password = password('новий_пароль');
```

Ви можете з пароль користувача за допомогою, наприклад,

```
set password for fred@localhost = password('новий_пароль');
```

Для цього вам потрібно мати право доступу до бази даних з ім'ям mysql.

Вираження WITH GRANT OPTION - це спеціальний привілей, що дозволяє користувачеві надавати привілею. Якщо ви виявили, що не можете надавати ніяких привілеїв користувачам, це значить, що саме ця привілея вам не надана. Точно так само ви не можете надавати іншим користувачам ті привілеї, який не володієте самі.

Вираження WITH з використатися для того, щоб обмежити число запитів, модифікацій або підключень, які користувачеві дозволяється виконувати за одна година. Значенням за замовчуванням для відповідних параметрів є 0, що означає відсутність обмежень.

Вираження REQUIRE дозволяє зажадати, щоб зазначений користувач при вході в систему використав захищене з'єднання. Для цього також необхідно правильно сконфігурувати MySQL.

2 Рівні привілеїв

Привілею, які надаються за допомогою оператора GRANT, можна поділити на дві основні категорії: привілею користувача й привілеї адміністратора.

Привілеї користувача

Привілею користувача наведені в таблиці 1.

Таблиця 1. Привілеї користувача

Привілея	Значення
CREATE	Дозволяється створювати таблиці
CREATE TEMPORARY TABLES	Дозволяється створювати тимчасові таблиці
DELETE	Дозволяється видаляти рядки
EXECUTE	Дозволяється виконувати процедури
INDEX	Дозволяється створювати індекси
INSERT	Дозволяється вставляти рядки
LOCK TABLES	Дозволяється блокувати таблиці
SELECT	Дозволяється вибирати рядка
SHOW DATABASES	Дозволяється виконувати команду SHOW DATABASES для одержання списку доступних баз даних
UPDATE	Дозволяється обновляти рядки
USAGE	Дозволяється тільки вхід у систему

Привілеї адміністратора

Привілеї, надавані тільки адміністраторові, показані в таблиці 2. Деякі з них можуть надаватися окремим користувачам на ваш розсуд і із застереженнями, але вуж ніяк не за замовчуванням.

Таблиця 2. Привілеї адміністратора

Привілея	Значення
ALL	Користувач має всі привілеї, крім WITH GRANT OPTION
ALTER	Користувач може змінювати таблиці. Можна дозволити це деяким з найбільш кваліфікованих користувачів, але дотримуйте обережності, оскільки це дає можливість змінювати таблиці привілеїв
DROP	Користувач може видаляти таблиці. Можна дозволити це користувачам, що заслуговують довіри
FILE	Користувач може завантажувати дані з файлу. Можна дозволити це користувачам, що заслуговують довіри. Бережіться користувачів, що намагаються завантажувати файли типу /etc/passwd або щось подібне!
PROCESS	Користувач може виводити повний список процесів, тобто бачити всі процеси, виконувані MySQL
RELOAD	Користувач може використати оператор FLUSH. Цей оператор може виконувати різні завдання.
REPLICATION CLIENT	Користувач може перевірити, де перебувають головні й підлеглі об'єкти
REPLICATION SLAVE	Спеціальний привілей, призначений для користувача реплікації на підлеглому пристрої.
SHUTDOWN	Користувач може викликати mysqladmin shutdown.
SUPER	Користувач може підключитися навіть тоді, коли MySQL має максимальне число з'єднань, і може виконувати команди CHANGE MASTER, KILL (процес), mysql admin debug, PURGE MASTER LOGS й SET GLOBAL

Привілея	Значення
WITH GRANT OPTION	Користувач може передавати іншим будь-які права, які він має

Є ще одна привілея - REFERENCES. Вона зарезервована для майбутнього використання, і хоча ви можете надати її в цей час, реально вона не надає ніяких прав.

Оцінка привілеїв

За допомогою оператора GRANT надаються наступні типи привілеїв.

– Глобальні привілеї застосовуються у всіх базах даних. Вони вказуються символами *. * в операторі GRANT. Наприклад:

```
grant all on *.* to fred;
```

– Привілеї рівня бази даних ставляться до однієї конкретної бази даних. Вони надаються за допомогою вказівки ім'яБД. * в операторі GRANT:

```
grant all on employee.* to fred;
```

– Привілеї рівня таблиці стосуються окремої таблиці. Вони надаються за допомогою вказівки імені конкретної таблиці в операторі GRANT:

