МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ФАХОВИЙ КОЛЕДЖ РАКЕТНО-КОСМІЧНОГО МАШИНОБУДУВАННЯ
ДНІПРОВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ім. О. ГОНЧАРА

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

«Бази даних»

ЗІ СПЕЦІАЛЬНОСТІ 121

«ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

Розробила: _____Ланська С.С.

Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Дніпро

2017

Лекція № 1

з навчальної дисципліни «Бази даних»

Тема Введення в бази даних

Мета заняття Ознайомити студентів з основними поняттями й визначення - ми теорії баз даних. Дати характеристику складових банку даних.

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Основні поняття й визначення теорії баз даних.

2 Характеристика складових банку даних.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 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'єкті. 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 множини зв'язаних файлів.

Система управління базами даних (СУБД) – сукупність язикових і програмних засобів, призначених для створення, ведення й спільного використання БД багатьма користувачами.

Автоматизована інформаційна система (АІС) – це система, що реалізує автоматизований збір, обробку, маніпулювання даними, функціонує на основі ЕОМ й інших технічних засобів і включає відповідне програмне забезпечення (ПЗ) та персонал. Надалі в цій якості буде використатися термін *інформаційна система* (ІС), що має на увазі поняття автоматизована. ІС 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 освіті, виробництві, бізнесі, науці й інших областях. У більшості випадків прибігають до розбивки всієї множини об'єктів Про на групи об'єктів, споріднених за структурою, що мають однакові властивості. Наприклад. у кожному фрагменті Про виділяється система об'єктів і зв'язків між об’єктами, виділяються процеси, що відбуваються в цьому фрагменті, а також кінцеве число користувачів.

2 Характеристика складових банку даних

Банк даних (БнД) є різновидом ІС. БнД – це система спеціальним образом організованих даних: баз даних, програмних, технічних, язикових, організаційно – методичних засобів, призначених для забезпечення централізованого накопичення 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 1.1.

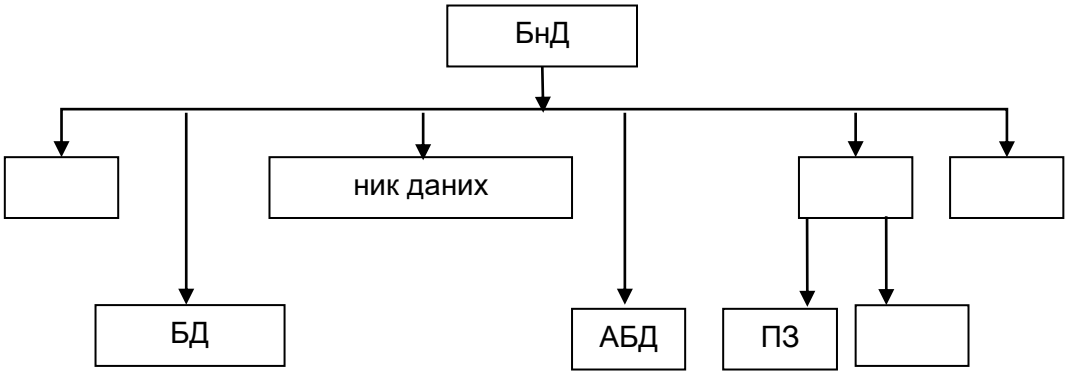


Рисунок 1.1 – Банк даних

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 відносини, 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 само описуваною, 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.

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 використають 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 ураховувати всі зміни, які вносяться іншими додатками. Така координація роботи додатків здійснюється СУБД.

Питання та завдання до контролю знань студентів

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?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні предметної (циклової)
комісії програмної інженерії
Протокол № 8 від130.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 2

з навчальної дисципліни «Бази даних»

Тема Розвиток технологій обробки даних.

Мета заняття Ознайомити студентів з поняттями баз даних та розвитком технологій обробки даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Поняття баз даних

2 Розвиток технологій обробки даних.

3 Сучасний стан технологій баз даних.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Поняття баз даних

База даних – це єдине, містке сховище різноманітних даних й описів їхніх структур, що після свого визначення, здійснюваного окремо й незалежно від додатків, використається 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.

В інформаційній системі дані можуть бути організовані в одну або декілька БД.

Інформаційна система може бути персональною – однокористувальницькою, тобто в один й той самий час до БД може одержати доступ тільки один користувач й може бути багатокористувальницькою – до бази даних одночасно можуть одержати доступ відразу кілька користувачів. Різновид створюваної інформаційної системи визначається потребами користувачів і тих обчислювальних ресурсів, які виділяються для ЫС.

БД, як правило, створюється як загальний ресурс усього підприємства, де дані є інтегрованими й загальними. Під поняттям інтегровані дані мається на увазі можливість представити базу даних як об'єднання декількох окремих файлів даних, які повністю або частково не перекриваються. Під поняттям загальні дані мається на увазі можливість використання окремих областей даних у БД декількома різними користувачами для різних цілей. Причому такий доступ у багато користувальницькій ІС може здійснюватися в оди й той самий час. Одним з наслідків даного положення є те, що будь–який конкретний користувач має справу з якоюсь невеликою частиною всієї БД, і одну й ту саму область БД два різних користувачі можуть сприймати по–різному.

Серед користувачів БД можна виділити чотири категорії осіб, кожна з яких має своє коло інтересів і пов'язана з певним етапом розробки й існування БД.

- адміністратори даних і баз даних;
- розроблювачі баз даних;
- прикладні програмісти;
- кінцеві користувачі.

2 Розвиток технологій обробки даних.

Програмне забезпечення перших СУБД, пов'язане з обробкою елементів або груп елементів даних. У подібній ситуації доступ до одних і тих же даних вже може здійснюватися з різноманітних прикладних програм різними шляхами. Програмне ж забезпечення повинно містити засоби зменшення надмірності. Різні логічні файли можуть бути отримані з одних і тих же фізичних даних. Багато елементів даних є загальними для різних прикладних програм. Дані адресуються на рівні полів і груп полів, а не на рівні записів. Пошук можливий за багатьма ключам. Ускладнення форм організації даних не впливає на прикладні програми.

Творці перших СУБД орієнтувалися на організацію баз даних на великих машинах (IBM 360/370, ЄС-ЕОМ) і на міні-ЕОМ (PDP11) із зберіганням їх у зовнішній пам'яті центральної ЕОМ. Інтерактивний режим користувача забезпечувався за допомогою консольних терміналів, використовуваних в якості пристроїв введення - виведення центральної ЕОМ. Програми обробки інформації баз даних використовували такі ж мови програмування, як і числові завдання.

У міру накопичення досвіду використання перших СУБД досить скоро стало очевидно, що необхідний додатковий рівень забезпечення програмними засобами логічної і фізичної незалежності даних. Логічна незалежність передбачає не зміна прикладних програм при зміні логічної структури даних. Фізична незалежність означає, що фізичне розташування і організація даних можуть змінюватися, не викликаючи змін в загальній логічній структурі даних і прикладних програмах. У таких системах обов'язково потрібно мати: адміністратора баз даних; ефективний захист даних; мова опису даних; мова запитів для користувача і мови програмування для прикладного програміста; ефективну систему контролю, статистики та виконання роботи системи.

Саме до цього часу відносяться такі досягнення:

– запропонована нова модель даних - реляційна, по обґрунтуванню і формалізації якої проводяться серйозні роботи; створюється перша СУБД, що реалізує цю ідеологію;

– виконуються теоретичні роботи по оптимізації запитів і управлінню розподіленим доступом до централізованої бази даних;

– великі успіхи досягнуті в області адміністрування даних.

І в той же самий час:

– функції управління розподілом ресурсів в основному здійснюються засобами операційної системи (ОС);

– маніпулювання даними реалізується за допомогою мов низького рівня.

Цей етап розвитку СУБД пов'язаний з організацією централізованих баз даних на великих машинах під управлінням потужних мультипрограмних операційних систем (MVS, SVM, OSRV, RSX, UNIX). Питання та завдання до контролю знань студентів.

Наступна фаза розвитку технології баз даних збігається за часом і багато в чому визначається появою і захопленням персональними комп'ютерами (ПК) всіх сфер людського життя: діловий, виробничої, наукової, побутової. З'явилося безліч простих у використанні і інтуїтивно зрозумілих програм з дружнім і зручним інтерфейсом, що дозволяють не тільки проводити різного роду обчислення, формувати тексти, документи, таблиці, графічні зображення, а й автоматизувати багато сфер діяльності.

Загальне визнання отримала інтуїтивно зрозуміла реляційна модель даних Кодда, що має в своїй основі капітальну теоретичну базу. Виникло доступний засіб, яке дозволяло автоматизувати процес зберігання, поповнення та обробки наявної інформації, вести документацію і автоматизувати багато облікові функції - в конторі, у відділі, будинки. На ринку з'явився новий різновид програмних продуктів - системи управління базами даних, які наділяють навіть початківця користувача можливістю організувати зберігання даних за допомогою вбудованих в СУБД різних інструментальних засобів: построителей, майстрів, конструкторів, організувати формування різноманітних форм введення і перегляду даних, звітів і багато чого іншого .

Однак, широка "демократизація" процесу доступу до ПК мала і негативну сторону. Негативність цього процесу "демократизації" полягала в тому, що вона привернула в область обробки даних величезну армію дилетантів, яка не

набувши систематичних знань в даній області і маючи в своєму розпорядженні комп'ютерного "Сезама", наплодили "велика кількість нетривких баз даних і додатків до них , що не відображають достовірно об'єкти і зв'язку обраної предметної області реального світу. Положення деяких фірм погіршили й деякі програмісти, прийнявши зовсім необгрунтовано рішення здійснити розробку СУБД св оімі силами, використовуючи стандартні мови програмування. Час показав, що такі системи не витримали гостру конкуренцію провідних фірм-виробників програмних продуктів в області створення систем обробки даних.

Основні характеристики СУБД, що встановлюються на персональних комп'ютерах:

1 Підтримка реляційної моделі даних.

2 Порівняно невисокі вимоги до технічних параметрів апаратного забезпечення.

3 Орієнтація на створення баз даних в монопольному режимі роботи.

4 Практична відсутність функцій адміністрування БД.

5 Розвинений і зручний інтерфейс.

6 Різноманітний інструментарій для розробки готових додатків без програмування.

7 Підтримка як з'явилися нові високорівневих мов маніпулювання даними, так і низькорівневих мов маніпулювання даними на рівні окремих рядків таблиці.

8 Підтримка тільки зовнішнього рівня предВідношення моделі даних.

9 Відсутність коштів підтримки посилальної і структурної цілісності БД.

Найбільшу популярність і поширення набули такі СУБД, як: Dbase III +, DbaseIV, Paradox, FoxPro, Clipper.

Потреби в персональних СУБД не всихають, оскільки число власників персональних комп'ютерів зростає з великою швидкістю. У багатьох випадках для вирішення необхідних проблем дешевше і простіше обійтися засобами індивідуального СУБД. Тому на ринок виходять все нові сучасні, вбирають в себе останні технічні та наукові досягнення в області обробки інформації версії персональних СУБД.

Ейфорія від процесу "персоналізації" баз даних закінчилася усвідомленням можливостей таких баз і сфер їх застосування. Завдання "інтеграції" інформації вийшла на перший план. Для її вирішення знадобилося розробляти нові підходи і концепції в організації баз даних, відшукувати нові апаратні і програмні їх реалізації. Розробникам комп'ютерних і програмних систем довелося високими темпами нарощувати потужність комп'ютерів і розширювати можливості колективної роботи з даними за рахунок об'єднання даних в мережі.

В результаті весь світ обплутаний павутиною різних мереж - локальних, корпоративних, глобальних, за якими миттєво, від комп'ютера до комп'ютера, передається різноманітна інформація. Такий стиль життя диктує свої правила. Кожна знову виникла фірма знає, що її першочергове завдання - оснастити себе комп'ютерами, слідом за якою необхідно вирішити інші - організувати куплені комп'ютери в локальну мережу і створити свою інформаційну систему, а потім - забезпечити вихід в глобальну мережу, з тим щоб мати доступ до будь-якої знадобилася інформації.

Швидкими темпами пройшов свій еволюційний шлях і основний інструмент для роботи з даними - СУБД: від одного користувача систем, які встановлюються на окремому ПК, до багатокористувацьких і серверних систем. На даний момент сервери баз даних являють собою найбільш потужне і надійне прикладне програмне забезпечення для колективної роботи з даними. Однак відома істина, що за все треба платити, і в цьому випадку довела свою справедливість: володіючи великими можливостями такі системи відрізняються підвищеною складністю і вартістю.

3 Сучасний стан технологій баз даних

Всебічному розгляду сучасних принципів організації баз даних присвячений майже весь подальший матеріал цієї книги. В даному підрозділі, завершуючи аналіз процесу історичного розвитку систем обробки інформації, в тому числі і технології баз даних, тільки коротко позначимо ці принципи.

1 Значна частина сучасних СУБД здатна працювати на комп'ютерах різної архітектури під управлінням різних операційних систем.

2 Переважна більшість сучасних СУБД забезпечують підтримку повної реляційної моделі даних, забезпечуючи цілісність категорій і цілісність на рівні посилань.

3 Сучасні СУБД для визначення даних і маніпуляції ними спираються на прийняті стандарти в області мов, а при обміні даними між різними СУБД базуються на існуючих технологіях з обміну інформацією.

4 Багато існуючі СУБД відносяться до так званим мережевим СУБД, які призначені для підтримки багато режиму роботи з базою даних і підтримки можливості децентралізованого зберігання даних.

5 Такі СУБД мають розвинені засоби адміністрування баз даних і засоби захисту інформації, що зберігається в них інформації.

6 Подібні СУБД мають засоби підключення клієнтських додатків, розроблених на ПК, і кошти експорту даних з форматів персональних СУБД.

7 Практично всі сучасні персональні СУБД також мають в складі свого ПО кошти, що дозволяють їм здійснювати підключення до мережевих СУБД.

8 Сучасні СУБД характеризуються дослідами застосування концепції об'єктно-орієнтованого підходу, що сприяє підвищенню рівня абстракції баз даних, що є перспективним етапом на шляху розвитку технологій баз даних.

В даний час число використовуваних СУБД перевищує кілька десятків.

Питання та завдання до контролю знань студентів

1 Поясните основні концепції технології баз даних.

2 Чим характеризується сучасний стан технології баз даних?

3 Якими можливостями володіє сучасна СУБД?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 3

з навчальної дисципліни «Бази даних»

Тема Загальна характеристика систем управління базами даних (СУБД)

Мета заняття Ознайомити студентів з основними поняттями про системи управління базами даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Поняття про системи управління базами даних.

2 Класифікація СУБД.

2 Компоненти середовища СУБД.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Поняття про системи управління базами даних

СУБД - це програмне забезпечення, за допомогою якого користувачі можуть визначати, створювати і підтримувати базу даних, а також здійснювати до неї контрольований доступ.

Система управління 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.

2 Класифікація СУБД

За ступенем універсальності розрізняються два класи СУБД - системи загального призначення і спеціалізовані системи.

СУБД загального призначення не орієнтовані на яку–небудь конкретну предметну область або на інформаційні потреби конкретної групи користувачів. Кожна система такого роду реалізується як програмний продукт, здатний функціонувати на деякій моделі ЕОМ у певній операційній обстановці. СУБД загального призначення має засоби настроювання на роботу з конкретної БД в умовах конкретного застосування. Використання СУБД загального призначення як інструментальний засіб для створення ІС, заснованих на технології БД, дозволяє істотно скорочувати строки розробки, заощаджувати трудові ресурси. Запас "потужності" таких систем дозволяє підтримувати еволюційний процес розвитку ІС у рамках її життєвого циклу.

Однак у деяких ситуаціях доступні СУБД загального призначення не дозволяють домогтися необхідних проектних й експлуатаційних характеристик (продуктивність, займаний обсяг пам'яті та інше).

Проте, *створення спеціалізованих СУБД* – досить трудомісткий процес і для того, щоб його реалізувати, потрібні дуже вагомі підстави. У процесі реалізації своїх функцій СУБД постійно взаємодіє з базою даних і з іншими прикладними програмними продуктами користувача, призначеними для роботи з даної БД і називаними додатками. Для того щоб СУБД успішно справлялася зі своїми завданнями, вона повинна мати певні можливості.

Можна дати наступну узагальнену характеристику можливостям сучасних СУБД:

1 СУБД включає мову визначення даних, за допомогою якої можна визначити базу даних, її структуру, типи даних, а також засобу завдання обмежень для збереженої інформації. У багатокористувацькому варіанті СУБД ця мова дозволяє формувати предВідношення як деяку підмножину бази даних, з підтримкою якої користувач може створювати свій погляд на збережені дані, забезпечувати додатковий рівень безпеки даних і багато чого іншого.

2 СУБД дозволяє вставляти, видаляти, обновляти й витягати інформацію з бази даних за допомогою мови управління даними. Ця мова носить ще одну назву – мова запитів і дозволяє користувачеві формувати різні по змісту запити до бази даних.

3 Більшість СУБД можуть працювати на комп'ютерах з різною архітектурою й під різними операційними системами, причому на роботу користувача при доступі до даних практично тип платформи впливу не робить.

4 Багатокористувацькі СУБД мають досить розвинені засоби адміністрування БД.

СУБД надає контрольований доступ до бази даних за допомогою:

– системи забезпечення безпеки, що запобігає несанкціонований доступ до інформації бази даних;

– системи підтримки цілісності бази даних, що забезпечує несуперечливий стан збережених даних;

─ системи управління паралельною роботою додатків, що контролює процеси їх. спільного доступу до бази даних;

─ системи відновлення, що дозволяє відновити базу даних до попереднього несуперечливого стану, порушеного в результаті апаратного або програмного збою;

─ доступного користувачам каталогу, що містить опис збереженої в базі дані інформації.

Стосовно до кожної конкретного СУБД наведені загальні відомості мають потребу в уточненні. Причому повною мірою всіма перерахованими можливостями володіють тільки деякі сучасні багатокористувацькі СУБД, використання яких припускає майже 100–процентну надійність при різних збоях у роботі СУБД.

3 Компоненти середовища СУБД

У середовищі СУБД можна виділити кілька основних компонентів: дані, користувачі, апаратне забезпечення, програмне забезпечення, процедури. Схематично це показано на рисунку 2.1.

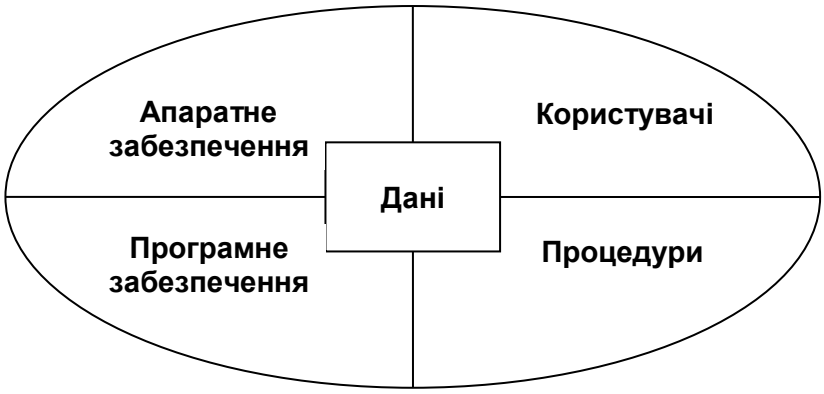


Рисунок 3.1 – Компоненти середовища СУБД

З точки зору користувача саме дані є найбільш важливим компонентом середовища СУБД. Заради спілкування з ними на належному рівні потрібна наявність інших компонентів. База даних зберігає як робочі дані, так і характеристики даних. Робочі дані містяться в БД в структурованому вигляді. Структури даних визначає схема бази даних, яка формується на етапі проектування БД. БД, перш за все, повинна містити:

– імена, типи і розміри елементів даних;

– імена зв'язків;

– обмеження цілісності даних;

– імена зареєстрованих користувачів і їх права щодо доступу до даних;

– використовувані індекси і структури зберігання.

Для зберігання даних і різного роду програмного забезпечення, а також функціонування ІС необхідно апаратне забезпечення. *Апаратне забезпечення* - це набір фізичних пристроїв, на яких існують база даних, СУБД і інші компоненти інформаційної системи. Апаратне забезпечення має відповідати вимогам використовує його організації, вимогам вживається СУБД, вимогам розроблюваної бази даних. Це може бути один персональний комп'ютер або мережу. Для успішної роботи інформаційної системи всі компоненти СУБД повинні бути ретельно підібрані, з тим щоб вони були в стані спільно працювати узгоджено.

Наступним компонентом є *програмне забезпечення*. До його складових необхідно віднести:

– операційну систему, включаючи мережеве ПО, якщо СУБД працює в мережі;

– програмне забезпечення самої СУБД;

– прикладні програми-додатки.

Прикладні програми, які створюються для потреб організації, можуть бути написані на стандартному алгоритмічній мові програмування високого рівня, наприклад, такому як C, Ada, Pascal, з впровадженими операторами мови SQL, який відносять до мов четвертого покоління. Прикладні програми використовують кошти СУБД для звернення до даних і їх обробки, створюючи форми, звіти та інші документи, необхідні для роботи даної організації. Багато сучасних СУБД мають спеціальні програмні засоби, звані інструментами, для швидкої розробки додатків з вживанням вбудованих непроцедурних мов.

Серед користувачів БД можна виділити чотири категорії осіб, кожна з яких має своє коло інтересів і пов'язана з певним етапом розробки і існування БД:

– адміністратори даних і баз даних;

– розробники баз даних;

– прикладні програмісти;

– кінцеві користувачі.

Дані - це важливий ресурс організації, і ними треба вміло керувати. Настільки важлива функція покладена на фахівців певного роду - *Адміністраторів даних (АД).* Вони працюють з даними з самого початку процесу проектування бази даних і відповідають за концептуальне і логічне проектування бази даних, управління даними, розробку і супровід стандартів, бізнес-правил і ділових процедур.

Фізичне проектування і фізична реалізація бази даних, забезпечення безпеки і цілісності даних, забезпечення максимальної про- тивність додатків - це область дії компетенції *Адміністратор бази даних (АБД).* Як видно в порівнянні з АТ, обов'язки АБД більше пов'язані з вирішенням технічних проблем.

Розробники баз даних - це така група користувачів, яка функціонує під час проектування, створення і реорганізації бази даних і результатом діяльності якої є добре спроектована БД, що забезпечує достовірної та несуперечливої інформацією всіх зацікавлених в цьому осіб. При проектуванні великих БД все розробники розпадаються на дві групи:

– розробники логічної бази даних;

– розробники фізичної бази даних.

Розробники логічної бази даних займаються виявленням цікавлять об'єктів і їх властивостей, зв'язків між об'єктами і тих обмежень, які необхідно накласти на збережені дані. Скажімо, при складанні розкладу занять студентів деякого університету, що має корпусу для занять в різних частинах міста (що дуже часто зустрічається), можна визначити наступні бізнес-правила:

– протягом одного навчального дня заняття групи студентів повинні проводитися в одному навчальному корпусі або в корпусах, розташованих поруч, щоб вони не були змушені витрачати час на переїзди;

– протягом одного навчального дня заняття, які повинні проводитися одним викладачем, зобов'язані плануватися в одному навчальному корпусі або в корпусах, розташованих поруч, щоб і він не був змушений витрачати час на переїзди.

Здійснення своєї діяльності зазначені розробники виконують в два етапи:

– розробка концептуальної моделі БД;

– розробка логічної моделі БД.

Розробники фізичної бази даних повинні розбиратися в функціональних можливостях обраної СУБД, знати всі варіанти можливого фізичного втілення отриманої логічної моделі даних і розуміти їхні переваги і недоліки, з тим щоб вибрати найбільш оптимальний варіант для даного випадку і правильно вибудувати всю стратегію зберігання і використання даних.

Відразу після створення бази даних слід приступити до розробки додатків, що надають користувачам необхідні їм функціональні можливості. Саме цю роботу виконують прикладні програмісти. Програми можуть створюватися на різних мовах програмування третього і четвертого покоління.

Користувачі. База даних проектується, створюється і підтримується для того, щоб обслуговувати інформаційні потреби кінцевих користувачів. Їх зазвичай називають клієнтами. Серед них є і недосвідчені користувачі, нічого не знають про базу даних. Для них розробляються такі програми, які дозволяють максимально спростити операції вони виконують, наприклад, шляхом вибору команди меню. Більш кваліфікованим користувачам, добре знайомим з моделлю бази даних, з можливостями встановленої СУБД, в принципі, під силу вирішення будь-яких інформаційних задач шляхом використання інструментарію СУБД, мови структурованих запитів SQL або створених своїми руками додатків.

Питання та завдання до контролю знань студентів

4 Якими властивостями повинна володіти проектована база даних?

5 Опишіть компонентний склад середовища СУБД і призначення її компонентів.

6 Поясните переваги, які несе в собі використання СУБД.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 4

з навчальної дисципліни «Бази даних»

Тема Трирівнева архітектура бази даних

Мета заняття Ввести поняття трирівневої архітектури бази даних та дати характеристику її рівнів

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Трирівнева архітектура бази даних.

2 Призначення трирівневої архітектури.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Трирівнева архітектура бази даних

Одним з найважливіших аспектів розвитку СУБД стала ідея відділення логічної структури БД і маніпуляцій даними, необхідних користувачам, від фізичного предВідношення, 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–, двох– і

трирівневі системи. У цей час найчастіше підтримується трирівнева архітектура опису БД (див. рис. 4.1), із трьома рівнями абстракції, на яких можна розглядати базу даних.

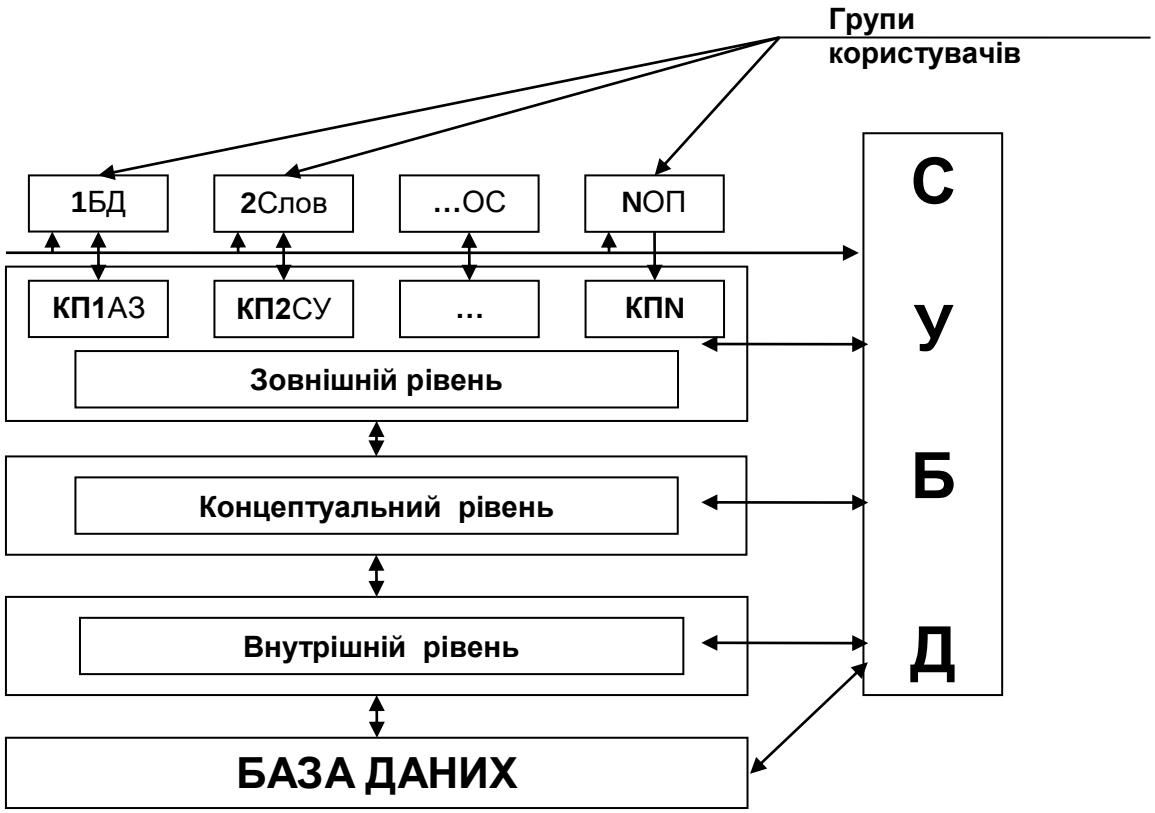


Рисунок 4.1 – Трирівнева архітектура СУБД

Така архітектура включає:

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

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

– *концептуальний рівень* ПредВідношення даних, призначений для відображення зовнішнього рівня на внутрішній рівень, а також для забезпечення необхідної їхньої незалежності друг від друга; він пов'язаний з узагальненим предВідношенням 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

ПредВідношення й користувальницьких подань. А в 1975 році Комітет планування стандартів і норм SPARC (Standards Plan–ning 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 ПредВідношенням даних. На концептуальному рівні опис бази даних називають *концептуальною схемою*, а на найнижчому рівні абстракції – *внутрішньою схемою*.

2 Призначення трирівневої архітектури.

Основним призначенням трирівневої архітектури є забезпечення незалежності від даних. Суть цієї незалежності полягає в тім, 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 х трьох названих рівнів.

Зовнішній рівень – це користувальницький рівень. Користувачем може бути програміст, або кінцевий користувач, або адміністратор бази даних. ПредВідношення бази даних з погляду користувачів називається зовнішнім предВідношенням. 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 користувальницьке предВідношення, причому ті ж самі дані можуть відображатися по-різному в різних користувальницьких предВідношеннях. Наприклад, в інформаційній системі ВУЗ користувача 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 користувачів або орієнтовані на окремі аспекти предметної області називають *підсхемою.* При розгляді зовнішнього рівня БД важливо відзначити, що кожен тип користувачів може застосовувати для роботи із БД свою мову спілкування. Кінцеві користувачі вживають або мову запитів, або спеціальну мову, підтримувану додатками й утворюючу певні для користувача екранні форми й користувальницькі меню. Прикладні програмісти частіше застосовують або мови високого рівня, наприклад С++, 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.

Концептуальний рівень є проміжним рівнем у трирівневій архітектурі й забезпечує предВідношення 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 користувачів 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 предВідношення, 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.

Внутрішній рівень є третім рівнем архітектури БД. Внутрішнє предВідношення 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 використаються 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 відносини й навпаки. Вигодою такого перекладу є незалежність логічного й фізичного предВідношення 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 утворять 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 предВідношення на відповідну частину концептуальної схеми.

Питання та завдання до контролю знань студентів

4 Зобразіть схематично трирівневу архітектуру бази даних.

5 Поясніть основне призначення трирівневої архітектури бази даних.

6 Дайте порівняльну характеристику її зовнішньому, концептуальному й внутрішньому рівням.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 5

з навчальної дисципліни «Бази даних»

Тема Функції СУБД

Мета заняття Дати огляд основних функцій системи управління базами даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Функції СУБД

1.1 Керування даними в зовнішній пам'яті

1.2 Керування транзакціями

1.3 Відновлення бази даних

1.4 Підтримка мов БД

1.5 Словник даних

1.6 Керування паралельним доступом та буферами оперативної пам'яті

1.7 Контроль доступу до даних

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Функції СУБД

У СУБД часто виділяють систему розроблювача й систему часу виконання. Система розроблювача містить у собі компоненти СУБД, які використаються 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:

1.1 Керування даними в зовнішній пам'яті

Дана функція надає користувачам можливості 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 використаються 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 використає СУБД файлову систему, і якщо використає, то як організовані файли.

1.2 Керування транзакціями

Транзакція – це послідовність операцій над БД, що розглядаються 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 ні. Тому всі часткові зміни повинні бути скасовані для повернення бази даних у колишній, несуперечливий 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 багатокористувацьких 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 активності користувача стосовно БД. При відповідному керуванні паралельно транзакціям, що виконуються, з боку СУБД кожний з користувачів може в принципі відчувати себе єдиним користувачем СУБД.

1.3 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 бути в змозі 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);

− *жорсткі 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 незавершеної. Першу ситуацію можна розглядати як особливий вид м'якого апаратного збою. При виникненні останньої потрібно ліквідувати наслідки 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 використається для відновлення, повинна зберігатися особливо надійно. Найпоширенішим методом підтримки такої надлишкової інформації є ведення журналу змін БД.

1.4 Підтримка мов БД

Для роботи з базами даних використаються 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 Queiy Language – мова структурованих запитів). Мова SQL дозволяє визначати схему реляційної БД і маніпулювати даними.

1.5 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 інформації й спосіб її застосування у великому ступені залежить від типу використовуваної СУБД. Можна виділити наступну інформацію, що звичайно міститься 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.

1.6 Керування паралельним доступом та буферами оперативної пам'яті

Одна з основних цілей створення й використання СУБД полягає в тім, щоб множина користувачів могла 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 обновити дані. СУБД повинна гарантувати, що при одночасному доступі до бази даних багатьох користувачів подібних конфліктів не відбудеться.

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

1.7 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 життєздатної, 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 використається 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 інструмен–тами 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 накладно 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.

Питання та завдання до контролю знань студентів

1 Як здійснює СУБД керування даними в зовнішній пам'яті?

2 Дати визначення поняття транзакція.

3 В чому полягає керування транзакціями з боку СУБД?

4 Коли виникає необхідність відновлення бази даних? Завдяки чому це стає можливим?

5 Яка інформація міститься в словнику даних СУБД?

6 Коли виникає необхідність керування паралельним доступом до даних?

7 Що таке буферізація?

8 Назвіть та охарактеризуйте основні складові контроля доступу до даних.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 6

з навчальної дисципліни «Бази даних»

Тема Типова організація сучасної СУБД

Мета заняття Ввести поняття ядра СУБД та дати характеристику програмних компонентів середовища СУБД. Дати оглядову характеристику мов баз даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Типова організація сучасної СУБД

2 Мови баз даних

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

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-який конкретної ИС 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 уведенням 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 Мови баз даних

Для роботи з базами даних використаються спеціальні мови, називані *підмовами даних*, оскільки вони включають підмножину операторів, пов'язане з об'єктами й операціями баз даних, але не містять конструкції для виконання всіх обчислювальних операцій мов програмування високого рівня. Багато СУБД підтримують можливість впровадження операторів підмов даних у мови високого рівня, таких, наприклад, як COBOL, Fortran, Pascal, Ada, C. У цьому випадку мова високого рівня вважається базовим, а підмова даних - впровадженим. Для реалізації даного впровадження СУБД повинна зі свій механізм. Найбільше часто відкомпільований об'єктний модуль базової мови

компонується з бібліотекою, у яку поміщені функції підмови даних. Після цього отриманий програмний текст буде готів до виконання. Крім механізму впровадження, для більшості підмов даних також надаються засоби інтерактивного виконання їхніх операторів, що вводять користувачем безпосередньо зі свого термінала. З погляду користувача базова мова й підмова даних можуть бути сильно зв'язаними, якщо вони майже нерозрізнені, і слабко зв'язаними, якщо вони легко розрізняються. Деякі фахівці в області обробки даних прогнозують поступове просування до більш сильно зв'язаних систем, здатним надати користувачам більше уніфікований набір можливостей. У СУБД підтримується декілька спеціалізованих за своїми функціями підмов. Їх можна розбити на дві категорії:

– мова визначення даних БД - ЯОД (DDL - Data Definition Language);

– мова маніпулювання даними -ЯМД (DML - Data Manipulation Language).

Мова визначення даних. Мова визначення даних - описова мова, за допомогою якого описується предметна область: іменуються об'єкти, визначаються їхні властивості й зв'язки між об'єктами. Він використається головним чином для визначення логічної структури БД. Схема бази даних, виражена в термінах спеціальної мови визначення даних, складається з набору визначень. Мова ЯОД використається як для визначення нової схеми, так і для модифікації вже існуючої. Випливаючи з трирівневої архітектурі БД, можна було б припустити наявність підмов ЯОД для опису схем для кожного із трьох рівнів. Однак зложилося так, що в цей час використається тільки одна загальна мова ЯОД для опису зовнішніх і концептуальних схем. Результатом компіляції ЯОД - операторів є набір таблиць, збережений у системному каталозі, у якому втримуються метадані , тобто дані, які включають визначення записів, елементів даних, а також інші об'єкти, що представляють інтерес для користувачів або необхідні для роботи СУБД. Перед доступом до реальним даних СУБД звичайно звертається до системного каталогу. Таким чином, мова ЯОД призначена тільки для опису схем, його не можна використати для

керування даними. Схема створюється в процесі проектування бази даних, причому передбачається, що вона змінюється досить рідко.

Мови маніпулювання даними. Мова маніпулювання даними містить набір операторів маніпулювання даними, тобто операторів, що дозволяють заносити дані в БД, видаляти, модифікувати або вибирати існуючі дані. Хоча схема БД змінюється досить рідко, однак інформація, що втримується в базі даних, може мінятися часто - наприклад, щораз при вставці відомостей про нового співробітника або студента. Сукупність інформації, що зберігається в базі даних у будь-який певний момент часу, називається станом бази даних. Отже, однієї й тій же схемі бази даних може відповідати безліч її різних станів. Гнучкість керування БД визначається безліччю операцій, дозволених над даними. За допомогою цих операцій БД переводиться з одного стану в інше. Безліч операцій над даними можна класифікувати в такий спосіб:

– операції селекції;

– дії над даними:

– вибірка - читання екземпляра запису із БД;

– включення - уведення екземпляра запису в БД із установкою його зв'язків;

– видалення - виключення екземпляра запису із БД із установкою нових зв'язків;

– модифікація - зміна вмісту екземпляра запису й корекція зв'язків при необхідності.

Таким чином, одна з основних функцій СУБД полягає в підтримці мови маніпулювання даними, за допомогою якого користувач може створювати вирази для виконання перерахованих вище операцій з даними. Поняття маніпулювання даними застосовано як до зовнішнього й концептуального рівнів, так і до внутрішнього рівня. Однак на внутрішньому рівні для цього необхідно визначити дуже складні процедури низького рівня, здатні забезпечити ефективний доступ до даних. На більше високих рівнях, навпаки, акцент переноситься убік більшої простоти використання, і основні зусилля направляються на забезпечення ефективної взаємодії користувача із системою.

Мови маніпулювання даними діляться на два типи, що обумовлено корінним розходженням у підходах до роботи з даними, а отже, розходженням у базових конструкціях у роботі з даними:

Перший тип - це процедурний ЯМД. Працюючи із процедурним ЯМД, користувач повинен чітко уявляти собі те, що він повинен одержати, і як він може це зробити за допомогою засобів ЯМД. Ці засоби являють собою оператори над даними, які необхідно виконати в певному порядку для одержання необхідної інформації. Звичайно процедурна мова дозволяє витягти запис, обробити її й, залежно від отриманих результатів, витягти інший запис, що повинна бути піддана аналогічній обробці й т.д. Подібний процес витягу даних триває доти , поки не будуть витягнуті всі запитувані дані. Мови ЯМД мережних й ієрархічних СУБД є процедурними. У таких процедурних мовах запису розглядаються окремо . До процедурних мов маніпулювання даними ставляться й мови, що підтримують операції реляційної алгебри, що основоположник теорії реляційних баз даних Э. Ф. Кодд увів для керування реляційною базою даних. Реляційна алгебра - це процедурна мова обробки реляційних таблиць, де в якості операндів виступають таблиці в цілому.

Другий тип - це декларативний (непроцедурний) ЯМД. Декларативні мови ЯМД надають користувачеві засобу, що дозволяють указати лише те, які дані потрібні. Рішення питання про те, як їх варто витягати, бере на себе процесор даної мови, що працює із цілими наборами записів. Непроцедурні мови ЯМД дозволяють визначити весь набір необхідних даних за допомогою одного оператора витягу або відновлення. СУБД транслює вираження мовою ЯМД у процедуру (або набір процедур), що забезпечує маніпулювання викликаним набором записів. Даний підхід звільняє користувача від необхідності знати деталі внутрішньої реалізації структур даних й особливості алгоритмів, використовуваних для витягу й можливого перетворення даних. Непроцедурні мови простіше зрозуміти й використати, чим процедурні мови. Реляційні СУБД включають підтримку непроцедурних мов маніпулювання даними - найчастіше це мова структурованих запитів SQL або мова запитів за зразком QBE.

У цей час нормою є підтримка декларативної мови SQL, в основі якого лежить реляційне числення, уведене Э. Коддом. Ця мова стала стандартом для мов реляційних баз даних, що дозволяє використати той самий синтаксис і структуру команд при переході від однієї СУБД до іншої. Зараз практично всі найбільші розроблювані СУБД створюють свої продукти з використанням SQL. Слід зазначити, що мова SQL має відразу два компоненти: мова DDL (ЯОД) для опису структури бази даних, і мова DML (ЯМД) для вибірки й відновлення даних.

Іншою широко використовуваною мовою обробки даних є мова QBE, що заслужила репутацію одного з найпростіших способів витягу інформації з бази даних. Особливо це коштовно для користувачів, що не є професіоналами в цій області. Мова надає графічні засоби створення запитів на вибірку даних з використанням шаблонів. Відповідь на запит також являє собою графічну інформацію.

Частина непроцедурної мови ЯМД, що відповідає за витяг даних, називається *мовою запитів*. Мова запитів можна визначити як високорівневувузькоспеціалізовану мова, призначену для задоволення різних вимог по вибірці інформації з бази даних. Аналізуючи тенденції в розвитку мов обробки даних, можна припустити, що й надалі воно піде, швидше за все, по напрямку розвитку непроцедурних мов з використанням компонентів високого рівня, які часто називають "інструментами четвертого покоління". Очікується, що мови четвертого покоління дозволять підвищити продуктивність роботи на порядок за рахунок обмеження типів завдань, які можна буде вирішувати з їхньою допомогою. Сучасні СУБД уже пропонують цілий ряд таких інструментів - це різного роду генератори:

– екранних форм для швидкого створення шаблонів уведення й відображення даних;

– звітів на основі збереженої в базі інформації;

– запитів до бази даних;

– графічного подання даних у вигляді різного роду діаграм;

– додатків для створення програм обробки даних.

Питання та завдання до контролю знань студентів

1) Яке призначення ядра СУБД?
2) Дайте коротку характеристику програмних компонентів середовища СУБД:

– процесор запитів;
– компілятор мови визначення даних;
– препроцесор мови маніпулювання даними;
– контролер словника;
– контролер файлів;
– контролер бази даних.

3) Назвіть та дайте коротку характеристику складових контролера бази даних.
4) Яке призначення мови визначення даних?
5) Яке призначення мови маніпулювання даними?
6) Дайте характеристику непроцедурного типу мови маніпулювання даними.
7) Дайте характеристику непроцедурного типу мови маніпулювання даними.
8) До якого типу відноситься мова запитів?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 7

з навчальної дисципліни «Бази даних»

Тема Архітектура багатокористувацьких СУБД.

Мета заняття Ввести поняття багатокористувацьких систем. Розглянути дворівневі та трирівневі моделі технології «клієнт-сервер»

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 4 години

План проведення лекції

1 Тенденції розвитку багатокористувацьких систем
2 Моделі дворівневої технології "клієнт - сервер"
    2.1 Файловий сервер.
    2.2 Модель віддаленого доступу до даних.
    2.3 Модель сервера баз даних.
3 Сервер додатків. Трирівнева модель

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1Тенденції розвитку багатокористувацькихсистем

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 локальні мережі, 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 центральної 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. 7.1).

Рисунок 7.1 – 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 плині 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 використають мережі персональних комп'ютерів, і привела до більше ефективного з погляду експлуатаційних витрат системам. Цей підхід відбиває організаційну структуру компанії, що логічно складається 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. 7.2).

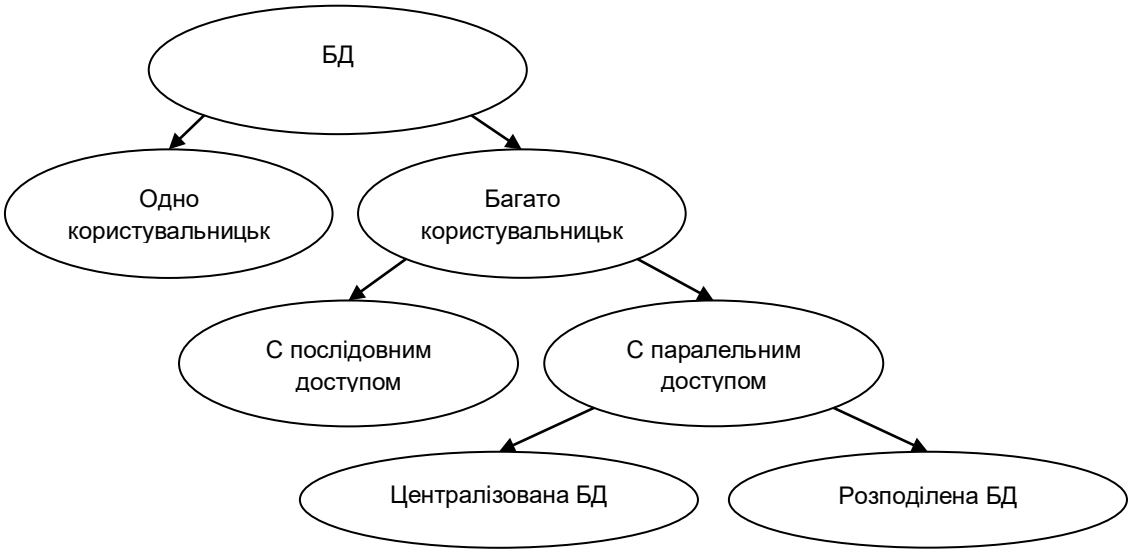


Рисунок 7.2 – Технології використання БД

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 використаються два типи технологій: технологія файлового сервера й технологія "клієнт - сервер", які є дворівневими структурами. Першу можна вважати часткою случаємо другий. Наступний розвиток останніх, прагнення усунути їх деякі недоліки викликали поява інших моделей і структур спільної роботи клієнтів і сервера.

2 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.7.3).

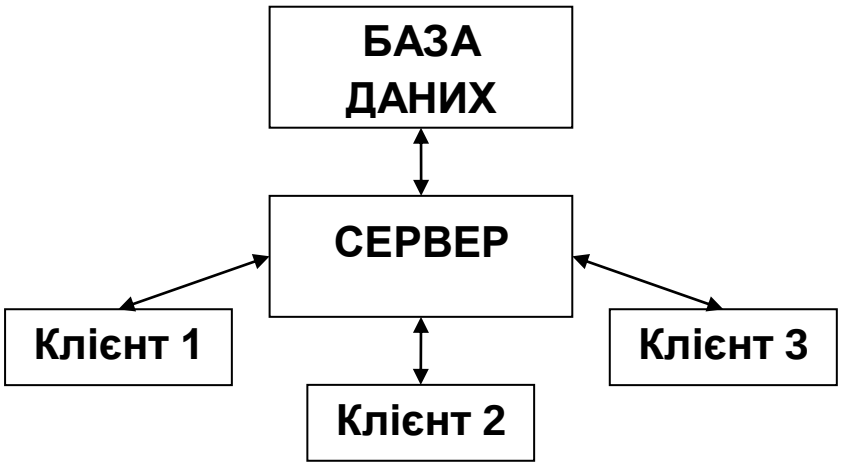


Рисунок 7.3 –. Структура системи БД із виділенням клієнтів і сервера

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 уведення й відображення даних* - презентаційна частина додатка - визначаються тим, що користувач бачить на своєму екрані, коли працює додаток. Тому основними завданнями цієї частини додатки 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 З, C*obol, SmallTalk, Pascal.*

3) Функції обробки даних пов'язані з обробкою даних усередині додатка. Даними управляє властиво СУБД. Для забезпечення доступу до даних використаються мова запитів і засобу маніпулювання даними стандартної мови SQL. Звичайно оператори мови SQL вбудовуються в мови, які використаються 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 дворівневої. Вона має кілька основних різновидів.

2.1 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. 7.4).

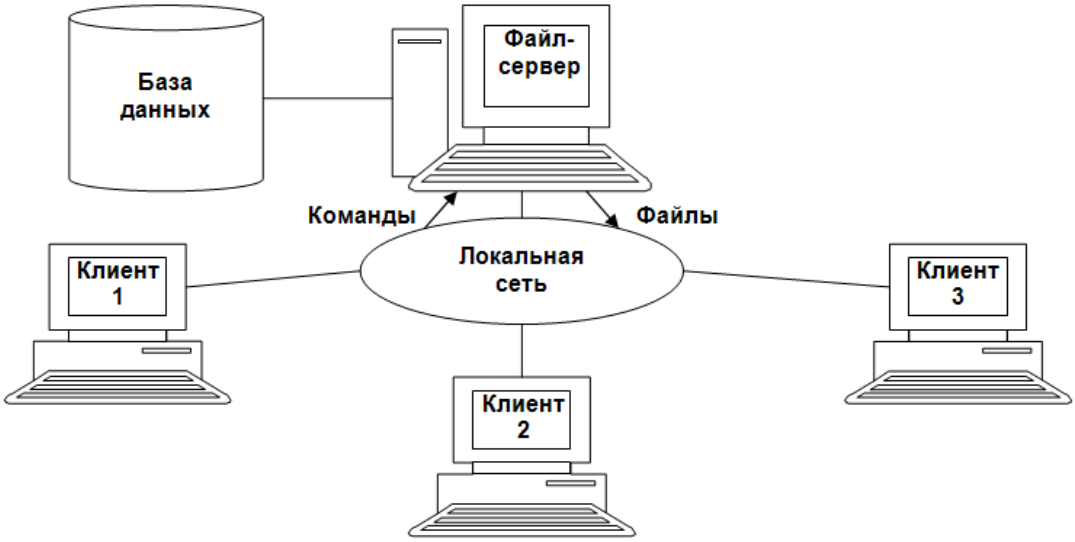


Рисунок 7.4 – Модель файлового сервера

СУБД посилає запити файловому 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 тім, 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.

2.2 Модель віддаленого доступу до даних.

У моделі віддаленого доступу база даних також зберігається на сервері. На сервері ж перебуває і ядро СУБД. На клієнті розташовуються частини додатка, що підтримують функції уведення й відображення даних і прикладні функції. Клієнт звертається до сервера із запитами мовою SQL. Структура моделі віддаленого доступу наведена на рисунку2.3.

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 операційної системі. З іншого боку, серверу БД виділяються тільки властиві йому функції; він завантажується операціями обробки даних, запитів і транзакцій.