```
grant select on department to fred;
```

– Привілеї рівня стовпця стосуються окремого стовпця. Вони вказуються у вираженні GRANT оператора GRANT. Наприклад:

```
grant select (employee) on employee to fred;
```

При спробі з'ясувати, чи має користувач право на виконання певних дій, MySQL розглядає зв'язану операторами OR комбінацію глобальних привілеїв цього користувача, його привілею рівня бази даних, привілею рівня таблиці й привілеї рівня стовпця.

Використання REVOKE

Оператор REVOKE - протилежність оператора GRANT. Він використовується для того, щоб позбавити користувача привілеїв. Наприклад:

```
revoke all on employee.* from fred;
```

Загальна форма оператора REVOKE виглядає так:

```
REVOKE привілея [(список_стовпців)]
```

```
[, привілея [(список_стовпців)] ... ]
```

```
ON {ім'я_таблиці | * | *.* | ім'яБД.*}
```

```
FROM ім'я_користувача [, ім'я_користувача ...]
```

Як бачите, оператор REVOKE використовує, в основному, ті ж вираження, що й оператор GRANT, але з їхньою допомогою скасовуються відповідні привілеї.

3 Таблиці привілеїв

Дані, які змінюються за допомогою операторів GRANT й REVOKE, зберігаються в базі даних з ім'ям mysql. Замість використання GRANT й REVOKE ви можете змінити таблиці в згоді базі даних безпосередньо, якщо, звичайно, ви знаєте, що робите. Можна також прочитати з них дані, що може виявитися корисним при рішенні проблем прав доступу, які можуть іноді виникати.

Після безпосередньої модифікації цих таблиць необхідно виконати оператор flush privileges;

щоб зміни набули чинності ,

У базі даних mysql є шість таблиць:

- user
- tables_priv
- db
- columns_priv
- host
- func

Тільки перші п'ять із цих таблиць стосуються привілеїв користувача. (Таблиця func зберігає інформацію про обумовлений користувачами функціях).

Перші три таблиці - user, db й host - використовуються для того, щоб з'ясувати, чи дозволено вам з'єднатися з базою даних. Всі п'ять із таблиць привілеїв використовуються для того, щоб з'ясувати, чи маєте ви право виконати запитану команду.

Таблиця user

Таблиця user містить інформацію про набір глобальних привілеїв користувача. Таблиця user складається з наступних стовпців.

Стовпці області дії. Вони використовуються для того, щоб визначити, коли застосовна той або інший рядок привілеїв. Ці стовпці наступні.

Host. Ім'я хосту, з якого підключається користувач.

User. Ім'я користувача.

Password. Пароль користувача, шифрований функцією PASSWORD ().

Стовпці привілеїв. Кожний з них відповідає одній із глобальних привілеїв. Допускаються значення Y (якщо користувач наділений соотвествующей глобальним привілеєм) і N (якщо користувач такої не має). Нижче наведений список імен цих стовпців.

Select_priv	Shutdown_priv
Insert_priv	Process_priv
Update_priv	File_priv
Delete_priv	Show_db_priv
Index_priv	Super_priv
Alter_priv	Create_tmp_table_priv
Create_priv	Lock_tables_priv
Drop_priv	Execute_priv
Grant_priv	Repl_slave_priv
References_priv	Repl_client_priv
Reload_priv	

Стовпці захисту з'єднання. Вони представляють інформацію з виразу REQUIRE оператора GRANT. Ці стовпці наступні.

ssl_type
ssl_cipher
x509_issuer
x509_subject

Стовпці обмежень. Вони містять інформацію про обмеження, які могли бути зазначені наприкінці оператора GRANT й які стосуються використання ресурсів користувачем. Ці стовпці наступні:

max_questions

max_updates

max_connections

Таблиця db

Таблиця db зберігає користувальницькі привілеї для конкретних баз даних і складається з наступних стовпців.

Стовпці області дії. MySQL використовує їх для того, щоб визначити, коли застосовний той або інший рядок привілеїв. Ці стовпці наступні:

Host

Db

User

Стовпці привілеїв. Вони вказують, чи буде дана комбінація Host, Db й User наділена відповідним привілеєм. Можуть містити значення Y або N. Ці стовпці наступні:

Select_priv

Create_priv

Insert_priv

Drop_priv

Update_priv

Grant_priv

Delete_priv

Create_tmp_table_priv

Index_priv

Lock_tables_priv

Alter_priv

Таблиця host