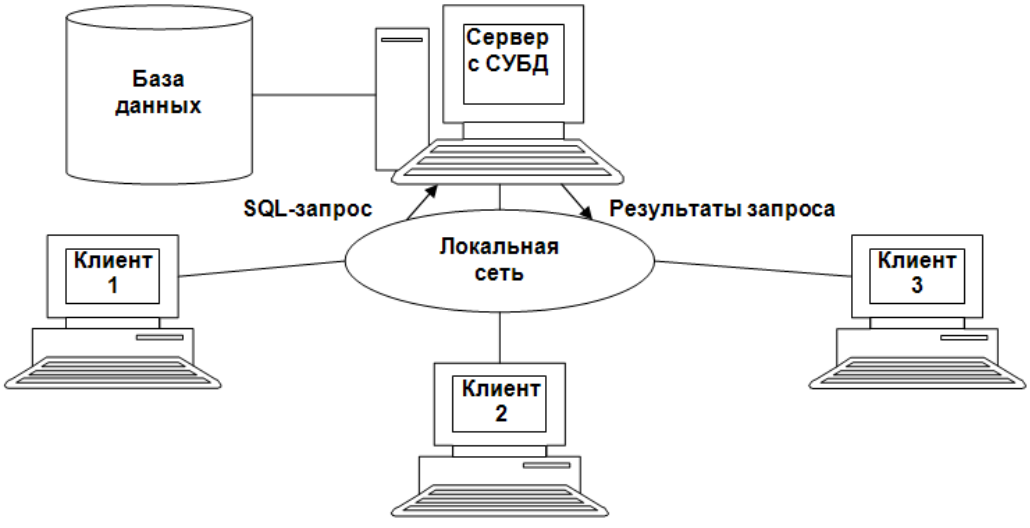


Рисунок 7.5 – Модель віддаленого доступу

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.

2.3 Модель сервера баз даних.

Технологію "клієнт - сервер" підтримують більшість сучасних СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server. Ця технологія є подальшим розвитком моделі віддаленого 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 використається для посилення додатків, що виконують багатокаскадні операції в БД. Тригери можуть викликати збережені процедури. У даній моделі (див. рис. 7.6) 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".

Рисунок 7.6 – Модель файлового 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 використається 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 трьохрівнева 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.2.5.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. 7.7, 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" (див. рис. 7.7.в).

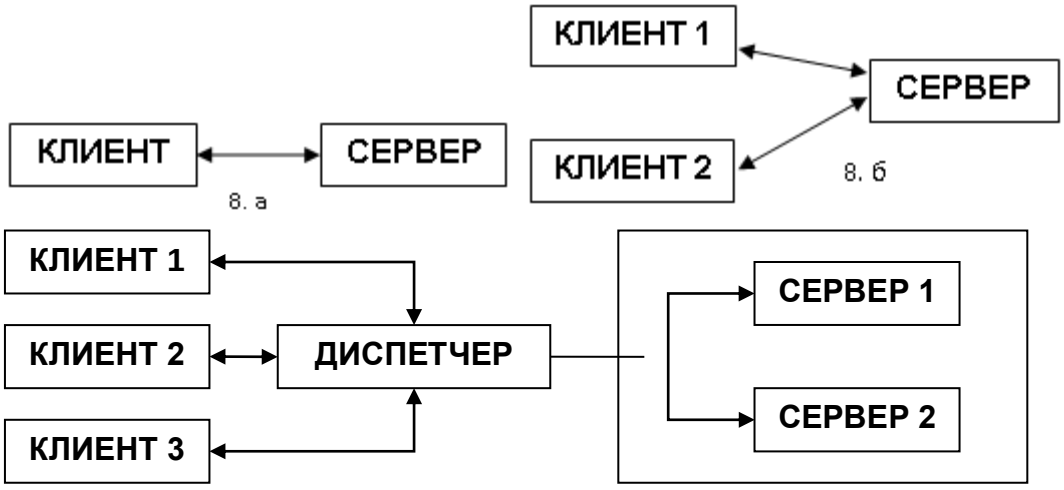


Рисунок 7.7 – (а, б, в). Топології дворівневих систем з архітектурою "клієнт - сервер"

Сучасне рішення проблеми СУБД для мультипроцесорних платформ полягає в можливості запуску декількох серверів бази даних, у тому числі й на різних процесорах. Ця архітектура пов'язана з питаннями распаралелювання виконання одного користувальницького запиту декількома серверними процесами.

3 Сервер додатків. Трирівнева модель

Ця модель є розширенням дворівневої моделі й у ній уводиться додатковий проміжний рівень між клієнтом і сервером. Архітектура трирівневої моделі наведена на рисунку 7.8.

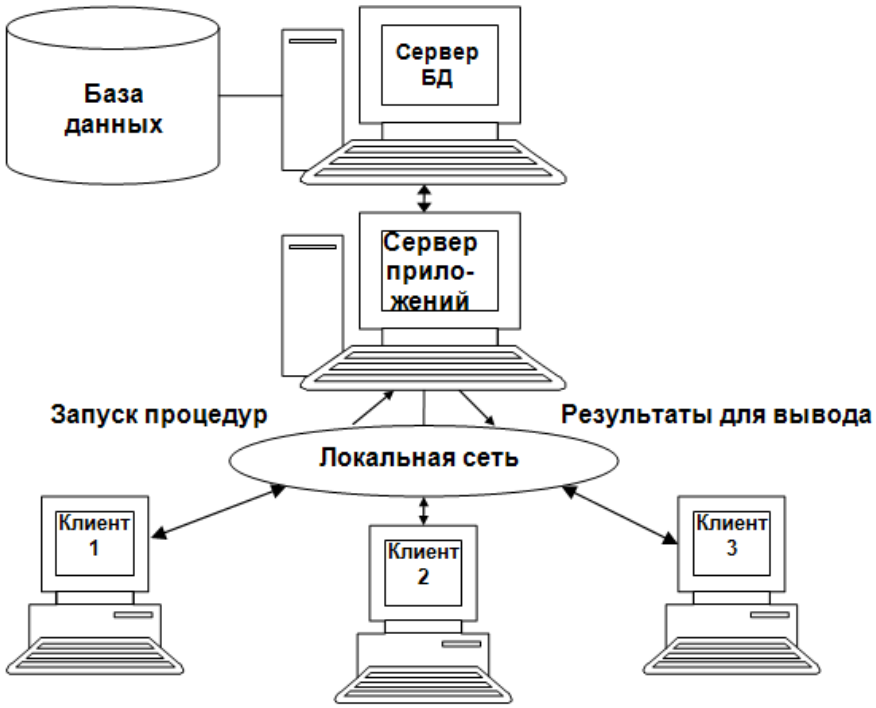


Рисунок 7.8 – Архитектура трирівневої моделі

Така архітектура припускає, що на клієнті розташовуються: функціївведення й відображення даних, включаючи графічний користувальницький інтерфейс, локальні редактори, комунікаційні функції, які забезпечують доступ клієнтові в локальну або глобальну мережу.

Сервери баз даних у цій моделі займаються винятково функціями керування інформаційними ресурсами БД: забезпечують функції створення й ведення БД, підтримують цілісність БД, здійснюють функції створення резервних копій БД і відновлення БД після збоїв, керування виконанням транзакцій і так далі.

Проміжному рівню, що може містити один або скільки серверів додатків, виділяються загальні функції, що завантажують не, для клієнтів: найбільш загальні прикладні функції клієнта, функції, що підтримують мережне доменне

операційне середовище, каталоги з даними, функції, що забезпечують обмін повідомленнями й підтримку запитів.

Ця модель, у якій компоненти додатка діляться між трьома виконавцями, має більшу гнучкість, більш високу переносимість й масштабованість системи, чим дворівневі моделі. Переваги трирівневої моделі найбільш помітні в тих випадках, коли клієнти виконують складні аналітичні розрахунки над базою даних.

Питання та завдання до контролю знань студентів

1) Зобразите схематично трьохрівневу архітектуру бази даних і поясните її основне призначення.
2) Дайте порівняльну характеристику її зовнішньому, концептуальному й внутрішньому рівням.
3) Поясните призначення словника даних.
4) Перелічите 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 віддаленого доступу до даних?
11) Охарактеризуйте модель сервера баз даних.
12) Чим викликана поява трьохрівневих моделей технології "клієнт - сервер"? Які переваги в цих моделей?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 8

з навчальної дисципліни «Бази даних»

Тема Життєвий цикл БД

Мета заняття Ввести поняття життєвого циклу бази даних та дати детальну характеристику основних етапів

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Поняття життєвого циклу бази даних.

2 Етапи життєвого циклу бази даних.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Поняття життєвого циклу бази даних

Як і будь-який програмний продукт, база даних має власний життєвий цикл (ЖЦБД). Головній складовій 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.

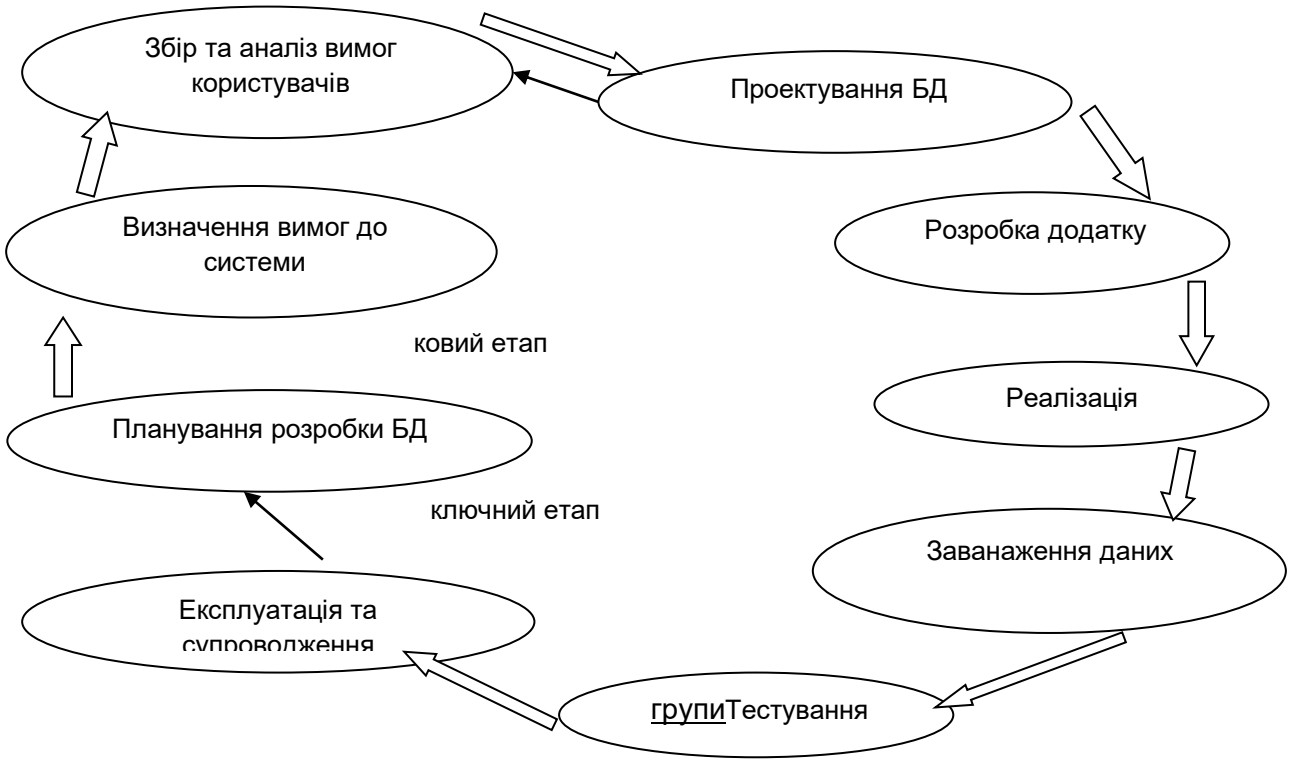


Рисунок 8.1 – Життєвий цикл БД

ЖЦБД містить у собі наступні основні етапи (див. рис. 8.1):

1) Планування розробки бази даних;
2) Визначення вимог до системи;
3) Збір й аналіз вимог користувачів;
4) Проектування бази даних:
   - концептуальне проектування бази даних;
   - логічне проектування бази даних;
   - фізичне проектування бази даних;
5) Розробка додатків:
   - проектування транзакцій;
   - проектування користувальницького інтерфейсу;
6) Реалізація;
7) Завантаження даних;
8) Тестування;

9) 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

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.

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 встаткування 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.

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 генеруєму 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.

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 відносини. Отримані відносини 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 моделям. Семантичне моделювання даних, зв'язане зі значеннєвим 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 міжпрограмні 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.

Розробка додатків

Паралельно із проектуванням системи БД виконується розробка додатків. Головні складові даного процесу - це проектування транзакцій і користувальницького 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 використаються 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 використаються для вибірки деяких даних з метою відображення їх на екрані або приміщення у звіт.

2 Транзакції відновлення використаються 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 користувальницького інтерфейсу.

Користувальницький 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 користувальницького 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 уведення даних;
- зручні засоби переміщення курсору;
- засобу виправлення окремих помилкових символів і цілих полів;
- засобу висновку повідомлень про помилки при уведенні неприпустимих значень;
- особливе виділення необов'язкових для уведення полів;
- засобу висновку пояснювальних повідомлень із описом полів;
- засобу висновку повідомлення про закінчення заповнення форми.

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 користувальницькі подання.

Прикладні програми реалізуються за допомогою мов третього або четвертого поколінь. 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 уведення даних і звіти. Варто враховувати, що

багато існуючі СУБД мають свої власні інструменти, розробки четвертого покоління, що дозволяють швидко створювати додатка за допомогою непроцедурних мов запитів, різноманітних генераторів звітів, генераторів форм, генераторів графічних зображень і генераторів додатків. На цьому етапі реалізуються також використовувані додатком засобу захисту бази даних і підтримки її цілісності.

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 уведення інформації використаються створені в процесі проектування БД зручні спеціальні форми, які дозволять 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.

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 заміняються 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 "потоку" 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 поводження 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.

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 настроювання 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 інформаційній 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.

Питання та завдання до контролю знань студентів

1) Поясните 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?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 9

з навчальної дисципліни «Бази даних»

Тема Концептуальне проектування бази даних: фундаментальні поняття.

Мета заняття Дати характеристику основних понять, що використовуються при побудові концептуальної моделі проектованої бази даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

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 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Фундаментальні поняття семантичної моделі даних

Після завершення початкових етапів ЖЦБД, 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.

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 значеннєвого, 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 проблемі 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.

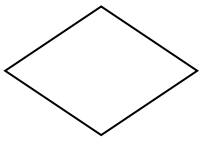
Ми будемо користуватися методологією, скомбінованої з моделі Чена й понять об’єктно-орієнтованого моделювання. Модель Чена формує базис концептуальної моделі даних, а об’єктно-орієнтоване моделювання вносить деякі істотні вдосконалення. Методологія моделювання даних, що представляє комп'ютерне відображення 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 надоб'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:

Об’єкт-тип



Зв’язок-тип

Атрибут

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 моделюємій 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. 9.1).

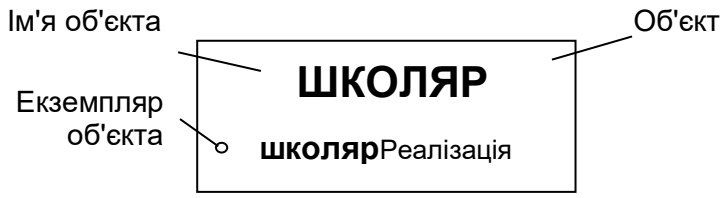


Рисунок 9.1 – Позначення об'єкта на ER-діаграмі

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. 9.2).

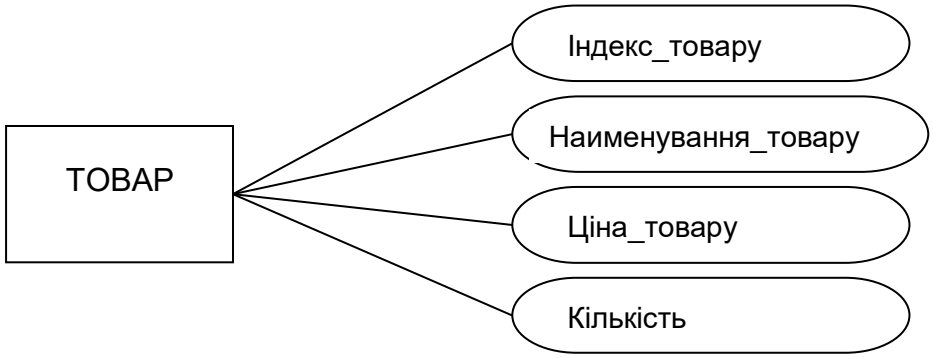


Рисунок 9.2 – Діаграма подання об'єкта ТОВАР і його атрибути

Значення кожного атрибута вибираються з відповідної множини значень, що включає всі потенційні значення, які можуть бути привласнені атрибуту. Цямножина 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 1 000.

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

Об'єкт: СТУДЕНТ (ФИО, 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). Кожний 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. 9.3).

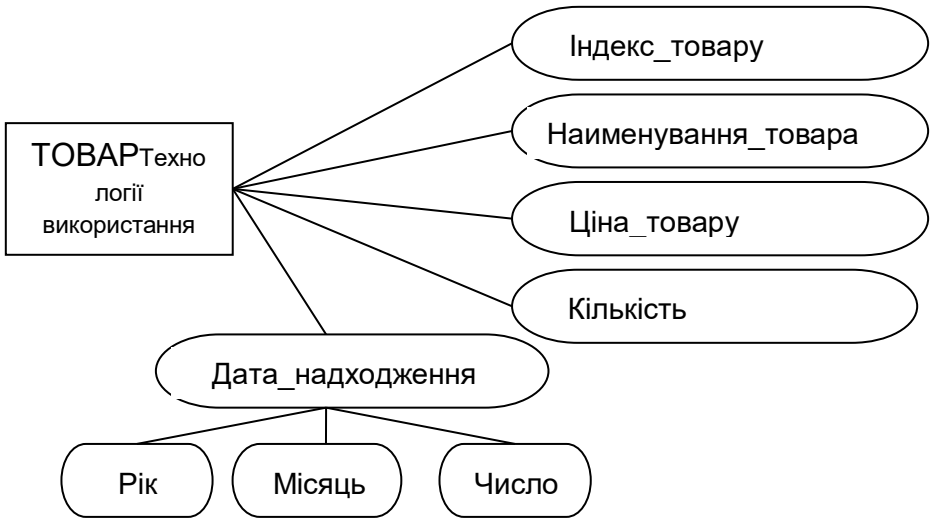


Рисунок 9.3 – Діаграма подання об'єкта ТОВАР зі складеним атрибутом Дата_надходження

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 уважається 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 вибирається 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 (ФИО лікаря, ФИО пацієнта, Дата призначення, Ліки)

однозначно ідентифікувати можна тільки складеним ключем:

(ФИО лікаря, ФИО 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. 9.4).

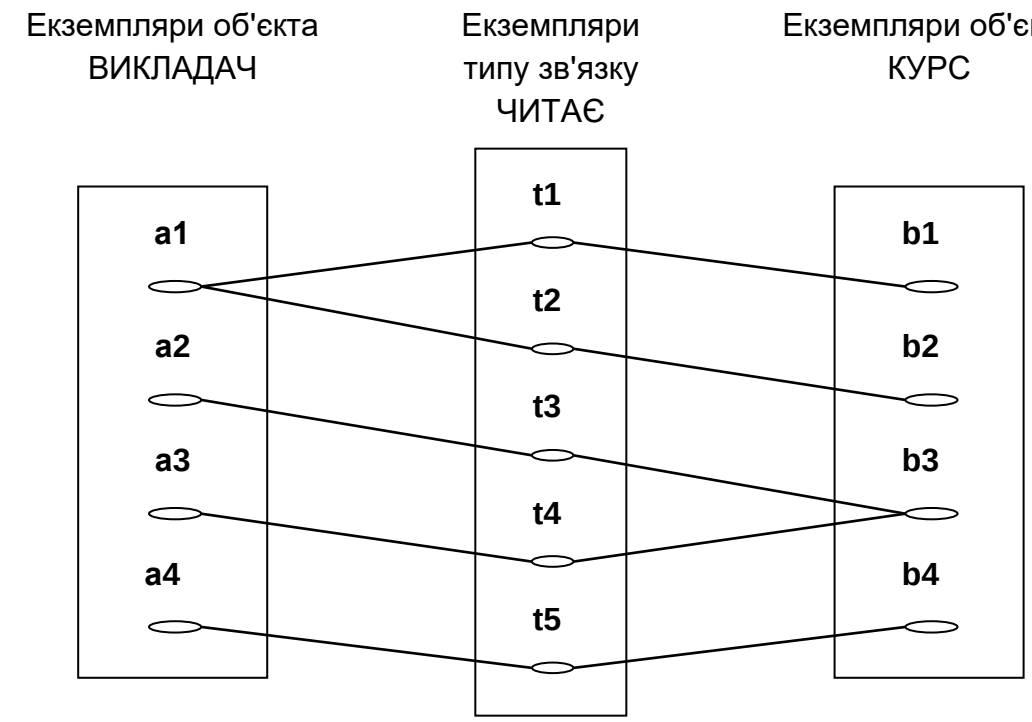


Рисунок 9.4 – Екземпляри типу зв'язку ЧИТАЄ

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. 9.5).

Рисунок 9.5.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:

- тернарі - 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. 9.6).

Рисунок 9.6 – 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 указується 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, використають показник кардинальності. *Показники кардинальності* зв'язків між об'єктами визначаються, насамперед , установленими на виробництві правилами й ставляться до бізнес-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 ⟷ 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 ⟷ СТУДЕНТ.

Концептуальна модель подібного зв'язку наведена на рисунку 9.7.

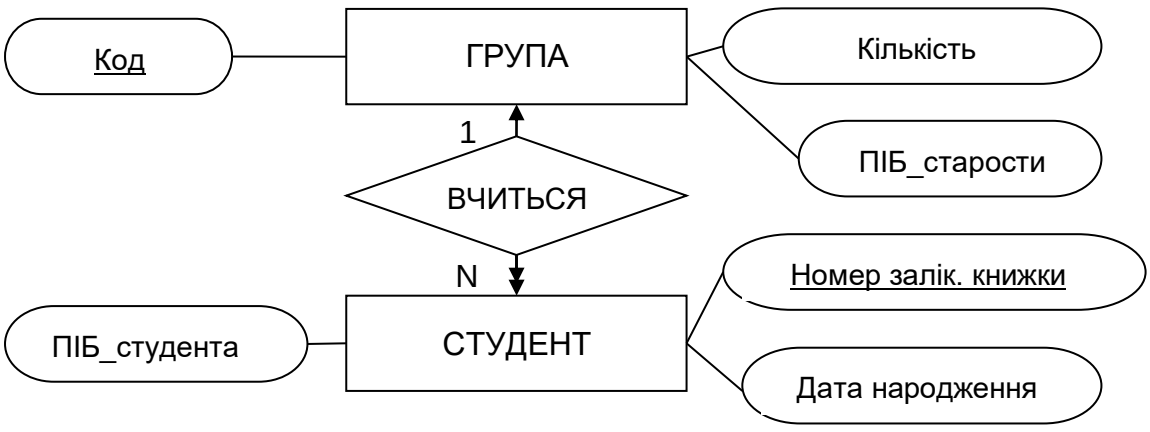


Рисунок 9.7 – Діаграма типу зв'язку ВЧИТЬСЯ із вказівкою показника кардинальності

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 ⟷ ГРУПА

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

Об'єкт:	СТУДЕНТ	ГРУПА
Атрибути:	Номер залікової книжки	Код групи
	ПІБ студента	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 учиться, 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 часткової, 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. 9.8). 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 9.8 – 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. 9.9).

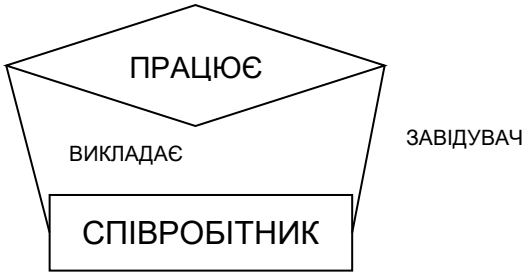


Рисунок 9.9 – Діаграма рекурсивного зв’язку

Питання та завдання до контролю знань студентів

1) Поясните 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?

5) Поясните своїми словами зміст термінів:

– об'єкт;

– атрибут;

– бінарний зв'язок;

– зв'язок "один-до-багатьох";

– потужність;

– показник кардинальності;

– ступінь участі;

– ключ;

– рекурсивний зв'язок;

– складовій об'єкт.

6) Поясните, якою інформацією можна скористатися для визначення наступних конструкцій концептуальної моделі даних:

– об'єктів;

– атрибутів;

– зв'язків.

7) Приведіть приклад концептуальної моделі деякої предметної області.

8) Побудуйте концептуальну модель деякої предметної області, використовуючи поняття складеного об'єкта.

9) Поясните поняття спеціалізації, генералізації, категоризації.

10) 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.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 11

з навчальної дисципліни «Бази даних»

Тема Концептуальна модель даних

Мета заняття Принципи побудови концептуальної моделі проектованої бази даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Моделювання локальної ПрО.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Функції СУБД

Приклад моделювання локальної ПрО. За допомогою розглянутих вище понять можуть бути отримані ER-моделі для більшості схем баз даних в традиційних адміністративно-управлінських додатках. Якщо ПрО велика, то побудова її концептуальної моделі буде протікати успішніше, якщо цю ПрО розбити на кілька локальних предметних областей. Обсяг локальної ПрО вибирається таким чином, щоб в неї входило не більше 6 - 7 об'єктів.

Відправними елементами для побудови ER-моделі локальної ПрО дуже часто є використовувані в організації документи.

Припустимо, що визначена локальна ПрО: поставка товарів на склад. Нехай використовувана форма поставки має вигляд, як на рис. 10.1.

Постачальник			
Адрес постачальника			
Индекс постачальника			
	Поставка		
Дата поставки	Індекс поставки		№ склада
	Товар		
Індекс	Найменування	Ціна	Кількість

Рисунок 10.1 – Форма поставки

Використовуючи наведену форму, можна побудувати концептуальну модель цієї невеликої локальної предметної області. Аналізуєма форма містить наступну інформацію: Постачальник, Індекс постачальника, Адреса постачальника, Товар, Індекс товару, Ціна товару, кількість товару, Поставка, Індекс поставки, Дата поставки та Номер складу.

Виділимо два об'єкти: ПОСТАЧАЛЬНИК і ТОВАР (див. рис. 10.2). Решта атрибути характеризують об'єкт - ПОСТАВКА. Сформуємо його і встановимо певні типи бінарних зв'язків між трьома об'єктами, виходячи з таких міркувань: один і той же постачальник може здійснити ряд поставок, але кожна поставка здійснюється тільки одним постачальником.

Рисунок 10.2 – ПрО ПОСТАВКА

Потужність зв'язку між об'єктами ПОСТАЧАННЯ і ТОВАР повинна бути встановлена M: N, так як кожна поставка може містити кілька товарів, і один і той же товар може міститися в декількох поставках. Виходячи з вищесказаного, діаграма моделі предметної області ПОСТАЧАННЯ прийме такий вигляд, як на рис. 10.3. Атрибути Індекс постачальника, Індекс поставки і Індекс товару були введені для однозначної ідентифікації примірників розглянутих об'єктів, так як жоден з інших атрибутів не підходить на цю роль. Такі ідентифікаційні атрибути називаються первинними ключами. Для роботи проектованої системи може знадобитися також виділення ключів іншого класу. Наприклад, таких, що кожному значенню подібного ключа може задовольняти деякий безліч елементів об'єктного безлічі, а не один. Такі ключі називаються вторинними ключами. При побудові концептуальної моделі слід уникати хати-точності інформації. Після того, як виділені об'єкти, ключі, визначають і видаляють наявні надлишкові зв'язку. Велика увага приділяється аналізу атрибутів. У добре спроектованої БД повинна дотримуватися правило: серед атрибутів об'єкта повинна спостерігатися залежність описового атрибута від ключового, але не повинна існувати залежність між описовими атрибутами.

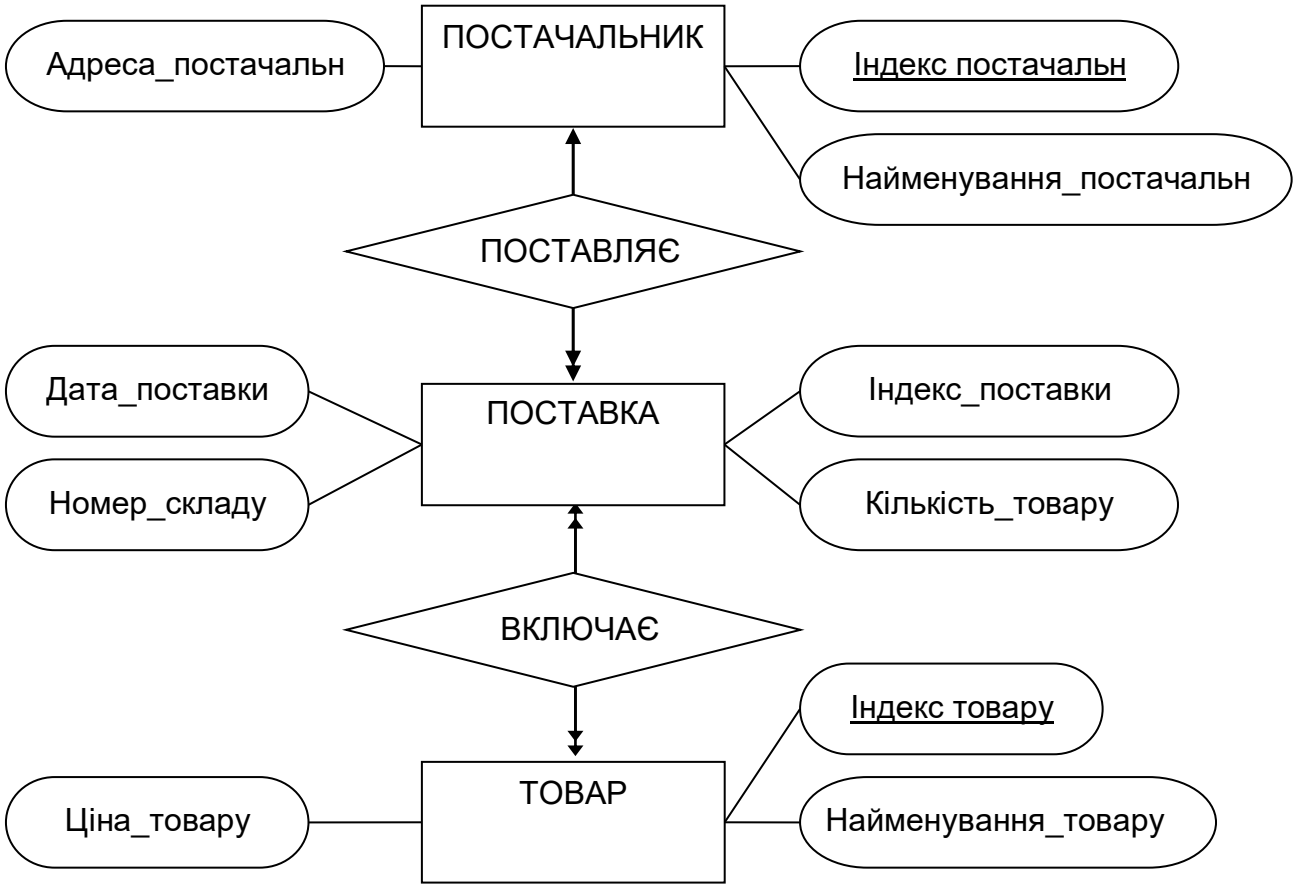


Рисунок 10.3 – Діаграма моделі предметної області ПОСТАЧАННЯ

Завершальним етапом побудови концептуальної моделі досліджуваної ПрО є специфікація всіх об'єктів, що входять в модель. Для даного прикладу результати цього кроку повинні бути зведені до наступного:

1 Специфікація об'єктів:

ПОСТАЧАЛЬНИК:	Індекс постачальника	– ідентифікаційний атрибут
	Адреса_постачальника	– описовий атрибут
	Найменування_постачальника	– описовий атрибут
ПОСТАВКА:	Індекс поставки	– ідентифікаційний атрибут
	Кількість_товару	– описовий атрибут
	Дата_поставки	– описовий атрибут
	Номер_складу	– описовий атрибут
ТОВАР:	Індекс товару	– ідентифікаційний атрибут
	Найменування_товару	– описовий атрибут
	Ціна товару	– описовий атрибут

2 Специфікація типів зв'язків:

Поставляє:	зв’язок	ПОСТАЧАЛЬНИК	⟷	ПОСТАВКА	1:N
Включає:	зв’язок	ПОСТАВКА	⟷	ТОВАР	M:N

3 Специфікація атрибутів:

Індекс постачальника: символьний, 6 символів

Адреса_постачальника: символьний, 50 символів

....

Цеіна_товару: грошовий

Сформовані специфікації заносяться в словник даних. Після створення моделей кожної виділеної ПрО проводиться об'єднання локальних концептуальних моделей в одну загальну.

Спеціалізація і генералізація. Для задоволення нових вимог, висунутих все більш ускладнюються додатками, в семантичне моделювання були введені додаткові концепції, що розширюють його можливості. Така модель отримала назву розширеної ER-моделі (EER-модель). Вона включає всі концепції ER-моделі плюс концепції спеціалізації, генералізації і категоризації. Всі ці нововведення створюють додаткові можливості для моделювання неоднорідних за своєю структурою об'єктів. Додаткові концепції базуються на таких поняттях, як суперклас і підклас, а також використовують процес успадкування атрибутів.

Суперклас - це об'єкт, що включає різні підкласи, які необхідно подати в моделі даних. Підклас - це об'єкт, який є членом суперкласу, але виконує окрему роль в ньому. Суперклас може мати кілька різних підкласів. Так, наприклад, підкласи. АССИСТЕНТ, СТАРШИЙ ВИКЛАДАЧ, ДОЦЕНТ, ПРОФЕСОР є членами суперкласу ВИКЛАДАЧ. Це означає, що кожен екземпляр підкласу є в той же час і екземпляром суперкласу. Зв'язок між суперкласом і підкласом відноситься до типу "один до одного".

Використання понять суперкласу і підкласів дозволяє при моделюванні виділити для підкласу свої власні атрибути і атрибути, успадковані їм від суперкласу. Так, наприклад, підклас ДОЦЕНТ повинен мати ті ж атрибути, що і

всі викладачі - це успадковані їм від суперкласу ВИКЛАДАЧ атрибути. Однак він має і свій власний атрибут, який не визначається для інших категорій викладачів - Номер_діплому_доцента. За відсутності виділених підкласів в цій ситуації довелося б для об'єкта ВИКЛАДАЧ ввести такий атрибут, який би мав невизначене значення для всіх викладачів, які не є доцентами. Підклас може мати свої власні зв'язки, які не підходять для всіх примірників суперкласу. Наприклад, професору дозволяється керувати аспірантами.

На діаграмі (див. рис. 10.4) підкласи з'єднуються лініями з гуртком, який в свою чергу з'єднується з суперкласом. На кожній лінії, що йде від підкласу, розташовується U-образний символ, який позначає напрямок включення. Верхня частина U "відкривається" в сторону суперкласу. Всередині гуртка розташовується буква D, якщо підкласи не перетинаються, і буква O - для пересічних підкласів. В останньому випадку екземпляр суперкласу може бути членом одразу кількох підкласів. Зображена на діаграмі ситуація виключає перетин підкласів, тому в гурток поміщений символ D.

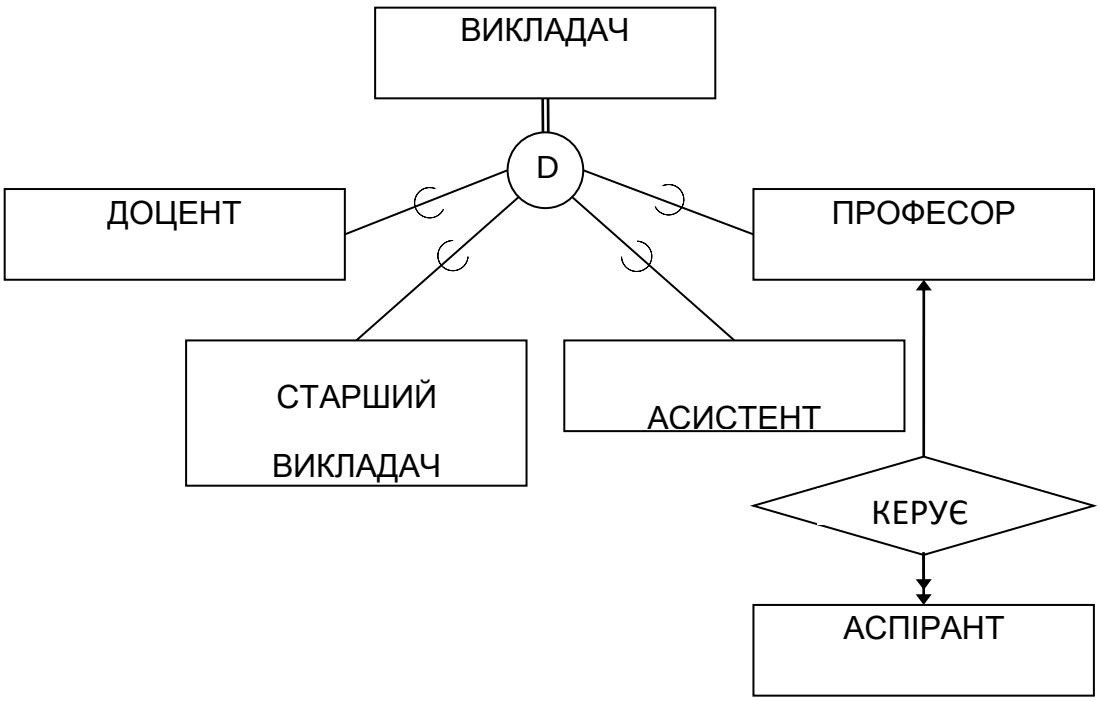


Рисунок 10.4 – Діаграма з використанням понять суперклас і підклас

Введення понять суперкласів і підкласів, дозволяючи уникнути повторного опису атрибутів, економить час проектувальника, підвищує

читабельність, скорочує обсяг невизначених значень. Крім цього використання цих понять збільшує кількість семантичної інформації, що міститься в моделі, дозволяє її зробити більш зрозумілою. У EER-моделі розрізняють два мають протилежні напрямки процесу: це спеціалізація і генералізація.

Спеціалізація є процес збільшення розходжень між від-слушними екземплярами об'єкта за рахунок виділення їх характерних характеристик. Цей процес являє собою спадний підхід до визначення безлічі суперкласів і пов'язаних з ним підкласів. При виявленні набору підкласів об'єкта виконується також виділення специфічних для кожного підкласу атрибутів і їх зв'язків з іншими об'єктами. Специфічні для підкласу атрибути з'єднуються з ним безпосередньо. Спираючись на різні характеристики об'єкта, для одного і того ж об'єкта можна виділити декілька незалежних спеціалізацій. Наприклад, той же об'єкт ВИКЛАДАЧ можна уточняти, розбивши його на два підкласу: ОСНОВНАЯ_РАБОТА і РАБОТА_ПО_СОВМЕСТІТЕЛЬСТВУ. До першого підкласу ставляться ті викладачі, для яких ця робота є основною, до другого - ті, які її поєднують з іншого основною роботою. Це приклад спеціалізації на пересічні підкласи, оскільки один і той же викладач може бути учасником обох підкласів. Така ситуація, наприклад, виникає при наявності вакансій, тобто тоді, коли кількість наявних викладачів менше необхідного. У цій ситуації деякого викладачеві може бути доручено виконання додаткового навантаження, а він сам буде оформлений ще і як сумісник.

Генералізація є висхідний підхід, а саме: процес зведення відмінностей між об'єктами до мінімуму шляхом виділення їх загальних характеристик. Цей підхід дозволяє створити суперклас на основі різних вихідних підкласів. Припустимо, що в наявності є наступні об'єкти: АССИСТЕНТ, СТАРШИЙ ВИКЛАДАЧ, ДОЦЕНТ, ПРОФЕСОР. Застосування методу генералізації до них призведе до виявлення їх загальних властивостей і зв'язків. Спільність їх полягає в тому, що всі вони є викладачами, а значить, можна визначити суперклас ВИКЛАДАЧ, який буде мати всі спільно використовувані атрибути.

На процедуру спеціалізації, крім обмеження перетину, може бути накладено ще одне обмеження, яке називається обмеженням участі, і воно може

бути повним або частковим. При *спеціалізації з повним участю* кожен екземпляр суперкласу повинен бути примірником цієї спеціалізації, тобто належати якомусь виділеному підкласу. Для позначення на діаграмах повної участі між суперкласом і гуртком проводять подвійну лінію. Прикладом спеціалізації з повним участю можуть служити суперклас ВИКЛАДАЧ і його підкласи, де кожен викладач повинен ставитися до однієї із зазначених чотирьох категорій (див. рис. 10.4). *Спеціалізація з частковою участю* означає, що кожен екземпляр об'єкта не обов'язково повинен бути членом підкласу цієї спеціалізації. Для позначення на діаграмах часткової участі між суперкласом і гуртком проводять одинарну лінію. Підкласи успадковують не лише атрибути, а й всі зв'язки суперкласу (див. рис. 10.5)

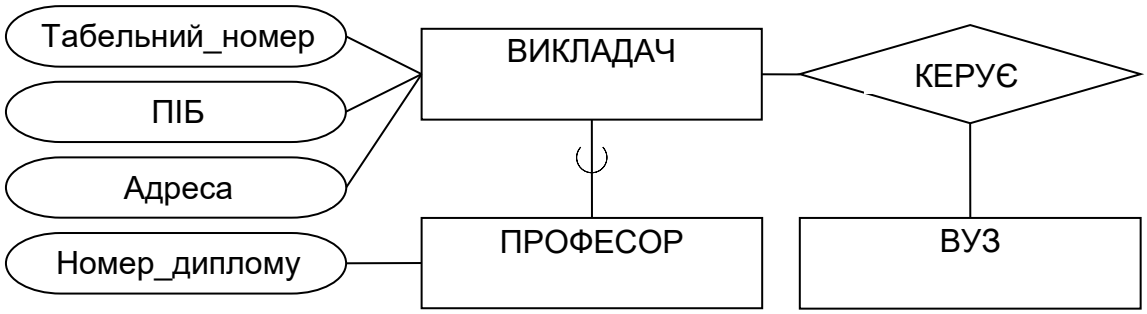


Рисунок 10.5 – Спадкування підкласом зв'язків суперкласу

Категоризація. Ієрархія об'єктів розробляється моделі може бути досить складною. Так, наприклад, можна уявити таку ситуацію, при якій один підклас може бути пов'язаний з декількома суперкласу, які в свою чергу є підкласами одного загального для них суперкласу. У цьому випадку даний підклас називається спільно використовуваних підкласом, так що кожен екземпляр підкласу повинен одночасно є екземпляром кожного пов'язаного з ним суперкласу (див. рис. 10.6).

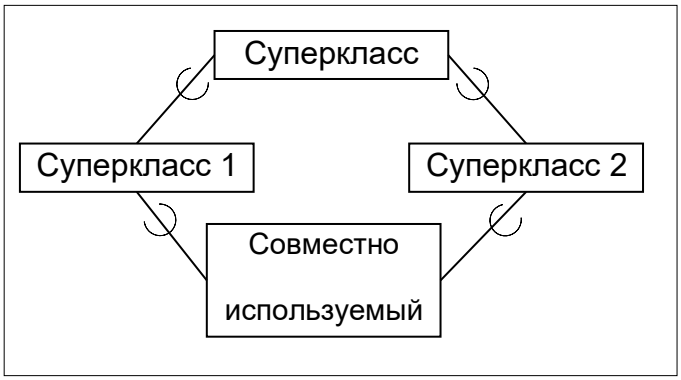


Рисунок 10.6 – Діаграма зі спільно використовуваними підкласами

. Підклас буде категорією, якщо він пов'язаний відразу з декількома суперкласу різних типів (див. рис.10.7).

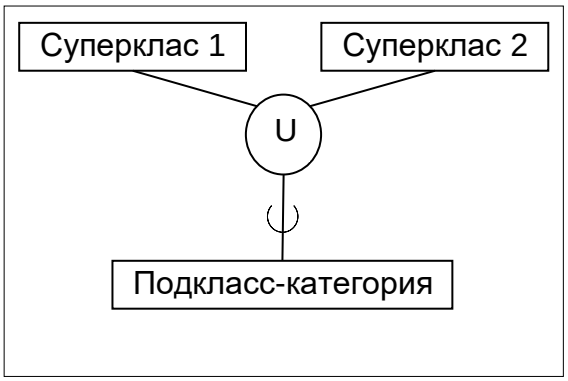


Рисунок 10.7 – Діаграма категоризації

Підклас категорії має вибірковим спадкуванням, що означає успадкування кожним екземпляром категорії атрибутів тільки одного суперкласу. Категорія може бути також двох типів: з повним участю і частковою участю. Найбільш жорстким є обмеження, пов'язане з повним участю, оскільки воно передбачає, що кожен екземпляр всіх суперкласів повинен бути представлений в даній категорії. При частковому ж участю присутність в категорії всіх примірників всіх суперкласів необов'язково. При повному участю від підкласу-категорії йде подвійна лінія до гуртка категоризації, при частковій участі - одинарна.

Приклад. Розглянемо приклад створення EER-моделі предметної області НАВЧАЛЬНИЙ ПРОЦЕС. Виділені в ній об'єкти і зв'язку не охоплюють всі нюанси даної проблеми (постановка завдання не претендує на повноту), але я остаточна діаграма цієї моделі вийде досить складною. Для полегшення сприйняття складних діаграм дозволяється не завдавати атрибути об'єктів (або

позначати тільки ідентифікаційні атрибути), а зв'язки між об'єктами тільки надписувати, опускаючи їх позначення. В результаті збору вимог в даній ПрО і їх аналізу були сформульовані наступні специфікації, які наведені в табл.10.1.

Таблиця 10.1 – Об'єкти і їх атрибути

ОБЄКТ	АТРИБУТ	ПЕРВИННИЙ КЛЮЧ
ФАКУЛЬТЕТ	Код факультета	Код факультета
	Найменування факультету	
КАФЕДРА	Код кафедри	Код кафедри
	Найменування кафедри	
ЗАГАЛЬНА ВЫПУСКНА		
СПІВРОБІТНИК	Табельный номер	Табельний номер
	ПІБ	
	Стаж	
ІНЖ_ЛАБОР_СКЛАД	Посада	
ВИКЛАДАЧ	Педагогічний стаж	
ЛЕКТОР		
АСИСТЕНТ		
СТАРШИЙ ВИКЛАДАЧ		
ДОЦЕНТ	Диплом доцента	
ПРОФЕСОР	Диплом професора	
СПЕЦІАЛЬНІСТЬ	Шифр спеціальності	Шифр спеціальності
	Найменування спеціальності	
КУРС	Номер курсу	Номер курсу
НАВЧАЛЬНИЙ ПЛАН	Шифр спеціальності	
ДИСЦИПЛІНА	Код дисциплини	Код дисципліни
	Найменування дисципліни	
ЛЕКЦИИ	Кількість годин	
ПРОЧИЕ		
ГРУППА	Код групи	Код групи
	Староста	
	Кількість студентів	
СТУДЕНТ	Номер залікової книжки	Номер зачетной книжки
	ПІБ	
	Адреса	
ДИПЛОМНИК	Тема диплому	

– факультет об'єднує ряд кафедр. Серед кафедр є загальноосвітні кафедри та кафедри, що випускають фахівців. Кафедра може готувати фахівців за кількома спеціальностями. Особи, що працюють на кафедрі, відносяться до категорії СПІВРОБІТНИК.

– всі співробітники діляться на дві частини: обслуговуючий персонал - інженерні-нерно-лаборантський склад і викладацький склад. Кожен викладач повинен ставитися до однієї з чотирьох категорій: асистент, старший викладач, доцент, професор, причому всі категорії, крім категорії асистент, входять в лекторський склад. Представникам лекторської складу доручається читання лекцій і керівництво дипломним проектуванням.

– викладання студентам кожної спеціальності ведеться відповідно до затвердженого навчального плану, де визначені дисципліни, що підлягають изуче-нию, види занять, що відводиться під них кількість годин, а також визначена форма контролю знань з цієї дисципліни: залік або іспит.

– за даною спеціальністю навчання студентів ведеться на одному або більше курсів. На кожному курсі може бути одна або кілька груп однієї і тієї ж спеціальності, де навчається ряд студентів.

Для спрощення концептуальної моделі бази даних залишилися не пропрацювали такі моменти навчального процесу, як: хто і де веде певний вид занять. З процесу навчання з даної дисципліни виділені тільки лекції, а всі інші види занять, а також форми контролю віднесені до об'єкта ІНШІ.

Об'єкти, виділені в даній моделі, а також належні до них атрибути, які виявлені з специфікацій вимог користувачів, наведені в таблиці. Вивчення складу атрибутів кожного об'єкта дозволило виділити ті з них, які можна використовувати в якості первинних ключів. Вони також поміщені в таблицю 1. Неважко помітити, що не всі позиції в колонці Первинний ключ заповнені. Вільними залишилися ті позиції, які відносяться до об'єктів-підкласів, отриманим шляхом спеціалізації. Оскільки підклас успадковує всі атрибути суперкласу, а отже, і ідентифікаційні атрибути і кожен екземпляр підкласу пов'язаний тільки з одним екземпляром суперкласу, то і ідентифікаційні атрибути у них одні й ті ж. Для виділених об'єктів необхідно встановити

найважливіші типи зв'язків, що відповідають тим, що існують в реальному світі. Для кожної з цим слід також визначити показник кардинальності і ступінь участі сторін в зв'язку. Все зв'язку та їх характеристики наведені в таблиці 10.2.

Таблиця 10.2 – Типи зв’язків та їх характеристики

Зв’язок	Об’єкти	Показник кардинальності	Ступінь участі
ВХОДИТЬ	ФАКУЛЬТЕТ	1:N	Повна
	КАФЕДРА		Повна
ПРАЦЮЄ	КАФЕДРА	1:N	Повна
	СПІВРОБІТНИК		Часткова
ГОТОВИТЬ	ВЫПУСКНА СПЕЦІАЛЬНІСТЬ	1:N	Повна
			Повна
ЗАТВЕРДЖЕН	СПЕЦІАЛЬНІСТЬ	1:1	Повна
	НАВЧАЛЬНИЙ_ПЛАН		Повна
МАЄ	СПЕЦІАЛЬНІСТЬ	1:N	Повна
	КУРС		Повна
СКЛАДАЄТЬСЯ	КУРС	1:N	Повна
	ГРУПА		Повна
ВЕДЕ	ВИКЛАДАЧ	N:M	Повна
	ДИСЦИПЛІНА		Повна
ВКЛЮЧАЄ	НАВЧАЛЬНИЙ_ПЛАН	1:N	Повна
	ДИСЦИПЛІНА		Повна
ВЧИТЬСЯ_В	ГРУПА	1:N	Повна
	СТУДЕНТ		Повна
ВЧИТЬСЯ_ПО	ДИСЦИПЛІНА	N:M	Повна
	СТУДЕНТ		Повна
ЧИТАЄ_ПО	ЛЕКТОР	1:N	Часткова
	ЛЕКЦІЇ		Повна
КЕРУЄ_ДИПЛОМНИКОМ	ЛЕКТОР	1:N	Часткова
	ДИПЛОМНИК		Повна
ГОТОВИТЬ_ДО_ЗАХИСТУ	СПЕЦІАЛЬНІСТЬ	1:N	Часткова
	ДИПЛОМНИК		Повна

Питання та завдання до контролю знань студентів

1) Приведіть приклад концептуальної моделі деякої предметної області.