MySQL звертається до таблиці host, коли потрібно знайти ім'я вільного хосту в таблиці db. Цього не можна зробити за допомогою оператора GRANT, але відповідне значення можна встановити вручну. Таблиця складається з наступних стовпців.

Стовпці області дії. MySQL використовує їх для того, щоб визначити, коли застосовний той або інший рядок привілеїв. Тут кожен рядок містить інформацію про одну базу даних і про ім'я хосту, з якого здійснюється доступ до неї. Стовпці, про які мова йде, наведені нижче.

Host

Db

Стовпці привілеїв. Вони вказують, чи буде дана комбінація Host й Db наділена відповідним привілеєм, і можуть містити значення Y або N. Ці стовпці наступні:

Select_priv

Create_priv

Insert_priv

Drop_priv

Update_priv

Grant_priv

Delete_priv

Create_tmp_table_priv

Index_priv

Lock_tables_priv

Alter_priv

Таблиця tables_priv

У таблиці tables_priv відображається інформація про користувальницькі привілеї, що стосуються окремих таблиць. Вона складається з наступних стовпців.

Стовпці області дії. Визначають, коли застосовна той або інший рядок привілеїв. Ці стовпці наступні.

Host

Db

User

Table_name,

Column_name

Стовпець Column_priv. Визначає, якими привілеями наділений об'єкт, заданий стовпцями області дії. Може містити наступні значення: Select, Insert, Update й References.

Стовпець Column_priv. Визначає, якими привілеями наділений користувач відносно всіх стовпців даної таблиці. Може містити наступні значення: Select, Insert, Update й References. Якщо яка-небудь із привілеїв тут відсутня, MySQL може звернутися до таблиці columns_priv за більше докладною інформацією про те, що дозволено що не дозволено робити із зазначеними стовпцями таблиці.

Таблиця columns_priv

У таблиці columns_priv відображається інформація про користувальницькі привілеї, що стосуються окремих стовпців. Вона складається з наступних стовпців.

Стовпці області дії. Аналогічні стовпцям таблиць, про які говорилось вище, але є додатковий стовпець Table_name, що вказує ім'я таблиці, до якої ставиться надаваний привілей.

Ці стовпці наступні:

Host

Db

User

Table_name

Стовпці донора. Містять інформацію про те, хто й коли надав привілеї. Ці стовпці наступні.

Grantor

Timestamp

Стовпець Table_priv. Визначає, якими привілеями наділений Host/ Db/User відносно таблиці, зазначеної в Table_name, Може містити наступні значення: Select, Insert, Update, Delete, Create, Drop, Grant, References, Index й Alter.

Стовпець Timestamp. Повідомляє про те, коли була надана привілея.

Питання та завдання до контролю знань студентів

- 1 Привілей GRANT OPTION дозволяє користувачеві
 - а) завантажувати дані з файлу;
 - б) передавати свої привілеї іншим користувачам;
 - в) зареєструватися в системі;
 - г) оновляти привілею.
- 2 Привілей USAGE дозволяє користувачеві
 - а) завантажувати дані з файлу;
 - б) передавати свої привілеї іншим користувачам;
 - в) зареєструватися в системі;
 - г) оновляти привілею.
- 3 Привілей RELOAD дозволяє користувачеві
 - а) завантажувати дані з файлу;

- б) передавати свої привілеї іншим користувачам;
- в) зареєструватися в системі;
- г) оновляти привілею.

4 Привілей FILE дозволяє користувачеві

- а) завантажувати дані з файлу;
- б) передавати свої привілеї іншим користувачам;
- в) зареєструватися в системі;
- г) оновляти привілею.

5 Значення стовпця `tables_priv`, `table_priv`

- а) містить список привілеїв користувача для даної таблиці;
- б) містить значення Y або N, залежно від того, чи дозволений користувачеві доступ до даної таблиці;
- в) указує окремий привілей, який наділений користувач у відносині даної таблиці;
- г) указує, чи існує в таблиці `columns_priv` елемент, що відповідає даній таблиці.

6 Створіть оператор GRANT, що створює обліковий запис для користувача під ім'ям `bill` з паролем `secret`, що повинен мати право вибирати, модифікувати, додавати й видаляти дані з таблиці `department`.

7 Створіть оператор REVOKE, що скасовує надані привілеї для зазначеного користувача.