2) Поясніть поняття спеціалізації, генералізації, категоризації.

3) Створіть концептуальну модель даних деякого университету, яка би дозволяла отримувати наступну інформацію:

– кількість викладачів, які працюють на факультеті;

– список викладачів факультету;

– дисципліни, які вивчають студенти певної спеціальності в шостому семестрі;

– об’єм лекцій за дисциплінами другого семестру для певної спеціальності факультету;

– прізвища викладачів, які викладають вищу математику на факультеті.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 11

з навчальної дисципліни «Бази даних»

Тема Класифікація моделей даних

Мета заняття Провести огляд моделей даних. Дати класифікацію моделей даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Моделі даних.

2 Класифікація моделей даних.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Моделі даних

Відповідно до концепції баз даних 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. Для того щоб з набору конкретних значень можна було б

витягати корисну для користувача інформацію, необхідно визначити структуру використовуваних даних або модель даних.

Модель даних - це деяка абстракція, у якій відбиваються найважливіші аспекти функціонування виділеної предметної області, а другорядні - ігноруються. Модель даних містить у собі набір понять для опису даних, зв'язків між ними й обмежень, що накладають на дані. У моделі даних розрізняють три головні складові:

– структурну частину, що визначає правила породження припустимих для даної СУБД видів структур даних;

– керуючу частину, що визначає можливі операції над такими структурами;

– класи обмежень цілісності даних, які можуть бути реалізовані засобами цієї системи.

Кожна СУБД підтримує ту або іншу модель даних. Поняття моделі даних узвичаїлося фахівців в області БД тільки разом з появою реляційного підходу. Всі ранні системи не ґрунтувалися на яких-небудь абстрактних моделях. Абстрактні подання ранніх систем з'явилися пізніше на основі аналізу й виявлення загальних ознак у різних систем. Ранні (дореляційні) СУБД активно використалися протягом багатьох лет. У свій час у подібні системи багато компаній інвестували більші засоби. Вони застосовувалися довше, ніж використається яка-небудь із реляційних СУБД, а деякі з ранніх систем використаються навіть у наш час. За час їхнього існування накопичені величезні бази даних, і однієї з актуальних проблем інформаційних систем є використання цих систем разом із сучасними системами.

Власне кажучи модель даних, підтримувана механізмами СУБД, повністю визначає безліч конкретних баз даних, які можуть бути створені засобами цієї системи, а також способи модифікації стану БД із метою відображення тих змін, які відбуваються в предметній області.

2 Класифікація моделей даних

У цей час описано багато різноманітних моделей, побудова яких переслідує різні цілі. З безлічі опублікованих моделей даних можна виділити три категорії:

- об'єктні моделі даних;
- моделі даних на основі записів;
- фізичні моделі даних.

Стосовно до трьохрівневої архітектури баз даних слід зазначити, що перші дві категорії використаються для опису даних на зовнішньому й концептуальному рівнях, а остання категорія - на внутрішньому рівні.

Об'єктні моделі даних. Серед об'єктних моделей варто виділити ER-модель, що найбільше часто використається в методології проектування баз даних, а також об’єктно-орієнтовну модель, останнім часом широко використовувану в технологіях баз даних. Об’єктно-орієнтовна модель розширює поняття об'єкта, включаючи в нього не тільки атрибути, що характеризують стан об'єкта, але й пов'язані з ним дії.

Моделі даних на основі записів. У моделі даних на основі записів база даних складається з декількох записів фіксованого формату, які можуть мати різні типи. У більшості комерційних СУБД використаються два типи такого роду моделей даних: теоретико-графові (ТГ) і теоретико-множинні (ТМ) моделі даних.

До теоретико-графових моделей ставляться два різновиди:

- мережні моделі;
- ієрархічні моделі.

У таких моделях даних передбачаються характерні для подібного роду структур операції навігації й маніпулювання даними. Принципове значення при цьому має та обставина, що передбачається одночасна обробка тільки одиночних об'єктів даних із БД - записів, сегментів або полів записів. Інтерактивний доступ до БД підтримується тільки шляхом створення відповідних прикладних програм із власним інтерфейсом. Користувальницькі додатки цих систем, використовуючи мови програмування, розширені

функціями СУБД, здійснюють явну навігацію в БД. У зв'язку із цим системи, побудовані на цих моделях даних, називають навігаційними. Апарат навігації в ТГ-моделях служить для установки тих об'єктів даних, до яких буде застосовуватися чергова операція маніпулювання даними. Такі об'єкти називаються поточними. Механізми доступу до даних і навігації за структурою даних у таких моделях досить складні, особливо в мережній моделі, і опираються на концепцію поточного стану механізму доступу. Навігаційна природа ранніх систем і доступ до даних на рівні записів змушували користувача самого робити всю оптимізацію доступу до БД, без якої-небудь підтримки системи.

Теоретико-множинні моделі використають математичний апарат, реляційну алгебру (знакова обробка множин), реляційне счислення. До моделей даного типу ставляться реляційні моделі. Відповідно до реляційної моделі даних БД представляється у вигляді сукупності таблиць, над якими можуть виконуватися операції, формовані в термінах реляційної алгебри й реляційного счислення. Той факт, що в цій моделі даних, на відміну від ТГ-моделей, операції над об'єктами БД мають теоретико-множинний характер, дає можливість формулювати запити більш компактно, у термінах більших агрегатів даних. Однак такий підхід породжує складні проблеми, пов'язані із забезпеченням досить високого рівня продуктивності СУБД цього класу, з наявністю інтерфейсу СУБД, що підтримує реляційну модель даних, із традиційними мовами програмування.

Фізична модель даних. Фізичні моделі даних описують те, як дані зберігаються в комп'ютері, представляючи інформацію про структуру записів, їхньої впорядкованості й існуючих шляхів доступу. Найпоширеніші з них наступні: узагальнююча модель і модель пам'яті кадрів.

Питання та завдання до контролю знань студентів

1) Що собою представляє модель даних? Які складові в ній розрізняють?
2) Приведіть класифікацію моделей даних.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 12

з навчальної дисципліни «Бази даних»

Тема Мережна модель даних

Мета заняття Розглянути структуру даних в мережевій моделі даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Мережна модель даних

2 Структури даних мережної моделі.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Мережна модель даних

Мережі - природний спосіб подання відносин між об'єктами, усіляких їхніх взаємозв'язків. Мережна модель опирається на математичну структуру, що називається спрямованим графом. Спрямований граф складається з вузлів, з'єднаних ребрами. У контексті моделей даних вузли являють собою об’єкти у вигляді типів записів даних, а ребра - зв'язку між об'єктами зі ступенем

кардинальності "один до одному" або "один до багатьох". Таким чином, мережна БД складається з набору екземплярів кожного типу із заданого в схемі БД набору типів записів і набору екземплярів кожного типу із заданого набору типів зв'язку.

Основна відмінність графових форм подання даних у мережній структурі даних від ієрархічної структури даних полягає в тому, що нащадок у графі може мати будь-яке число предків. Таким чином, мережна модель даних, знімаючи основні обмеження ієрархічної структури, що вже використається на той час, даних, є результатом дій по узагальненню ситуацій моделювання існуючих реалій і спроб установлення докладних стандартів у цій області.

Хоча знайомство з теоретико-графовими моделями почато з розгляду мережної моделі даних, варто підкреслити, що їй передувала в часі ієрархічна модель даних. Першої представляється мережна модель даних тому, що, ієрархічна модель даних є часткою случаємо мережний.

2 Структури даних мережної моделі.

У технології, розробленої CODASYL, використаються кілька різних типових структур даних, головними з яких є типи записів і набори. Для побудови цих структур застосовуються такі конструктивні елементи, як елемент даних й агрегат (див. рис. 12.1).

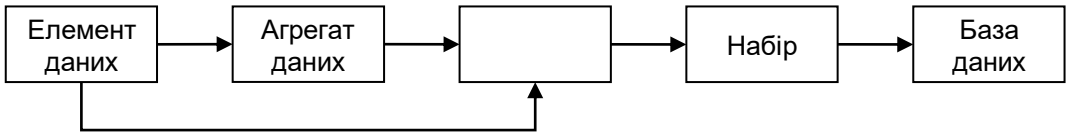


Рисунок 12.1 – Основні структури мережної моделі даних

Структуризація даних базується на концепціях агрегації й узагальнення. Агрегація використається для композиції елементів даних у запис. Узагальнення використається для об'єднання однотипних записів файлів. Основні елементи мережної моделі даних:

1) Елемент даних - це найменша пойменована інформаційна одиниця даних, доступна користувачеві. Елемент даних повинен мати свій тип (не структурний, простій).

2) Агрегат даних відповідає наступному рівню узагальнення - пойменована сукупність елементів даних усередині запису або іншого агрегату (див. рис. 12.2).

Дата		
День	Місяць	Рік

Рисунок 12.2 – Агрегат Дата

3) Запис - кінцевий рівень агрегації. Кожен запис являє собою іменовану структуру, що містить один або більше іменованих елементів даних, кожний з яких має свій особливий формат. Наприклад, агрегат даних Дата входить до складу запису Співробітник (див. рис. 12.3).

<table>
<tr><td colspan="6">Співробітник</td></tr>
<tr><td rowspan="2">Табельний номер</td><td rowspan="2">ПІБ</td><td colspan="3">Дата</td><td rowspan="2">Адреса</td></tr>
<tr><td>День</td><td>Місяць</td><td>Рік</td></tr>
</table>

Рисунок 12.3 – Запис Співробітник

4) Тип записів - це сукупність логічно зв'язаних екземплярів записів. Тип записів моделює деякий клас об'єктів реального миру. Як елемент даних можуть бути використані тільки прості типи, а як агрегати можуть бути використані складні типи: вектор і повторювана група.

Агрегат типу вектор відповідає лінійному набору елементів даних (див. рис. 12.2). Агрегат типу повторювана група відповідає сукупності векторів даних. Так, наприклад, у замовленні може бути зазначено кілька видів товарів із числом повторень 10 (див. рис. 12.4).

<table>
<tr><td colspan="6">Заказ</td></tr>
<tr><td rowspan="2">Номер заказу</td><td colspan="3">Дата заказу</td><td colspan="3">Товар</td></tr>
<tr><td>День</td><td>Місяць</td><td>Рік</td><td>Шифр товару</td><td>Колькість товару</td><td>Найменування товару</td></tr>
<tr><td colspan="3"></td><td colspan="3">Група, що повторюється</td></tr>
</table>

Рисунок 12.4 – Запис Замовлення

5) Набір - це пойменована дворівнева ієрархічна структура, що містить запис власника й запису членів. Набори виражають зв'язку "один до багатьох" або "один до одного" між двома типами записів. Тип набору підтримує роботу із внутрішніми структурами типів записів. Тип набору - це не аналог файлу, він визначає зв'язок між типами записів. Кожен екземпляр типу набору може містити тільки один екземпляр запису власника й скільки завгодно екземплярів записів членів. Набір може бути порожнім, тобто в ньому може перебувати тільки один запис-власник. Тип набору є основним композиційним елементом, за допомогою якого за технологією CODASYL будується в мережній моделі структура всієї БД. Набір, наведений на рисунку 12.5, визначає тип запису - власника Відділ і тип запису-члена набору Співробітник, а також тип зв'язку між ними "один до багатьох" - з ім'ям Працює.

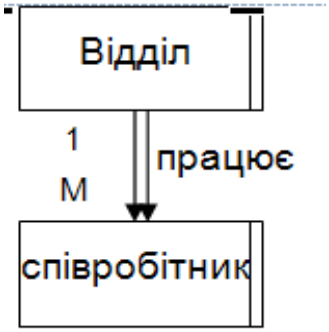


Рисунок 12.5 – Діаграма типу набору Працює

Ім'я набору - це мітка, привласнена стрілці. Зв'язок типу "один до багатьох" допускає можливість того, що з даним екземпляром запису-власника може бути зв'язаний нуль, один, або кілька екземплярів запису-члена. Використовуючи поняття мережної моделі даних, можна одержати інше зображення такого набору (див. рис. 12.6), де представлені логічні типи записів Відділ і Співробітник, їхня структура й зв'язок між типами записів Працює. Для позначення зв'язку в напрямку "багато" можна використати альтернативне позначення показника кардинальності - подвійні стрілки, опускаючи буквене.

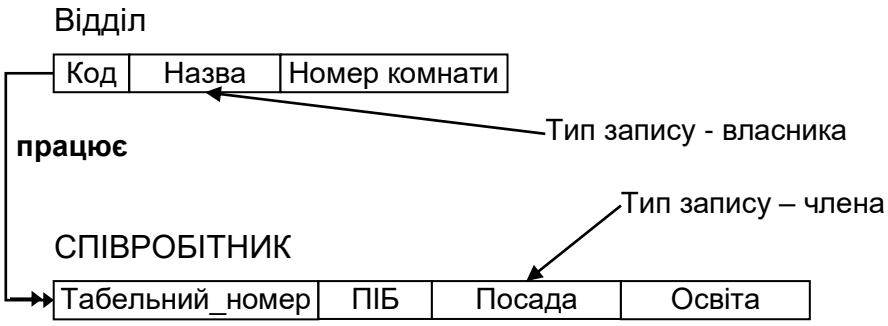


Рисунок 12.6 – Набір Працює між двома типами записів Відділ і Співробітник

Типи записів і набори використаються для побудови на концептуальному рівні мережного графа, що відбиває глобальну структуру бази даних. Отже база даних у мережній моделі даних - це пойменована сукупність екземплярів записів різного типу й екземплярів наборів, що містять зв'язки між ними.

Мережний граф БД. Мережний граф будується за певними правилами:

– БД може містити будь-яка кількість наборів і записів;

– між двома типами записів може бути будь-яка кількість наборів; наприклад, викладач може не тільки займатися із групою, але й бути куратором цієї групи (див. рис. 12.7);

Рисунок 12.7 – Набори між двома типами записів

– тип запису може бути власником в одних типах наборів і членом в інших типах наборів;

– тільки один тип запису може бути власником у кожному наборі;

– типи наборів можуть бути визначені так, що в результаті вони утворять циклічну структуру;

– тип запису не обов'язково повинен бути членом якого-небудь типу набору;

– той самий тип запису може бути власником декількох типів наборів й одночасно може бути членом декількох типів наборів.

Питання та завдання до контролю знань студентів

1) Опишіть мережну модель даних, її структурні елементи.

2) За якими правилами будується мережний граф?

3) Як здійснюється реалізація наборів у мережній моделі даних?

4) Які обмеження цілісності збережених даних повинні підтримуватися в мережній моделі даних?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 13

з навчальної дисципліни «Бази даних»

Тема Перетворення концептуальної моделі в мережеву

Мета заняття Принципи перетворення концептуальної моделі в мережеву

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Перетворення концептуальної моделі в мережну.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Перетворення концептуальної моделі в мережну.

Мережна модель даних може бути без ускладнень отримана з концептуальної моделі. Для цього треба припустити, що в останній використаються тільки бінарні зв'язки. Причому вони повинні належати до типів "один до одному" або "один до багатьох". При цьому замість об'єктів концептуальної моделі необхідно використати типи записів мережної моделі, де імена об'єктів стають іменами типів записів, атрибути об'єктів стають полями записів, зв'язок між об'єктами перетворюється у зв'язок між типами записів.

Бінарні зв'язки, що належать до типу "один до багатьох", переносяться в мережну модель у такий спосіб: тип запису з боку "один" стає власником, а тип запису з боку "багато" стає типом запису-члена. Для зв'язку типу "один до одному" вибір типу запису-власника й типу запису-члена може бути здійснений довільно.

Властивим мережним моделям внутрішні обмеження не дозволяють прямо моделювати деякі реально існуючі в предметній області типи зв'язків. До таких типів зв'язків ставляться рекурсивні зв'язки й зв'язки типу M:N. Для того щоб їх відбити в мережній моделі, застосовують різні підходи, які приводять до певних перетворень графа. Рекурсивний зв'язок виникає тоді, коли екземпляр типу запису бере участь у зв'язку з іншим екземпляром того ж типу запису, наприклад один зі співробітників підрозділу є його керівником. Рекурсивні зв'язки можуть мати тип 1:1, 1:M, M:N.

Оскільки в мережній моделі ситуація, при якій той самий тип запису є власником і членом того самого набору, неприпустима, то для моделювання в ній рекурсивних зв'язків прибігають до деяких її перетворень. Один з можливих варіантів подання рекурсивного зв'язку зображений на рисунку 9, де використаний сполучний тип запису, що не містить ніяких елементів і за допомогою якого визначені два типи набору. Оскільки в конкретному екземплярі набору екземпляр запису-члена набору не може мати більше одного екземпляра власника набору, то в мережній моделі не існує прямого способу подання зв'язку типу M:N. І в цьому випадку також необхідно щось змінити, щоб можна було отриману структуру вписати в ті рамки, які визначає мережна модель. Проведені в таких випадках перетворення приводять до збільшення кількості типів записів, тому що в подібних моделях зв'язок повинна бути представлена самостійним типом запису. У результаті замість однієї неприпустимої для мережної моделі зв'язку типу M:N з'являться два зв'язки типу 1:N, за допомогою яких можна буде легко управляти типами наборів. Графічні діаграми, наведені на рисунках 13.1 та 13.2, ілюструють ці перетворення.

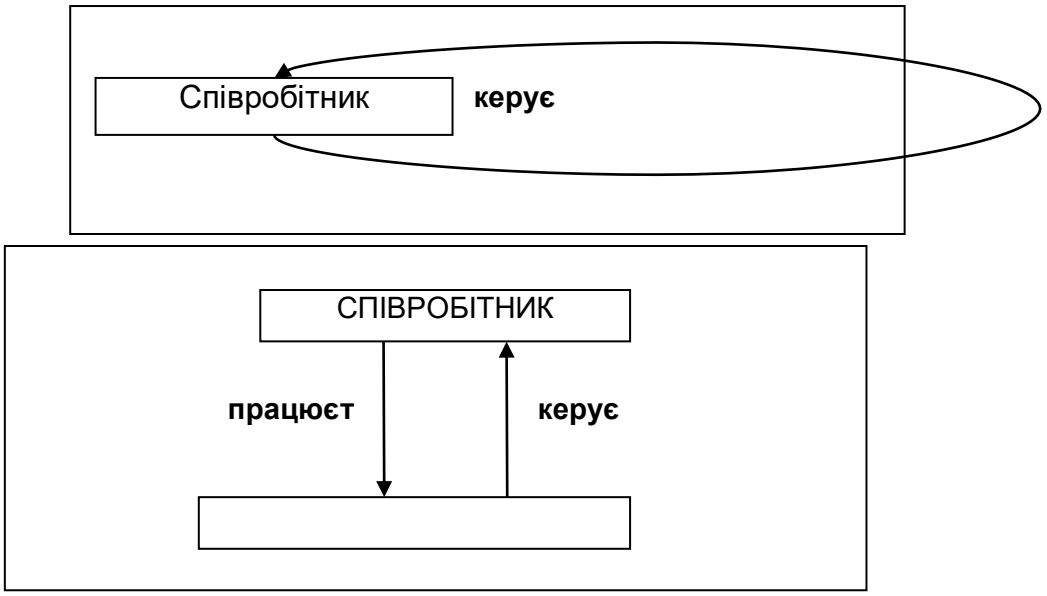


Рисунок 13.1 – Рекурсивний зв'язок Співробітник — Керівник

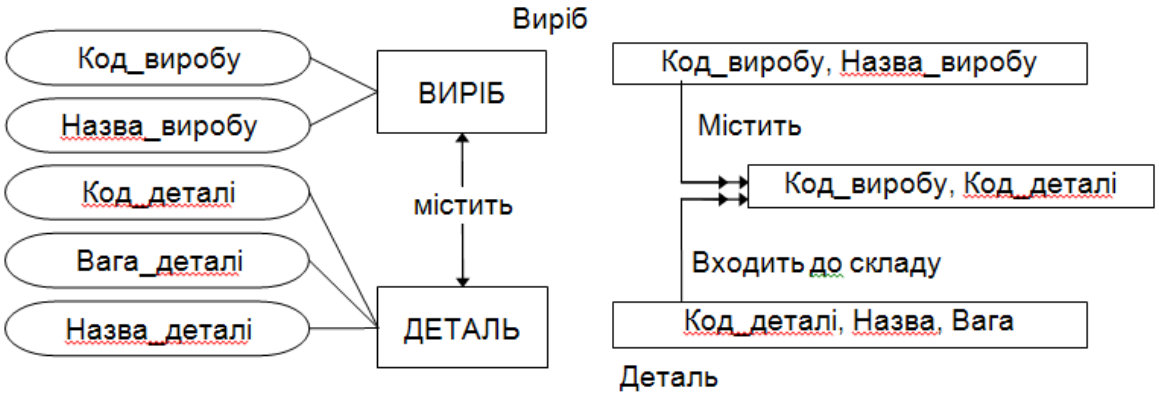


Рисунок 13.2 – Діаграма подання зв'язку типу M:N

На відміну від простих мереж складні мережі, що включають одну або кілька зв'язків типу "багато до багатьох", мають потребу в додаткових перетвореннях, що здійснюють заміну зв'язку "багато до багатьох" на більше прості дозволені зв'язки. Підводячи підсумки проблемам перетворення концептуальної моделі в мережну модель даних, варто звернути увагу на наступний факт. Деякі концептуальні моделі містять не тільки бінарні зв'язки, але й зв'язку більше високих порядків. Перетворення таких зв'язків у мережну модель даних вимагає створення нового типу запису - запису зв'язку, що складає із ключових полів кожного об'єкта. Причому тип зв'язку між існуючими записами й записом зв'язку завжди має показник кардинальності "один до багатьох" і запис зв'язку завжди є записом-членом набору.

Питання та завдання до контролю знань студентів

1 Поясните методику перетворення концептуальної моделі в мережну.

2 Як здійснюється реалізація наборів у мережній моделі даних?

3 Які обмеження цілісності збережених даних повинні підтримуватися в мережній моделі даних?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 14

з навчальної дисципліни «Бази даних»

Тема Ієрархічна модель даних

Мета заняття Розглянути структуру даних в ієрархічній моделі даних них компонентів середовища СУБД. Дати оглядову характеристику мов БД

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Ієрархічна модель даних.

2 Структурна частина ієрархічної моделі.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Ієрархічна модель даних

Перші системи керування базами даних використали ієрархічну модель даних. Для ієрархічної моделі дані стандарти не розроблялися, і її вивчення може бути здійснене шляхом розгляду реально існуючих СУБД. Найвідомішої СУБД, що використає модель даних цього типу, є система фірми IBM - Information Management System (IMS). Саме ця система зареєстрована як перша промислова СУБД. Вона була створена для підтримки місячного проекту

"Аполлон" і покликаний була управляти величезною кількістю деталей, ієрархічно зв'язаним між собою - з деталей збиралися вузли, які входили в ще більш великі модулі й так далі. Структур даних з такою організацією існує досить багато, а системи з ієрархічною моделлю даних працюють досить ефективно й простіше в реалізації, чим мережні.

2 Структурна частина ієрархічної моделі.

Основними інформаційними одиницями в ієрархічній моделі даних є сегмент і поле.

Поле даних визначається як найменша неподільна одиниця даних, доступна користувачеві.

Для сегмента визначаються тип сегмента й екземпляр сегмента. Екземпляр сегмента утвориться з конкретних значень полів даних. Тип сегмента - це пойменована сукупність вхідних у нього типів полів даних.

Ієрархічна модель даних базується на графовій формі побудови даних, і на концептуальному рівні вона є просто часткою случаємо мережної моделі даних. Між конструктивними елементами цих моделей існують певні аналогії. У мережній моделі вершини графів відображають типи записів, дуги - типи зв'язків. В ієрархічній моделі даних вершині графа відповідає тип сегмента або просто сегмент, а дугам - типи зв'язків предок - нащадок. В ієрархічних структурах сегмент - нащадок повинен мати в точності одного предка. У СУБД, що базуються на ієрархічній моделі даних, використаються такі структури й способи їхньої реалізації, які дозволяють віднести системи такого типу, також як і мережні, до класу "навігаційних".

Ієрархічна модель являє собою зв'язного неорієнтованого графа деревоподібної структури, що поєднує сегменти. Ієрархічна БД складається з упорядкованого набору дерев; більш точно, з упорядкованого набору декількох екземплярів одного типу дерева, а не довільних графів. При цьому граф - дерево має наступні властивості (див. рис. 14.1):

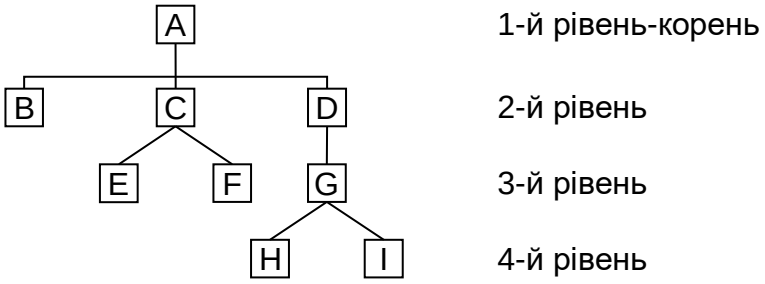


Рисунок 14.1 – Ієрархічна модель даних

- є тільки одна вершина графа - корінь, у яку не заходить жодне ребро;
- є вершини графа n-го рівня, називані вихідними, куди заходить одне ребро (n-1) рівня; виходить із цього вузла нуль, один або кілька породжених вузлів, які називаються нащадками;
- єдиний прохід до породженого вузла лежить через його вихідний вузол;
- кожен нащадок може мати тільки одного предка;
- немає замкнутих петель або циклів;
- сегмент, у якого немає нащадків, називається листовим сегментом.

Граф, що відображає ієрархічну модель даних, називається деревом визначень. Вершини графа - типи сегментів - не повинні бути простими. При роботі з узагальненою деревоподібною структурою використаються два методи доступу до всіх вузлів усередині дерева: прямій порядок обходу дерева й зворотний порядок обходу дерева.

При прямому (спадному) порядку обходу дерева доступ до будь-якого породженого вузла завжди починається з кореня з поступовим обходом під дерев ліворуч праворуч і завершенням обробки в самих нижніх вузлах.

Зворотний порядок обходу дерева починається з доступу до самих нижніх вузлів з поступовим висхідним переходом від одного під дерева до іншому ліворуч праворуч і завершенням обробки в корені.

В ієрархічній моделі баз даних найчастіше використається прямий порядок обходу дерева, оскільки найважливіші дані, як правило, розташовуються на найвищих рівнях дерева, разом з якими можуть бути витягнуті й підлеглого запису.

Питання та завдання до контролю знань студентів

1 Дайте характеристику ієрархічної моделі даних.

2 Освітите структурну частину ієрархічної моделі.

3 Що собою представляє керуюча частина ієрархічної моделі?

4 Які обмеження цілісності збережених даних повинні підтримуватися в ієрархічній моделі даних?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 15

з навчальної дисципліни «Бази даних»

Тема Перетворення концептуальної моделі в ієрархічну

Мета заняття Принципи перетворення концептуальної моделі в ієрархічну

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Перетворення концептуальної моделі в ієрархічну модель даних

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Перетворення концептуальної моделі в ієрархічну модель даних

Перетворення концептуальної моделі в ієрархічну структуру даних багато в чому схоже з перетворенням її в мережну модель, але й має деякі відмінності у зв'язку з тим, що ієрархічна модель вимагає організації всіх даних у вигляді дерева. Перетворення зв'язку типу "один до багатьох" між предком і нащадком здійснюється практично автоматично в тому випадку, якщо нащадок має одного предка, і відбувається це в такий спосіб. Кожен об'єкт із його атрибутами, що бере участь у такому зв'язку, стає логічним сегментом. Між двома логічними сегментами встановлюється зв'язок типу "один до багатьох".

Сегмент із боку "багато" стає нащадком, а сегмент із боку "один" стає предком. Ситуація значно ускладнюється, якщо нащадок у зв'язку має не одного, а двох і більше предків. Так як подібне положення є неможливим для ієрархічної моделі, те відбивана структура даних має потребу в перетвореннях, які зводяться до заміни одного дерева, наприклад, двома (якщо є два предки). У результаті такого перетворення в базі даних з'являється надмірність, тому що єдино можливий вихід із цієї ситуації - дублювання даних.

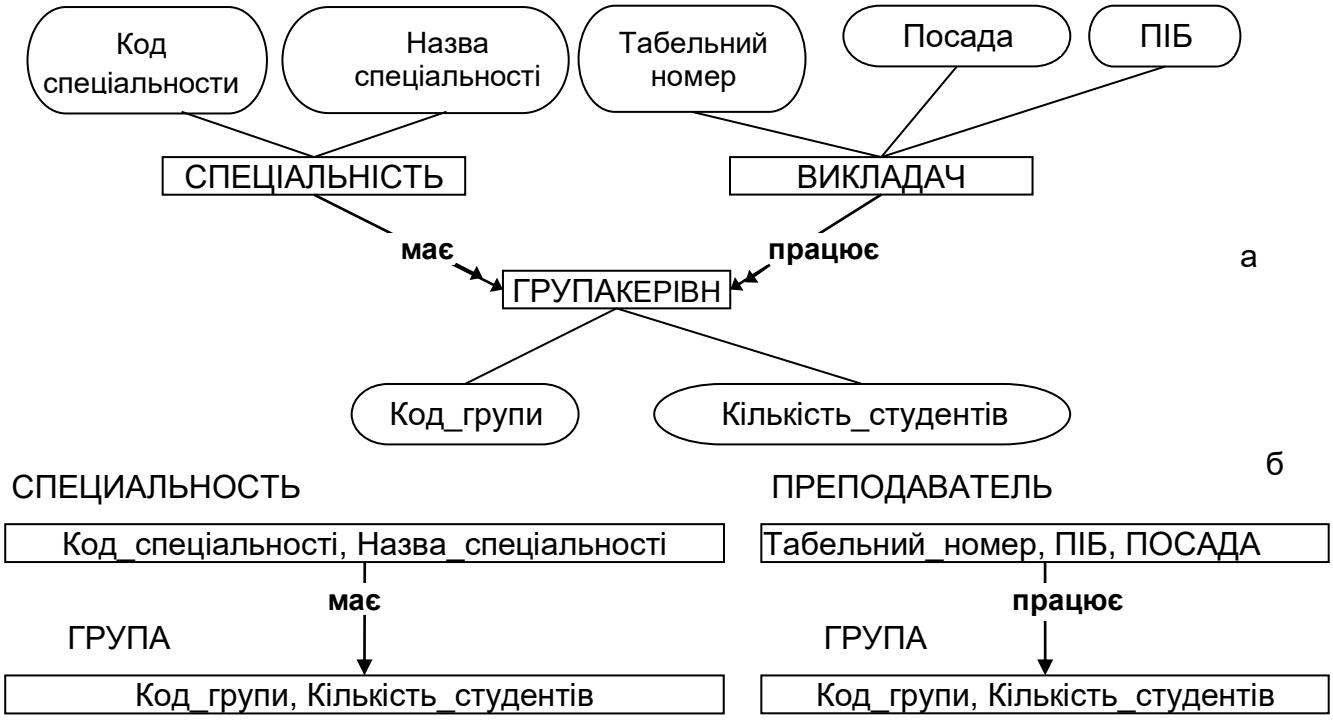


Рисунок 15.1 – Перетворення зв'язку із двома предками
(а - концептуальна модель; б - ієрархічна модель)

Нехай концептуальна модель має вигляд, як на рисунку 15.1, а. Тоді відповідна їй ієрархічна структура повинна містити два дерева, як показано на рисунку 15.2,б. Реалізація подібної структури прямо вимагає дублювання інформації. У наведеному прикладі це стосується інформації про групи. Але дублювання інформації - це небажане явище в інформаційних системах. Через його присутність виникає можливість порушення несуперечності даних, при цьому й обсяг пам'яті витрачається неефективно, а виходить, воно повинне бути мінімізоване. Внесена отриманою структурою надмірність даних може бути обмежена за допомогою віртуальних сегментів і покажчиків у такий спосіб. Сегмент запам'ятовується повністю тільки один раз. Коли сегмент

повинен дублюватися у двох або більше деревах всі наступні входження сегмента запам'ятовуються як покажчики на місце зберігання даного сегмента. Такі входження називаються віртуальними сегментами. При їхньому використанні надмірності даних не виникає, а потрібно лише додаткова пам'ять для зберігання покажчиків.

Перетворення бінарного зв'язку типу "багато хто до багатьох" здійснюється по наступний правилу. Кожен об'єкт, що бере участь у такому зв'язку, з його атрибутами стає логічним сегментом. Нехай це сегменти С1 і С2. У відбиваній структурі даних роблять перетворення, які зводяться до заміни одного дерева двома деревами. Перше дерево включає обидва сегменти, між якими встановлюється зв'язок типу "один до багатьох", де С1 - предок, а С2 - потомок. Друге дерево включає знову обоє цих сегментів, між якими встановлюється зв'язок типу "один до багатьох", але тільки в цьому випадку С2 - предок, а С1 - нащадок.

Описана ситуація добре проглядається на наведеному прикладі (див. рис. 15.2), де кожен викладач може працювати в різних групах, а та сама група займається з різними викладачами. Тут для вихідної концептуальної моделі (див. рис. 15.2,а) наведена відповідна їй ієрархічна структура (див. рис. 15.2,б).

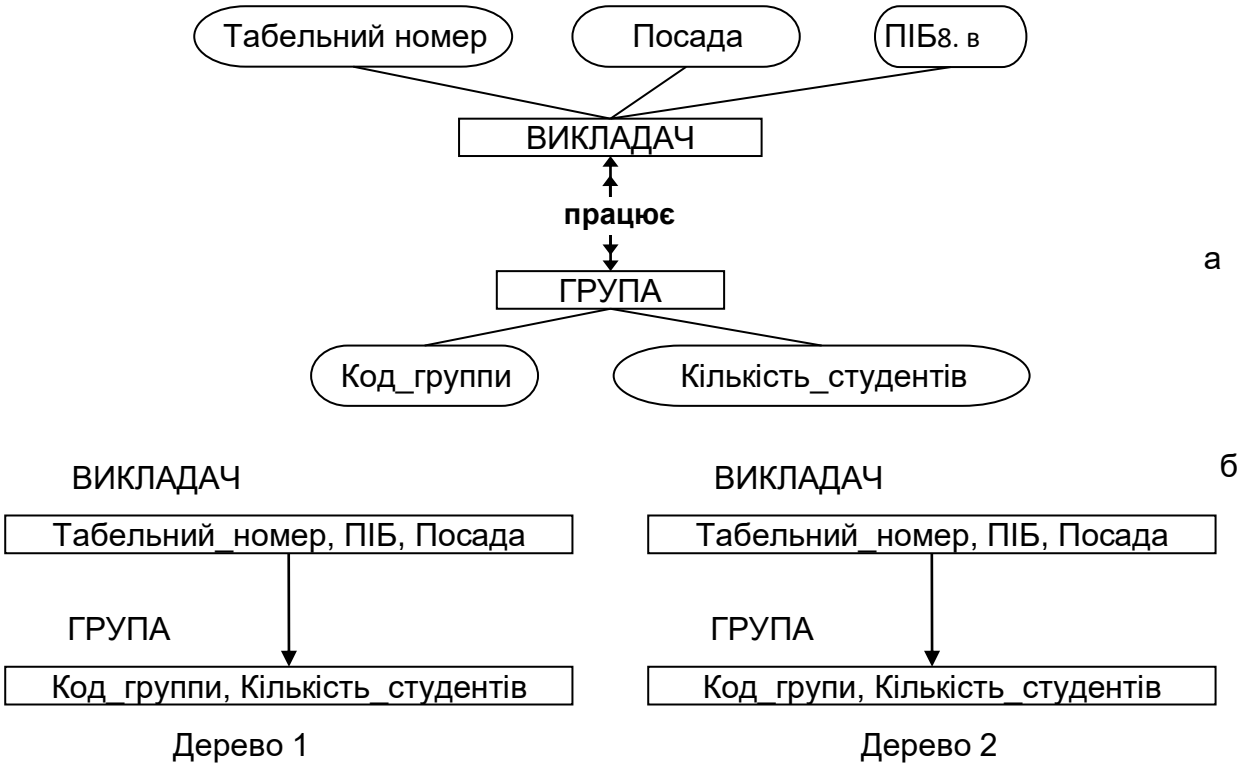


Рисунок 15.2 – Перетворення типу зв'язку "багато хто до багатьох"

(а - концептуальна модель; б - ієрархічна модель)

У тому випадку, коли тип зв'язку сам має атрибути, при перетвореннях в ієрархічну структуру даних створюється додатковий тип сегмента перетинання. Наприклад, викладач зі студентами групи займається в певній аудиторії (див. рис. 15.4). Кожний з типів сегментів, створених з об'єктів, буде функціонувати як корінь окремого дерева. Між типами сегментів двох об'єктів уставляється новий сегмент. Між сегментами предка й нащадка встановлюється зв'язок типу "один до багатьох".

Рисунок 15.3 – Ієрархічна структура подання зв'язку "багато - до - багатьох" з атрибутами

Слід зазначити, що екземпляри нащадки одного типу, пов'язані з одним екземпляром сегмента предка, називають близнюками, а набір всіх екземплярів сегментів, підлеглих одному екземпляру кореневого сегмента, - фізичним записом. Фізичні записи ієрархічних графів розрізняються по довжині й за структурою.

Питання та завдання до контролю знань студентів

1 Викладете правила перетворення концептуальної моделі в ієрархічну модель даних.

2 Що собою представляє керуюча частина ієрархічної моделі?

3 Які обмеження цілісності збережених даних повинні підтримуватися в мережній й ієрархічній моделі даних?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 16

з навчальної дисципліни «Бази даних»

Тема Реляційна структура даних

Мета заняття Дати характеристику основним поняттям реляційної моделі даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

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 відношень.

2.3 Реляційні ключі.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

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 розробленої 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.

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.

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 всім і кожним елементом даних (атомарному значенню) у реляційної базі даних повинен забезпечуватися шляхом використання комбінації імені таблиці, первинного ключа й імені стовпця.

3 правило : Правило підтримки недійсних значень. У дійсної реляційної 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 використаються для подання відсутніх даних незалежно від типу цих даних.

4 правило : Правило динамічного каталогу, заснованого на реляційної 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 обновити, 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 реляційної 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).

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. Відносини використаються 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 пойменований стовпець відносини. 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 відносини r називається кінцева безліч імен атрибутів {A1, A2, ... An}. Заголовки стовпців відносини містять імена його атрибутів й, отже, усе разом відбивають його схему. Схема відносини ВИКЛАДАЧ може бути представлена в такий спосіб: {Табельний номер викладача, Прізвище, Посада} або:

ВИКЛАДАЧ
Табельний номер викладача
Прізвище
Посада

Тоді заголовок відносини ВИКЛАДАЧ прийме вид:

Табельний номер викладача	Прізвище	Посада

Відношення будується з урахуванням ряду факторів. Кожному імені атрибута Ai, 1<=i<=n ставиться у відповідність безліч припустимих для відповідного стовпця значень. Ця множина Di 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 одному 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 використається, 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 здатної забезпечити всі аспекти роботи з ними.

Відносини зображуються у вигляді таблиць, де імена атрибутів виносяться в шапку таблиці. Кожен рядок відносини є безліччю значень, узятих по одному з домену кожного імені атрибута. Домени є довільними непустими

кінцевими або рахунковими множинами й утворять множину: D = D1 D2 ... Dn. Відношення r зі схемою R - це кінцева безліч відображень {t1, t2, ... tp} з R в D. Причому кожне відображення t r повинне задовольняти наступному обмеженню: t(Ai) належать Di, де 1 <= i <= n. Ці відображення називаються кортежами. Кожен кортеж відносини відображає екземпляр об'єкта, а атрибут відносини 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 відносини. Тіло відносини відбиває стан об'єкта, тому в часі воно постійно міняється. Тіло відносини характеризується кардинальним числом, що дорівнює кількості кортежів, що втримуються в ньому.

Рисунок 16.1 – Відношення СПІВРОБІТНИК

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

Всі названі характеристики відносини позначені на рисунку 1, де наведене відношення СПІВРОБІТНИК. У даному відношенні є рядок заголовків стовпців і тіло відносини. У рядку заголовків визначені чотири атрибути, отже, ступінь відносини дорівнює 4. Атрибути мають свої унікальні імена: Р#, ПІБ, Посада, Стаж. Кожен атрибут визначений на своєму домені. Тіло відносини містить ряд рядків - кортежів, де кожен кортеж представляє одного співробітника, а тому в даному відношенні всі рядки різні. Кількість кортежів визначає кардинальне число відносини. У кожному кортежі представлені значення атрибутів, що відповідають даному співробітникові. 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 ПІБ. 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 відносини, 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 заміняти 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 відносини 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 відносини, 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 реляційної 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 реляційної теорії зустрічається кілька видів відносин, але не всі вони підтримуються реальними системами. Розрізняють:

− іменоване відношення - це змінна відносини, 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 відносини;

− подання - це іменоване віртуальне похідне відношення, представлене в системі винятково через визначення в термінах інших іменованих відносин;

− знімки - це відносини, подібні до подань, але вони зберігаються, доступні для читання й періодично обновляються;

− 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 відносини. Це – супер ключі.

В особливу категорію необхідно виділити потенційні ключі. Говорять, що безліч атрибутів ДО = (Ai, Aj, ..., Ak) відносини 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 Ai, Aj, ..., Ak;

− мінімальність: жоден з атрибутів Ai, Aj, ..., Ak 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 використаються 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 посилається. 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 відносини. 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. Відносини 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 відносини змінюються в часі: кортежі можуть додаватися, віддалятися або модифікуватися. Проте , передбачається, що сама схема відносини інваріантне в часі. Відношення повинне сприйматися як безліч можливих станів, які може приймати відношення.

Питання та завдання до контролю знань студентів

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 відносини;
   - 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?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 17

з навчальної дисципліни «Бази даних»

Тема Надмірність даних в базі даних. Аномалії оновлення в базі даних.

Мета заняття Ввести поняття надмірності даних та дати характеристику аномалій модифікації даних.

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Надмірність даних.

2 Аномалії модифікації даних.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Надмірність даних.

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 зазначеної мети:

- 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.

Надмірність даних у БД відноситься до небажаних явищ, оскільки веде до збільшення обсягу пам'яті, необхідного для фізичного зберігання відношень. 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 залишилася тільки частка надмірності, необхідна для втримання бази даних. Характерний приклад відносини, що містять небажану надмірність наведено у таблиці 17.1:

Таблиця 17.1 – 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 уведенні значень атрибутів, у результаті чого база даних перейде в неузгоджений стан. При роботі з відношеннями, що містять надлишкові дані, можуть виникнути проблеми, які називаються аномаліями відновлення.

2 Аномалії модифікації даних.

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. 17.1) відношенні можуть ви-никнути 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_зал_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. 17.1 присутні аномалії включення, а отже, це відношення повинне бути перетворене таким чином, щоб від них позбутися.

Таблиця 17.2 – 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 17.3 – Відношення ГРУПА

Код групи	**ПІБ_старости**	**Куратор**
20-Т-11	Рябов В.С	Доц. Фок И.И.
20-Т-12	Сизов М.М.	Доц. Докин С.С.
20-Т-13	NULL	NULL

Структура відношень, що містить ту ж інформацію, що й відношення СТУДЕНТ, але позбавлена аномалій включення, представлена в табл. 17.2 й 17.3. Для того щоб домогтися зазначеної мети, треба було замість одного відношення сформувати 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 ввести додаткову групу, позначивши невизначені значення описових атрибутів NULL-значеннями.

Аномалії видалення. Повернемося до аналізу відношення, представленого в табл. 17.1. 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-Т-215 Петров Я.Р.20-Т-12 Сизов М.М. Доц. Докин С.С.

з бази даних будуть вилучені весь відомості про групу 20-Т-12. Така ситуація представляє собою аномалію видалення. Для виключення з бази да-ні 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. 17.2 й 17.3. При видаленні кортежу

20-Т -215 Петров Я.Р. 20-Т-12

з відношення СТУДЕНТ відомості про групу 20-Т-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 уведенні 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. 17.1. 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 мінімальної;
- 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.

Питання та завдання до контролю знань студентів

1) Яким образом реалізується підтримка цілісності бази даних?

2) Дайте визначення двох основних правил цілісності бази даних.

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

4) Поясните 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;

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

6) Розглянете відношення з нав'язаною функціональною залежністю. Продемонструйте алгоритм виявлення функціональної залежності.

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

8) Якщо таблиця перебуває в другій нормальній формі, то ця таблиця

- перебуває в першій нормальній формі;
- перебуває в третій нормальній формі;
- не містить транзитивних залежностей;
- містить атрибути, які не є повністю функціонально залежними від ключа,

9) Якщо таблиця перебуває в другій нормальній формі, то ця таблиця

- перебуває в нормальній формі Бойса-Кодда;

– містить неатомарні атрибути;

– не містить транзитивних залежностей;

– містить атрибути, які не є повністю функціонально залежними від ключа.

10) Трьома видами аномалій є аномалії

– вставки, вибору, видалення;

– вставки, модифікації, видалення;

– вибору, модифікації, видалення.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 18

з навчальної дисципліни «Бази даних»

Тема Нормалізація відношень реляційної бази даних

Мета заняття Дати визначення нормальних форм реляційної схеми БД.

Пояснити на прикладах методи приведення відношень до відповідної нормальної форми

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Нормальні форми відношень.

1.1 Перша нормальна форма.

1.2 Друга нормальна форма.

1.3 Третя нормальна форма.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 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 об-межень 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 да-них 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 відношення R(a1 a2, ..., an). Атрибут a2 функціонально залежить від атрибута a1, якщо кожному значенню a1 відповідає єдине значення a2 (тобто кожен кортеж, що має одне й те ж саме значення a1, повинен мати одне й те ж значення a2). Позначається подібна ситуація так: a1 $\rightarrow$ a2. Ліва частина функціональної залежності називається детермінантом, а права частина - залежною частиною. Від-сутність функціональної залежності позначається a $\not\rightarrow$ b. Створення баз да-них переслідує дві основні цілі: понизити надмірність даних і підвищити їхня надійність.

1.1 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 противному 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 використається вроздріб . Розглянемо приклад відносини, представленого в табл. 18.1.

Таблиця 18.1 – 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 частині так, як показано в табл. 18.2.

Таблиця 18.2 – Відношення НАРОДЖЕННЯ

Ім'я	**День_народження**	**Місяць_народження**	**Рік_народження**
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. 18.4). 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. 18.2) таких труднощів не виникає, тому що можна записати залежність: (День_народження, Мі-сяць_народження) $\rightarrow$Знак.

Або, наприклад, табл. 18.3 є ненормалізованою, і вона не перебуває в 1НФ тому, що включає величини, що є сукупністю атомарних значень. Щоб

оде-ржати відношення СТАТЬ, що перебуває в 1НФ, необхідно його представити так, як це зроблено в табл. 18.4.

Таблиця 18.3 – Відношення СТАТЬ

І'мя	**Стать**
(Олександр, Федір)	Чоловіча
(Ганна, Ольга)	Жіноча

Таблиця 18.4 – Відношення СТАТЬ

І'мя	**Стать**
Олександр	Чоловіча
Федір	Чоловіча
Ганна	Жіноча
Ольга	Жіноча

1.2 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 → Ціна) (по визначенню 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 прийдеться 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 прийдеться, 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.

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

– 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 прикладу, якщо розглядати інформаційну цінність використовуваних атрибутів, може виникнути питання про надмірність атрибутів Номер_ лікаря, Номер_ пацієнта. Дійсно, як ключі у відношеннях 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 → (ПІБ_ студента, Тема_ диплома).

Відсутність повної функціональної залежності кожного не первинного атрибута відношення від первинного ключа є джерелом аномалій відновлен-ня й вносить свою частку надмірності в базу даних. Усунення даних негатив-них 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.

1.3 Третя нормальна форма

Розглянемо транзитивну залежність наступного типу: якщо A → B, B-/→A (B не є ключем), B → C, то A → C. У цьому випадку вважається, що C транзитивною залежить від A. Проаналізуємо БД прикладу 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) Ліки; Ліки 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 прикладу 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 прикладу 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) Аудиторія 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 прикладу, 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) Поясните своїми словами зміст термінів:
   - 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) Розглянете відношення з нав'язаною 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

- перебуває в першій нормальній формі;
- перебуває в третій нормальній формі;
- не містить транзитивних залежностей;
- містить атрибути, які не є повністю функціонально залежними від ключа,

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

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

7) Трьома видами аномалій є аномалії

- вставки, вибору, видалення;
- вставки, модифікації, видалення;
- вибору, модифікації, видалення.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 19

з навчальної дисципліни «Бази даних»

Тема Логічне проектування реляційної бази даних. Спрощення концептуальної моделі даних

Мета заняття Охарактеризувати фази проектування реляційної БД

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Фази проектування БД.

2 Логічне проектування реляційної бази даних.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

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 кожну дію.

Питання та завдання до контролю знань студентів

1 Освітите процес проектування реляційної бази даних.

2 Які дії припускає фаза логічного проектування реляційної БД?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 20

з навчальної дисципліни «Бази даних»

Тема Методика перетворення концептуальних структур даних в реляційні структури

Мета заняття Дати визначення принципам перетворення концептуальної моделі в реляційну

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Перетворення концептуальної моделі в реляційну.

1.1 Виключення зв'язку типу "багато - до - багатьох".

1.2 Виключення складних зв'язків.

1.3 Виключення рекурсивних зв'язків.

1.4 Виключення зв'язків з атрибутами.

1.5 Виключення множинних атрибутів.

1.6 Виключення надлишкових зв'язків.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Перетворення концептуальної моделі в реляційну

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 зштовхнутися 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.

1.1 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 потужність зв'язку "багато" - до знову створеного об'єкта. Наприклад, допустимо ситуацію (див. рис.20.1), коли один викладач може читати кілька курсів, і один курс може читатися декількома викладачами.

Рисунок 20.1 – Діаграма зв'язку типу"багато - до - багатьох"

Створимо додатковий об'єкт ЧИТАННЯ й визначимо нові два зв'язки (див. рис. 20.2):

ВИКЛАДАЧ_ЧИТАЄ типу 1:N між об'єктами ВИКЛАДАЧ і ЧИТАННЯ;

КУРС_ЧИТАЄТЬСЯ типу 1:M між об'єктами КУРС і ЧИТАННЯ.

Рисунок 20.2 – Перетворена діаграма зв'язку типу "багато - до - багатьох"

1.2 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 з дорівнює ступеня 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 20.3. 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. 20.4): зв'язок ВИКЛАДАЧ_ ВЕДЕ для об'єктів ВИКЛАДАЧ, ЛАБОРАТОРНІ_ ЗАНЯТТЯ; зв'язок В для об'єктів ЛАБОРАТОРІЯ, ЛАБОРАТОРНІ_ ЗАНЯТТЯ; зв'язок ВЕДУТЬСЯ для об'єктів ЗАНЯТТЯ, ЛАБОРАТОРНІ_ ЗАНЯТТЯ.

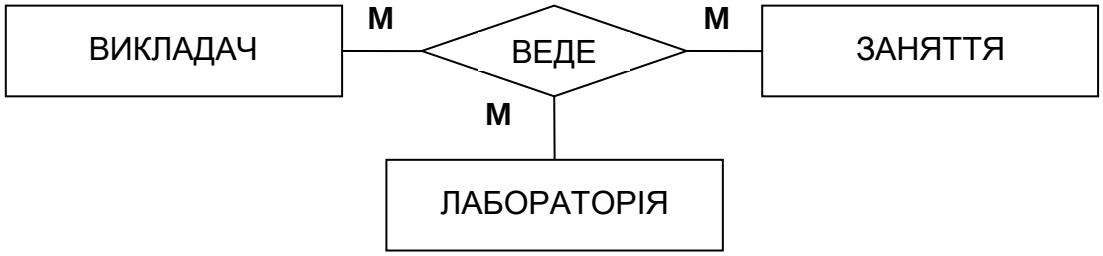


Рисунок 20.3 – Тернарний зв'язок ВИКЛАДАЧ - ЗАНЯТТЯ — ЛАБОРАТОРІЯ

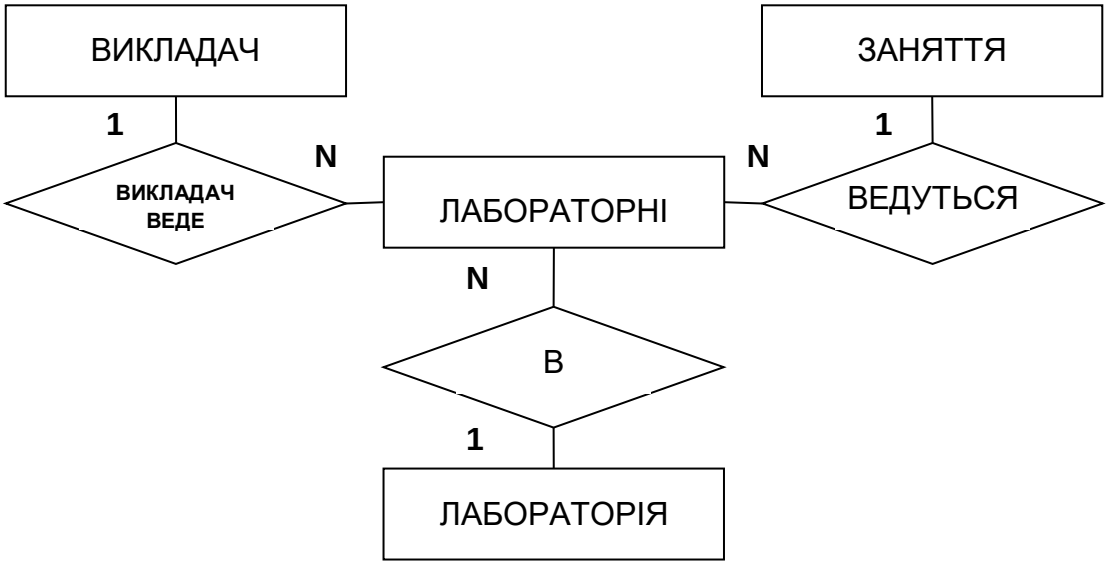


Рисунок 20.4 – Заміна тернарного зв’язку бінарними зв’язками

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

1.3 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 уведення 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. 21.5), 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. 21.6): ПРАЦЮЄ - зв'язок типу "один - до - багатьох"; КЕРУЄ – зв'язок типу "один - до - одного".Отримана в результаті даного перетворення концептуальна модель уже не містить небажані для реляційної схеми бази дані структури.

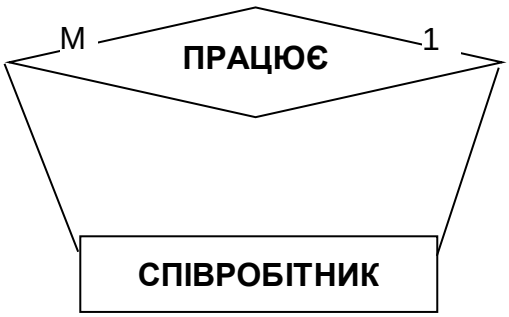


Рисунок 20.5 – Рекурсивний зв'язок

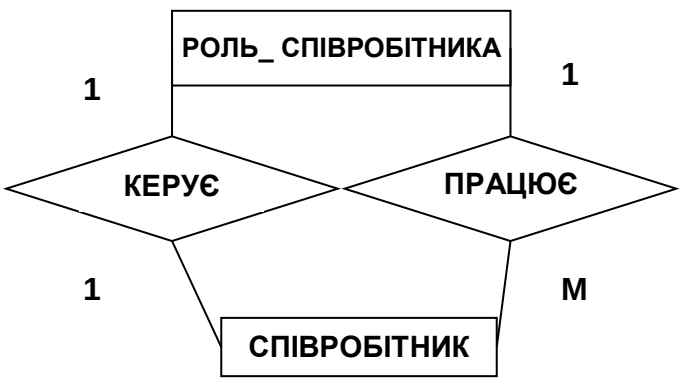


Рисунок 20.6 – Виключення рекурсивного зв’язку

1.4 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 20.7, де зв'язок ЧИТАЄ має свій атрибут Час.

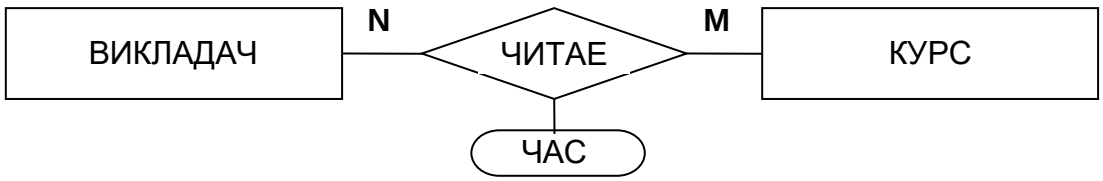


Рисунок 20.7 – Зв'язок з атрибутами

Для виключення з моделі такого зв'язку введемо додатковий об'єкт ЛЕКЦІЯ й припишемо йому атрибут Час (див. рис. 20.8). 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.

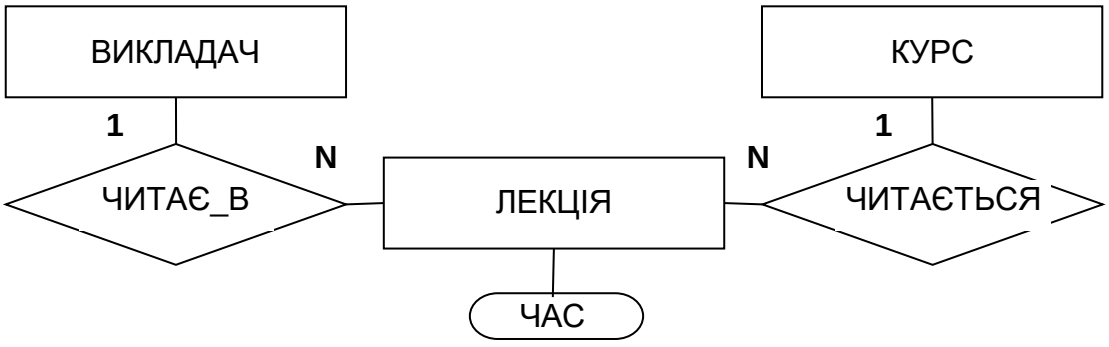


Рисунок 20.8 – Виключення зв’язку з атрибутами

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

1.5 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 20.9.

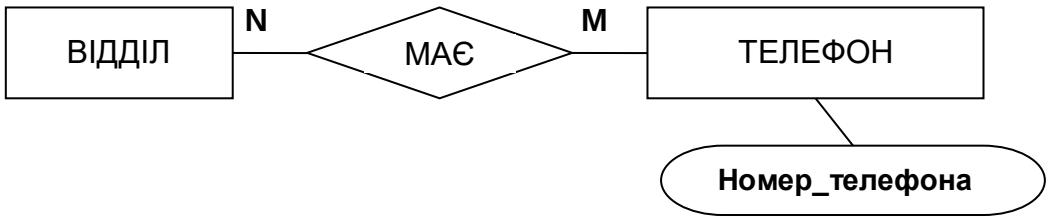


Рисунок 20.9 – Виключення множинного атрибуту

1.6 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.

Методика перетворення концептуальних структур даних в реляційні структури Створенням спрощеної концептуальної моделі предметної області закладається основа для отримання реляційної схеми БД. Оскільки в такій моделі присутній дуже вузьке коло дозволених структур, перетворення кожної з яких має свої особливості, то методика отримання реляційної схеми бази даних являє собою сукупність правил їх перетворення в набір відносин. У спрощеній концептуальної моделі після виключення небажаних структур можуть бути присутніми наступні структури даних:

– об'єкти і атрибути;

– бінарні зв'язку типу 1: 1 і типу 1: N;

– зв'язку типу суперклас - підклас.

Розглянемо правила перетворення кожної із зазначених структур.

Перетворення об'єктів і атрибутів. Нехай проектується БД, призначена для зберігання інформації про викладачів факультету університету і про тих курсах, які вони читають. База даних, отже, включає об'єкти ВИКЛАДАЧ і КУРС (див. рис. 20.10). Об'єкт ВИКЛАДАЧ має в даному випадку два атрибути, один з яких (Табельний номер) може бути використаний для ідентифікації примірника викладача, т. Е. Бути первинним ключем. Т.ч., верхня діаграма може бути перетворена в наступне відношення: ВИКЛАДАЧ (Табельний номер, ПІБ). Об'єкт КУРС має тільки один атрибут, який не можна використовувати в якості ключа, оскільки дисципліни з однаковою назвою можуть мати різні робочі програми для різних

Рисунок 20.10– Об’єкти та їх атрибути

спеціальностей. Тому в даній ситуації для ідентифікації примірника об'єкта необхідно додати ще один атрибут: номер_ курсу (див. рис. 20.11).

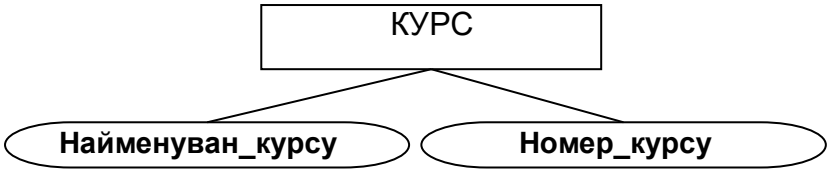


Рисунок 20.11 – Діаграма об'єкта КУРС з введенням ідентифікує атрибута

В результаті маємо наступне відношення: КУРС (Номер курсу, Найменування курсу). Отже, об'єкт з атрибутами може бути перетворений в відношення з ім'ям об'єкта в якості імені відносини і атрибутами об'єкта в якості атрибутів відносини. Якщо деякий набір атрибутів може бути використаний в якості ключа відносини, то він призначається ключем відносини. В іншому випадку в відношення необхідно додати ідентифікує атрибут. Загальний підхід до перетворення об'єктів концептуальної моделі ПрО в відносини реляційної бази даних складається в наступному:

– побудувати набір попередніх відносин і вказати первинні ключі для кожного відносини;

– підготувати список всіх представляють інтерес атрибутів (тих з них, які не було перераховано в діаграмі як первинних ключів об'єктів) та призначити кожен з цих атрибутів одному з попередніх відносин з тією умовою, щоб ці відносини були в НФБК. На цьому кроці для кожного відносини повинні бути визначені межатрібутние функціональні залежності, за допомогою яких перевіряється відповідність відносин НФБК. Якщо отримані відносини в результаті не знаходяться в НФБК або якщо деяким атрибутам чи не знаходиться логічно обґрунтованих місць в попередніх відносинах, то в цих випадках діаграми необхідно переглянути.

Перетворення бінарних зв'язків Кожен об'єкт перетвориться в певне Відношення, а значить, зв'язок між об'єктами перетворюється в зв'язок між відносинами. Виняток з концептуальної моделі складних зв'язків і зв'язків типу "багато до багатьох" вельми спрощує модель ПрО, залишаючи в ній тільки бінарні зв'язки типу "один до одного" і "один до багатьох". Зв'язки між

відносинами в реляційної моделі даних реалізуються за допомогою механізму первинних і зовнішніх ключів. Щоб цей механізм діяв, необхідно в першу чергу визначитися з тим, яке з двох відносин є батьківським, а яке - дочірнім. Батьківським вважається таке відношення, яке передає копію набору значень свого первинного ключа іншого Відношення (дочірньому), де цей набір значень представлятиме зовнішній ключ. Останнє Відношення в цьому випадку буде дочірнім Відношенням. Розглянемо бінарну зв'язок ЧИТАЄ між об'єктами ВИКЛАДАЧ і КУРС (див. рис. 20.12).

Рисунок 20.12 Діаграма бінарного зв'язку

Цей зв'язок може бути зображено за допомогою діаграми, яка містить всю інформацію, необхідну для генерації відповідних відносин РБД. Об'єкт ВИКЛАДАЧ і КУРС однозначно ідентифікуються за допомогою ТН (табельний номер викладача) і НК (номер курсу). Якщо всі елементи даного об'єкта повинні брати участь в зв'язку, то така участь називається обов'язковим. Наприклад, якщо кожен викладач повинен читати один будь-якої курс, то клас приналежності такого об'єкта - обов'язковий. Клас приналежності об'єктів, а також потужність зв'язку між об'єктами є факторами, що визначають структуру проектованої БД.

Попередні відносини для бінарних зв'язків типу 1: 1 Попередні відносини для бінарних зв'язків з показником кардинальності, рівним 1: 1 можуть бути отримані внаслідок перегляду декількох логічних альтернатив і вибору з них найбільш підходящою. Наприклад, нехай в проектованої БД (див. рис. 21.14) повинна зберігатися наступна інформація: ТН - табельний номер викладача; ПІБ - прізвище, ім'я, по батькові викладача; Тел_ Викладача - телефон викладача; НК - номер курсу; Назв_ Курсу - назва курсу.

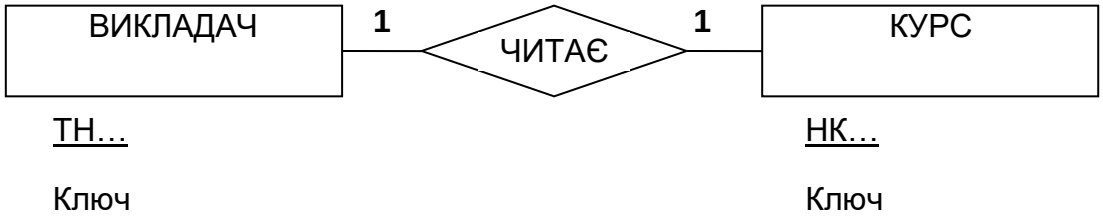


Рисунок 20.13 – Діаграма бінарної зв'язку типу 1: 1

1) *Клас приналежності для обох об'єктів - обов'язковий*. Вважаючи, що клас приналежності є обов'язковим для обох об'єктів, отримуємо Відношення: ВИКЛАДАЧ (ТН, ПІБ, Тел_ Викладача, НК, Назв_ Курсу). У цьому зведеному відношенні знаходиться вся необхідна інформація. Оскільки показник кардинальності зв'язку дорівнює 1: 1 і клас приналежності явля¬тся обов'язковим для обох об'єктів, то гарантується одноразове поява кожного значення ТН і НК в будь-якому екземплярі відносини. Це означає, що відношення ніколи не буде містити ні порожній інформації, ні повторюваних груп надлишкових даних. Первинним ключем цього відношення може бути ключ будь-якого з двох об'єктів. Існування єдиного Відношення в даній ситуації тим більш переважно тоді, коли вихідні два об'єкти не беруть участі в інших зв'язках. Якщо ж з якихось міркувань бажано для кожного об'єкта мати окреме Відношення, то вибір батьківського і дочірнього відносин виконується довільно.

2) *Клас приналежності одного з об'єктів - обов'язковий*. Ситуація буде інша, якщо клас приналежності одного з об'єктів (ВИКЛАДАЧ) - обов'язковий, другого (КУРС) - немає. У цьому випадку одного відносини в БД недостатньо. Прогалини з'являються в ньому у всіх кортежі, що містять інформацію про курсах, що не читаються жодним з викладачів. Існує тільки один вихід з такої ситуації - побудова двох відносин: по одному під кожен об'єкт. При цьому ключ кожного об'єкта повинен служити первинним ключем для відповідного Відношення. Об'єкт, для якого клас приналежності є необов'язковим, перетворюється в батьківське Відношення, а об'єкт, який бере участь в зв'язку з обов'язковим класом приналежності, перетворюється в дочірнє відношення. Для зв'язку отриманих відносин ключ батьківського відносини додається як

атрибут (зовнішнього ключа) в дочірнє відношення. В результаті отримуємо наступну реляционную схему: ВИКЛАДАЧ (ТН, ПІБ, Тел_ Викладача, НК); КУРС (НК, Назв_ Курсу).

3) *Клас приналежності для обох об'єктів - необов'язковий*. У подібній ситуації можна розглянути два альтернативних рішення. Перше рішення передбачає існування двох відносин. Причому в даному випадку абсолютно байдуже, яке з двох відносин буде визначено в якості батьківського відносини. Можливий такий варіант: КУРС (НК, Назв_ Курсу); ВИКЛАДАЧ (ТН, ПІБ, Тел_ Викладача, НК). Можливий і інший варіант: ВИКЛАДАЧ (ТН, ПІБ, Тел_ Викладача); КУРС (НК, Назв_ Курсу, ТН). Але обом цим варіантам властиво поява прогалин у всіх кортежі, що містять інформацію про курсах, що не читаються жодним з викладачів (перша схема), і у всіх кортежі, що містять інформацію про викладачів, які не ведуть ні один курс (друга схема). Кращим рішенням при обставинах, що склалися є визначення трьох відносин - по одному для кожного об'єкта і одного для зв'язку: ВИКЛАДАЧ (ТН, ПІБ, Тел_ Викладача); КУРС (НК, Назв_ Курсу); ЧИТАЄ (ТН, НК). Відносно ЧИТАЄ будь-які значення ТН і НК можуть з'являтися тільки один раз, так як показник кардинальності зв'язку дорівнює 1: 1, і в ньому з'являються тільки ті викладачі, які і читають ті курси, які читаються. Відношення ВИКЛАДАЧ містить відомості про всі викладачів, а відношення КУРС - про всі курсах. Відношення для зв'язку повинно мати серед своїх атрибутів по одному ключу від кожного об'єкта.

Попередні відносини для бінарних зв'язків типу 1: N Нехай в рас-розглядати концептуальної моделі існує бінарна зв'язок типу 1: N (див. рис. 20.14). Для таких зв'язків існує тільки два правила. Варіант визначається класом приналежності N-зв'язкового об'єкта, клас приналежності 1-зв'язкового об'єкта не впливає на кінцевий результат в обох випадках.

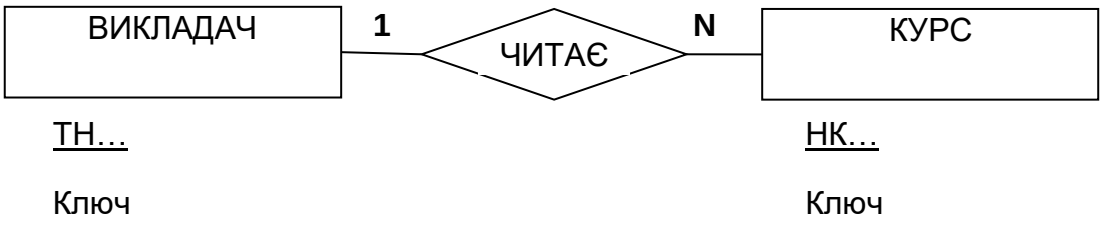


Рисунок 20.14 – Діаграма бінарної зв'язку типу 1: N

1) Перший варіант. ВИКЛАДАЧ - клас приналежності необов'язковий. КУРС - обов'язковий. Якщо в такій ситуації всю інформацію помістити в одне відношення, то в ньому будуть присутні повторювані відомості про викладачів, які читають кілька курсів, і прогалини в полях атрибутів курсу, який не читається ні одним викладачем. Якби клас приналежності об'єкта ВИКЛАДАЧ був обов'язковий, то зникли б прогалини, але повторювані дані залишилися б. Вирішення цього завдання стає можливим, якщо використовувати два відносини, по одному на кожен об'єкт, за умови, що ключ кожного об'єкта служить в якості первинного ключа для відповідного Відношення. Для такої діаграми існує тільки одне безальтернативне визначення батьківського і дочірнього відносин. Як батьківського призначається відношення, відповідне "одиничною" стороні зв'язку, а в якості дочірнього призначається відношення, що представляє "множинну" сторону зв'язку. Для подання зв'язку з цим необхідно скопіювати первинний ключ батьківського відносини в дочірнє відношення, де даний ключ повинен бути описаний як зовнішній. Остаточно в БД увійдуть два відносини: ВИКЛАДАЧ (ТН, ПІБ, Тел_ Викладача); КУРС (НК, Назв_ Курсу, ТН).

2) Другий варіант. Клас приналежності для обох об'єктів - необов'язковий. В одиночному відношенні, моделирующем таку систему, з'являється ряд проблем: виникають прогалини в атрибутах курсу, де викладач не читає курс; виникають прогалини в атрибутах викладача, де курс не читається ні одним викладачем; повторюються відомості про викладачів, які читають більше одного курсу. Якщо скористатися попереднім правилом, то зникнуть перша і третя проблеми, але друга залишиться. Рішення другої проблеми - у формуванні трьох відносин: по одному на кожен об'єкт (причому ключ кожного об'єкта служить первинним ключем відповідного Відношення) і одного відносини для зв'язку. Відношення для зв'язку повинно мати серед своїх атрибутів ключ кожного об'єкта: КУРС (НК, Назв_ Курсу); ВИКЛАДАЧ (ТН, ПІБ, Тел_ Викладача); ЧИТАЄ (ТН, НК).

Перетворення зв'язку типу "суперклас / підклас". Для кожної присутньої в логічної моделі даних зв'язку типу "суперклас / підклас" об'єкт суперкласу необхідно визначити як батьківський, а об'єкт підкласу - як дочірній. Існують різні варіанти предВідношення подібних зв'язків у вигляді одного або декількох відносин. Вибір найбільш відповідного варіанту залежить від обмежень участі і перетину, накладених на учасників зв'язку типу "суперклас / підклас". Якщо суперклас з його підкласами має тотальні і непересічні зв'язку, де кожен екземпляр суперкласу обов'язково повинен бути членом одного і тільки одного підкласу, то найдоцільнішим рішенням є уявлення кожного з підкласів у вигляді окремого відносини, що містить копію первинного ключа суперкласу. Розглянемо підкласи АССИСТЕНТ, СТАРШИЙ ВИКЛАДАЧ, ДОЦЕНТ, ПРОФЕСОР, які є членами суперкласу ВИКЛАДАЧ (див. рис. 20.15).

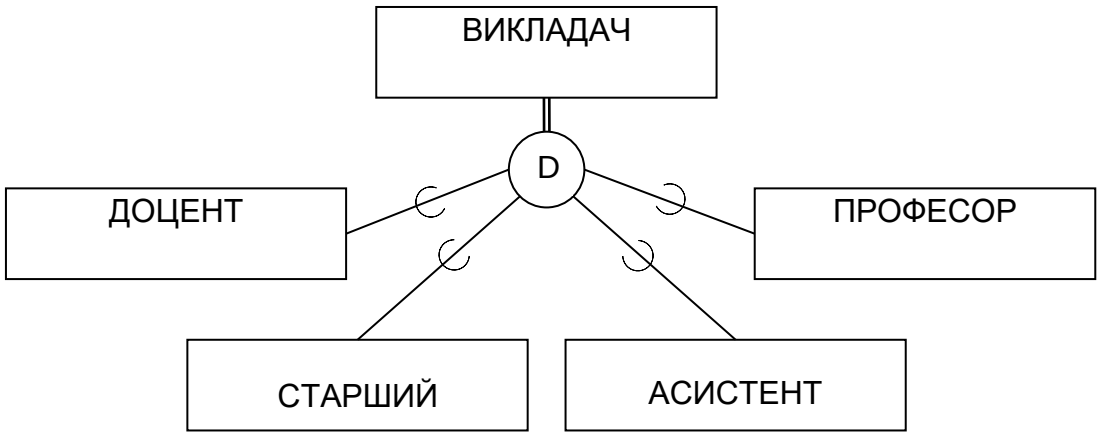


Рисунок 20.15 – Діаграма зв'язку "суперклас / підклас"

Це означає, що кожен екземпляр підкласу є в той же час і екземпляром суперкласу. Причому кожен викладач обов'язково належить одному і тільки одному підкласу. Подібна діаграма перетворюється в наступну реляційну схему відносин: ВИКЛАДАЧ (Табельний номер, ПІБ, Адреса, Педагогічний_ стаж); ПРОФЕСОР (Табельний номер, номер_ диплома_ професора); ДОЦЕНТ (Табельний номер, номер_ диплома_ доцента); СТАРШИЙ_ ВИКЛАДАЧ (Табельний номер): АССИСТЕНТ (Табельний номер). Представлений варіант перетворення концептуальної моделі "суперклас / підклас" в реляційну схему природно не можна рекомендувати як єдино можливий, оскільки він підходить не до всякої ситуації. Розглянутий варіант передбачає організацію відносини

для кожного підкласу. Кількість відносин може бути і меншим - аж до одного для суперкласу і всіх його підкласів. На остаточне рішення можуть вплинути фактори різної природи, наприклад такий: включені чи ні підкласи в окремі зв'язку. Діапазон можливих варіантів досить великий і завжди можна підібрати найбільш оптимальний.

Проте, можна навести приклади, коли виключення такого роду погіршують сприйняття моделі. Наприклад, бінарним зв'язків, за допомогою яких здійснюється уявлення зв'язку високого порядку, не завжди вдається надати осмислене тлумачення, відповідне реальної ситуації в даній галузі. У зв'язку з цим розглянемо способи перетворення в реляційну схему відносин деяких, перш рекомендованих до виключення, структур даних.

Попередні відносини для бінарних зв'язків типу M: N При такої міри зв'язку потрібно три відносини незалежно від класу приналежності обох об'єктів: по одному для кожного об'єкта, причому ключ кожного об'єкта використовується в якості первинного ключа відповідного Відношення, і одного відносини для зв'язку. Останнє повинно мати в числі своїх атрибутів ключ кожного об'єкта. Єдино допустимий варіант в ситуації, що склалася - уявити БД трьома відносинами: КУРС (НК, Назв_ Курсу); ВИКЛАДАЧ (ТН, ПІБ, Тел_ Викладача); ЧИТАЄ (ТН, НК). Вся інформація про курс буде міститися щодо КУРС, інформація про викладача - щодо ВИКЛАДАЧ, а відношення ЧИТАЄ буде містити тільки екземпляри зв'язку, наявні в моделі. Використання прямого перетворення концептуальних зв'язків в логічні структури виявляється дуже корисним при моделюванні складових об'єктів.

Перетворення складових об'єктів Дуже зручно при побудові концептуальної моделі використовувати складові об'єкти. Розглянемо концептуальну модель, представлену на рис.20.16. Внутрішній складений об'єкт ВИКОНУЄ має показник кардинальності зв'язку "багато до багатьох". Відповідно до викладеного правилами перетворення цієї моделі в реляційну модель вимагає генерації трьох відносин: СПЕЦІАЛІСТ (Табельний номер, ПІБ, Посада); ВІД_ РОБОТИ (Код виду, Характеристика); ВИКОНУЄ (Табельний номер, Код виду). Якби перші два відносини мали тільки ключові

атрибути, то ці відносини можна було б виключити, так як ключові атрибути вже містяться в відношенні ВИКОНУЄ

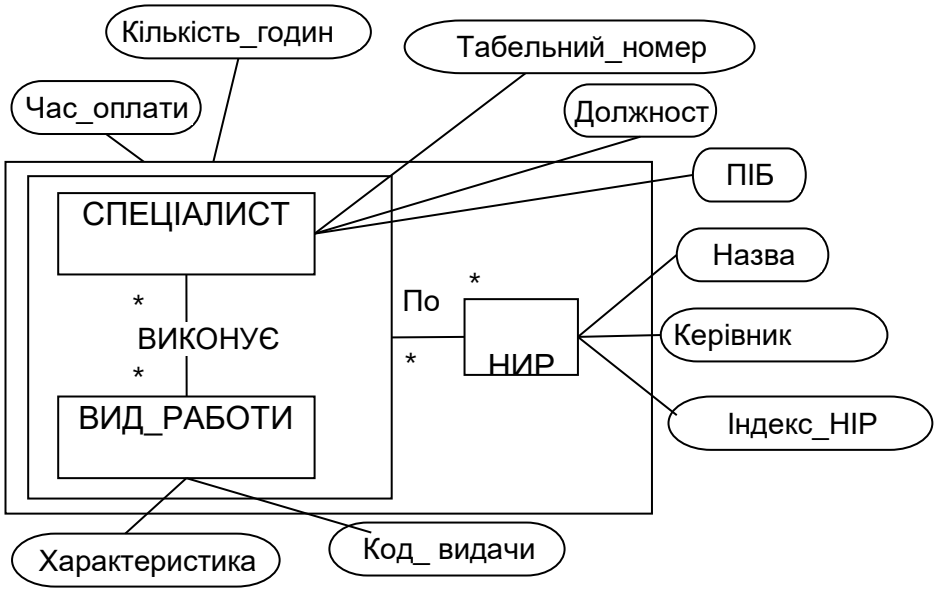


Рисунок 20.16 – Складений об'єкт

Внутрішній складений об'єкт ВИКОНУЄ разом з об'єктом НДР утворюють новий складений об'єкт. Показник кардинальності зв'язку цих об'єктів, також, визначено як "багато до багатьох". Перетворення подібної моделі в реляційну модель вимагає генерації ще двох додаткових відносин: одного для об'єкта НДР, іншого - для зв'язку ПО: НДР (Індекс НДР, Назва, Керівник); ПО (Табельний номер, Код виду, Індекс НДР, Час_ оплата, Кіл_ годин). Останнім відношення крім ключових атрибутів перших двох відносин увійшли атрибути, відносяться до всього цього складеному об'єкту Час_ оплата і Кол_ годин. В результаті перетворення вихідної концептуальної моделі отримана реляційна модель, яка містить п'ять відносин: СПЕЦІАЛІСТ (Табельний номер, ПІБ, Посада); ВІД_РАБОТИ (Код виду, Характеристика); ВИКОНУЄ (Табельний номер, Код виду); НДР (Індекс НДР, Назва, Керівник); ПО (Табельний номер, Код виду, Індекс НДР, Час_оплата, Кіл_годин). Однак слід звернути увагу на надмірність інформації в даній базі. Дійсно, вся інформація, що міститься в відношенні ВИКОНУЄ, присутній також щодо ПО. Виключити зазначену надмірність можна видаленням зі схеми бази даних відносини ВИКОНУЄ, що не можна було б зробити, якби Відношення ВИКОНУЄ містити не ключові атрибути. Кінцевий результат перетворення

виглядає так: СПЕЦІАЛІСТ (Табельний номер, ПІБ, Посада); ВІД_РАБОТИ (Код виду, Характеристика); НДР (Індекс НДР, Назва, Керівник); ПО (Табельний номер, Код виду, Індекс НДР, Час_оплата, Кіл_годин). Неважко помітити, що дана модель являє тернарний зв'язок з власними атрибутами.

Перетворення тернарних зв'язків Хоча за допомогою бінарних зв'язків можуть бути описані багато ситуацій реального світу, проте неминуче виникнення і таких ситуацій, в яких побудова розумної моделі організації не може бути зведене до реалізації тільки таких зв'язків. Не так уже й рідко виникає потреба моделювання тернарних зв'язків. У подібних ситуаціях необхідно використовувати чотири попередніх відносини, по одному для кожного об'єкта (причому ключ кожного об'єкта повинен служити в якості первинного ключа для відповідного відношення) і одне для зв'язку. Відношення, що породжується зв'язком, матиме серед своїх атрибутів ключі кожного об'єкта. Наприклад, моделювання зв'язку між трьома об'єктами ВИКЛАДАЧ, КУРС, СТУДЕНТ вимагає генерації чотирьох відносин: ВИКЛАДАЧ (ТН, ПІБ, Тел_Виклад, ...); КУРС (НК, Назв_ Курсу, ...); СТУДЕНТ (НС, Прізв_ Студ, ...); П_К_С (ТН, НК, НС, ...). Аналогічно, коли зв'язок n-арная, потрібно n + 1 попереднє Відношення. Первинний ключ останнього відносини можна буде визначити лише тоді, коли будуть розподілені між відносинами всі інші, що представляють інтерес атрибути.

Перетворення рекурсивних зв'язків В деяких ситуаціях для повноцінного моделювання ПрО одних об'єктів і зв'язків між ними може виявитися недостатньо. Одна з таких ситуацій виникає тоді, коли елементи деяких об'єктів повинні грати різні ролі в діяльності підприємства. Наприклад, припустимо, що необхідно зберігати інформацію про виробничому персоналі. Розрізняють дві категорії службовців: майстрів і збирачів. Майстри отримують фіксований оклад, в той час як у складальників погодинна оплата. Перша спроба побудови діаграми може виглядати так, як зображено на рис. 20.17.

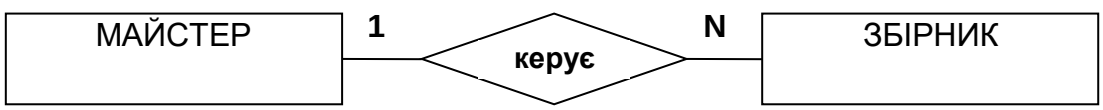


Рисунок 20.17 – Діаграма зв'язку КЕРУЄ

Ключем кожного об'єкта є табельний номер кожного службовця. Передбачається, що жоден майстер не керує іншим майстром, жоден майстер не є збирачем, і жоден збирач не є майстром. Оскільки показник кардинальності зв'язку КЕРУЄ дорівнює 1: N і клас приналежності об'єктів для обох сторін зв'язку є обов'язковим, то відповідно до загального правила будуть побудовані два попередніх відносини: МАЙСТЕР (Номер майстра, ...); ЗБІРНИК (Номер збірника, ..., номер_ майстра). Припустимо, що іншими представляють інтерес атрибутами є: ПрізвСл - прізвище (і ім'я) службовця; РТелМ - номер службового телефону майстра, збирач службового телефону не має; ДТелСл - номер домашнього телефону службовця; АдресСл - домашня адреса службовця; ТСтавкаСб - погодинна тарифна ставка збірника; ОкладМ - місячний оклад майстра. Чи не виникає особливих проблем при розміщенні в одному з двох відносин наступних атрибутів: РТелМ, ТСтавкаСб, ОкладМ. Після того як кожен з цих атрибутів буде логічним чином поміщений в одне з двох відносин, попередні стосунки візьмуть вид: МАЙСТЕР (Номер майстра, Оклад, РТелМ, ...); ЗБІРНИК (Номер збірника, ТСтавкаЗб, ..., номер_ майстра). Нерозміщеними залишилися тільки атрибути ПрізвСл, ДТелСл і АдресаСл. Для повноти подачі варто було б ці три атрибути помістити в обидва відносини. Однак відповідно до загального правила кожен не ключовий атрибут слід розміщувати тільки в одному відношенні. Може виникнути необхідність перевизначити три залишилися атрибута з метою отримання з них наступних шести атрибутів: ПрізвМ - прізвище (і ім'я) майстра; ПрізвЗб - прізвище (і ім'я) збірника; ДТелМ - номер домашнього телефону майстра; АдресаМ - домашня адреса майстра; ДТелЗб - номер домашнього телефону збирача; АдресЗб - домашня адреса іррника. Атрибути Прізв, ДТелМ і АдресаМ можуть бути поміщені в відношення МАЙСТЕР, а атрибути ПрізвЗб, ДТелЗб і АдресаЗб - в відношення ЗБІРНИК. Це невдале рішення, що дозволяє уникнути проблеми відсутності відповідного місця для двох вихідних атрибутів. Якщо все ж змінити запропонованим чином назви атрибутів, то виникне наступна проблема. Припустимо, що потрібно знайти номер домашнього телефону,

наприклад службовця Іванова. Оскільки невідомо, є Іванов майстром або збірником, то необхідно попередньо переглянути спочатку одне відношення, потім інше з метою знаходження прізвища - Іванов. Якщо існують два службовців з прізвищем Іванов - один майстер, а інший - збірник, результатом пошуку, якщо обрано не то відношення, буде невірний номер телефону. Краще рішення досягається шляхом розгляду всієї проблеми з іншої точки зору. Всі майстри і збірники розглядаються в якості суперкласу СЛУЖБОВЦІ, а МАЙСТЕР і ЗБІРНИК - це його підкласи, що визначають ролі, які даний службовець може грати: деякі службовці є майстрами, в той час як інші є збірниками (див. рис. 20.18).

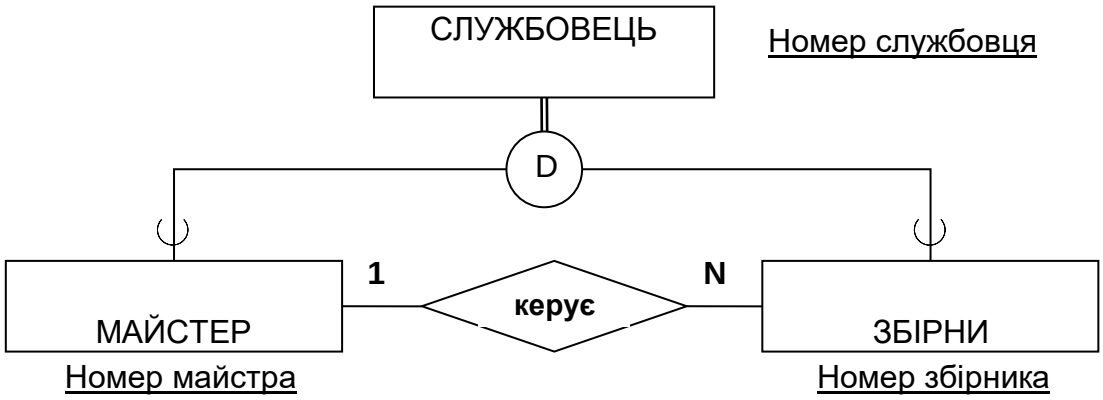


Рисунок 20.18 – Діаграма з рекурсивної зв'язком

На рисунку службовець є об'єктом, ключем якого є табельний номер. Елементи даного об'єкта можуть грати роль або майстра, або збирача. Два підкласу - МАЙСТЕР і ЗБІРНИК з'єднуються зв'язком КЕРУЄ. При генерації попередніх відносин можна керуватися таким правилом: суперклас служить джерелом генерації одного відносини, причому ключ цього суперкласу служить в якості ключа відносини; підкласи і їх зв'язки породжують таке число відносин, яке визначається раніше описаними правилами, причому кожен підклас трактується як звичайний об'єкт. При розподілі всіх інших атрибутів між відносинами, що входять в даний набір, з'ясовується, що кожному атрибуту знаходиться природне місце. Одержуваний в результаті остаточний набір відносин має вигляд: службовець (Номер службовця, ..., ПрізвСл, ДТелСл, АдресСл); МАЙСТЕР (Номер майстра, Оклад, РТелМ, ...); ЗБІРНИК (Номер

збірника, ТОкладЗб, ..., номер_ майстра). Відношення службовець містить інформацію, загальну для всіх службовців. Відношенняи МАЙСТЕР і ЗБІРНИК містять специфічну для кожного підкласу інформацію. Кожне породжене підкласом Відношення пов'язано з відношенням, джерелом породження якого послужив суперклас, через атрибут із загального домену (в даному прикладі - Номер). У розглянутій діаграмі зв'язок КЕРУЄ з'єднує два підкласу одного суперкласу. Такий тип зв'язку називається рекурсивним. Цей зв'язок рекурсивна в тому сенсі, що з точки зору суперкласу службовці керують службовцями. Для збільшення швидкості доступу при запитах проектувальник може розглянути можливість додавання в відношення службовець атрибута, що визначає, ким конкретно є службовець - майстром або збирачем. Це дозволить уникнути необхідності організації пошуку в стосунках МАЙСТЕР і ЗБІРНИК, мета якого полягає у визначенні типу роботи окремого службовця, а можливість - обумовлюється відомим табельною номером.

Отже, розглянуті способи перетворення конструкцій концептуальної моделі - об'єктів, атрибутів, зв'язків і складових об'єктів - в реляційних відношеннях. Після того як перетворення всіх конкретних конструкцій закінчено, отриману реляційну схему необхідно ще раз переглянути на предмет позбавлення від надмірності. Будь-які надлишкові відношення (тобто відношення, інформація яких повністю міститься в інших відношеннях схеми) необхідно видалити зі схеми. Крім проведення зазначеного аналізу отримана модель даних повинна бути піддана ще цілої низки перевірок:

- перевірці моделі за допомогою концепцій послідовної нормалізації;
- перевірці моделі щодо транзакцій користувачів;
- перевірці підтримки цілісності даних.

Питання та завдання до контролю знань студентів

1) Яким чином можна спростити концептуальну модель даних, щоб полегшити процес переходу від концептуальної моделі до логічної моделі?

2) Викладете правила методики перетворення концептуальних структур даних у реляційної структури.

3) Побудуйте концептуальну модель із Про Бібліотека й з її допомогою одержите реляційну модель відповідної бази даних.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 21

з навчальної дисципліни «Бази даних»

Тема Перевірка реляційної моделі даних

Мета заняття Визначити принципи перевірки реляційної моделі щодо нормалізації цілісності даних та транзакцій

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Перевірка моделі за допомогою концепцій послідовної нормалізації.

2 Перевірка моделі щодо транзакцій користувачів.

3 Перевірка підтримки цілісності даних.

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

Після того як перетворення всіх конкретних конструкцій закінчено, отриману реляційну схему необхідно ще раз переглянути на предмет позбавлення від надмірності. Будь-які надлишкові відносини (тобто відносини, інформація яких повністю міститься в інших відносинах схеми) необхідно

видалити зі схеми. Крім проведення зазначеного аналізу отримана модель даних повинна бути піддана ще цілої низки перевірок:

– перевірці моделі за допомогою концепцій послідовної нормалізації;
– перевірці моделі щодо транзакцій користувачів;
– перевірці підтримки цілісності даних.

1 Перевірка моделі за допомогою концепцій послідовної нормалізації

Створений на попередньому етапі попередній набір відносин логічної моделі даних повинен обов'язково бути підданий аналізу на коректність об'єднання атрибутів в кожному з відносин. Перевірка коректності складу кожного з відносин повинна проводитися за допомогою застосування до них процедури послідовної нормалізації. Метою застосування цієї процедури є отримання гарантій того, що кожне з відносин, отриманих на основі концептуальної моделі, знаходиться, по крайній мере, в НФБК. Якщо в процесі аналізу відносин моделі будуть знайдені відносини, що не відповідають вимогам НФБК, то це буде означати, що десь на попередніх етапах були допущені помилки. Можливо, помилки з'явилися при побудові концептуальної моделі, а можливо - в процесі її перетворення в логічну модель. Для забезпечення коректності логічної моделі в такій ситуації доведеться повернутися на ранні етапи проектування і перебудувати помилково створені фрагменти моделі. Правильно виконані дії з проектування моделі предметної області повинні обов'язково привести до створення набору відносин в НФБК. Функціональні залежності, визначені для реляційної моделі, є атрибутами зв'язків з показниками кардинальності "один до одного" або "один до багатьох". Описаний процес перетворення кожної з цих конструкцій в атрибути реляційних відносин гарантує, що вони будуть залежати тільки від ключових атрибутів. Таким чином, кожна отримана реляційна таблиця буде мати НФБК.

2 Перевірка моделі щодо транзакцій користувачів

Крім виконання описаної перевірки, створена реляційна модель предметної області повинна бути проаналізована в плані можливості виконання всіх транзакцій, передбачених уявленнями користувачів. Перелік транзакцій визначається відповідно до специфікацій, що описують дії, що виконуються

користувачами. Найбільш дієвий метод здійснення зазначеної перевірки полягає в нанесенні безпосередньо на ER-діаграми всіх шляхів, які будуть потрібні для виконання кожної з транзакцій. Такий підхід дозволяє виділити в моделі цінні з точки зору транзакцій її області, і навпаки, виявити області, даремні для транзакцій. По відношенню до марним для транзакцій областям концептуальної моделі можна цілком справедливо поставити питання про доцільність їх присутності в моделі. У той же час, якщо в моделі присутня область, що не дозволяє виконати деяку необхідну користувачу транзакцію, то це може свідчити про те, що в даній області не вистачає якогось об'єкта або зв'язку, а отже, модель потребує доопрацювання.

3 Перевірка підтримки цілісності даних

Про необхідність підтримувати в базі даних узгоджені несуперечливі дані вже декларувалося неодноразово. Побудувавши логічну модель даних предметної області, проектувальник якраз підійшов до такого моменту, коли від декларацій потрібно перейти до справи і необхідно зайнятися цим питанням впритул. При цьому слід звернути увагу на наступні питання:

- можливість для атрибутів мати порожні значення;
- обмеження для доменів атрибутів;
- категорна цілісність;
- посилальна цілісність;
- бізнес-правила існують у даній галузі.

Всі ці складові підтримки цілісності бази даних вже досить повно обговорювалися і в даний момент просто необхідно виконати копітку роботу з перевірки кожної складової стосовно кожного об'єкту, до кожного атрибуту, до кожної зв'язку логічної моделі предметної області. У тому випадку, якщо і ця перевірка дасть позитивний результат, можна переходити до наступного етапу проектування бази даних - фізичній проектування.

Питання та завдання до контролю знань студентів

1) Яким чином можна спростити концептуальну модель даних, щоб полегшити процес переходу від концептуальної моделі до логічної моделі?

2) Викладіть правила методики перетворення концептуальних структур даних в реляційні структури.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 22

з навчальної дисципліни «Бази даних»

Тема Фізична організація даних

Мета заняття Визначити принципи організації збереження даних в СУБД та організації файлової структури

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Технології зберігання даних в СУБД.

2 Сторінкова організація даних в СУБД

3 Файлові структури баз даних

4 Хешування

5 Індексування

Література

1 Малыхина, М.П. Базы данных: основы, проектирование, использование [Текст] / М.П. Малыхина – СПб.: БХВ-Петербург, 2004, – 512с.: ил.

2 Пасічник, В.В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл.

3 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

Навчальні матеріали лекції

1 Технології зберігання даних в СУБД

Спочатку розглянемо вступний огляд технологій фізичного зберігання і здійснення доступу до даних на пристроях прямого доступу. Фізична

організація сучасних БД, по-кільки вона багато в чому визначає продуктивність СУБД, являє собою комерційну таємницю. У такій ситуації кожним постачальником доводиться створювати свою унікальну структуру, бажано перевершує за якостями продукцію своїх конкурентів. У зв'язку з цим основною метою викладу є висвітлення загальних підходів і технологій, призначених для використання в цій галузі організації СУБД, без зайвих подробиць їх реалізації.

Потім розглянемо огляд технологій зберігання даних конкретної СУБД.

Будь-яке впорядкування даних на диску називається структурою зберігання. На жаль, до сих пір не створена така ідеальна структура зберігання, яку можна було б використовувати як оптимальну для будь-яких завдань. Але зате можна організувати найрізноманітніші структури збер-ня, що володіють різною продуктивністю, область застосування яких залежить від того, наскільки вони ефективні в тому чи іншому випадку. Організація обробки даних посредст-вом реляційних СУБД має свою специфіку. Файлова структура і система управління фай-лами підкоряються законам операційної системи. По відношенню до баз даних ці принципи можуть бути далекі від оптимальності, а тому СУБД підпорядковується дещо іншими принципами і стратегіям управління зовнішньою пам'яттю. Позначимо найбільш важливі особливості, що впливають на організацію зовнішньої пам'яті.

1) Наявність двох рівнів системи: рівня безпосереднього управління даними у зовнішній пам'яті і мовного рівня. При такій організації підсистема нижнього рівня повинна під-тримувати у зовнішній пам'яті набір базових структур, конкретна інтерпретація яких входить в число функцій підсистеми верхнього рівня.

2) Необхідність організації відносин-каталогів з метаданими, підтримуваних підсистем-мій мовного рівня. З точки зору структур зовнішньої пам'яті відношення-каталог нічим не відрізняється від звичайного відносини бази даних.

3) Регулярність структур даних забезпечується тим, що основним об'єктом реляційної моді-ли даних є плоска таблиця.

4) Необхідність забезпечити можливість ефективного виконання операторів мовного рівня як над одним відношенням, так і над декількома відносинами. Для цього у зовнішній пам'яті повинні підтримуватися додаткові "керуючі" структури - індекси.

5) Для виконання вимоги надійного зберігання баз даних необхідно підтримувати хати-точність зберігання даних, що зазвичай реалізується у вигляді журналу змін бази даних.

Для функціонування СУБД у зовнішній пам'яті бази даних виникає необхідність зберігати такі різновиди об'єктів:

– рядки відносин - основна частина бази даних, здебільшого безпосередньо видима користувачам;

– керуючі структури - індекси, що створюються з ініціативи користувача (адміністративної та кримінальної відповід-ра) або верхнього рівня системи з міркувань підвищення ефективності виконання за-просо і зазвичай автоматично підтримувані нижнім рівнем системи;

– журнальну інформацію, підтримувану для задоволення потреби в надійному збе-неніі даних;

– службову інформацію, підтримувану для задоволення внутрішніх потреб ниж-нього рівня системи.

Оскільки СУБД в силу зазначеної специфіки не може використовувати загальносистемні ме-ханізм буферизації і управління файловими структурами, то вона змушена сама брати на се-бе повноваження з управління розміщенням даних і розміщувати їх на своїх зовнішніх носіях, де простір зовнішньої пам'яті управляється безпосередньо самої СУБД. Такі системи організації даних у зовнішній пам'яті спираються на деякі механізми, що використовують сторінковий принцип організації. Розглянемо технологію доступу до бази даних.

Доступ до бази даних Пошук і надання даних користувачеві здійснюються з по-міццю кількох програм доступу даних і включають в себе три основних етапи (див. рис. 22.1):

1) Визначається шукана запис, для вилучення якої запитується диспетчер файлів.

2) Диспетчер файлів визначає сторінку, на якій знаходиться шукана запис, і для через потягу цієї сторінки запитує диспетчер дисків.

3) Диспетчер дисків визначає фізичне становище шуканої сторінки на диску і посилає відповідне питання на введення-виведення даних.

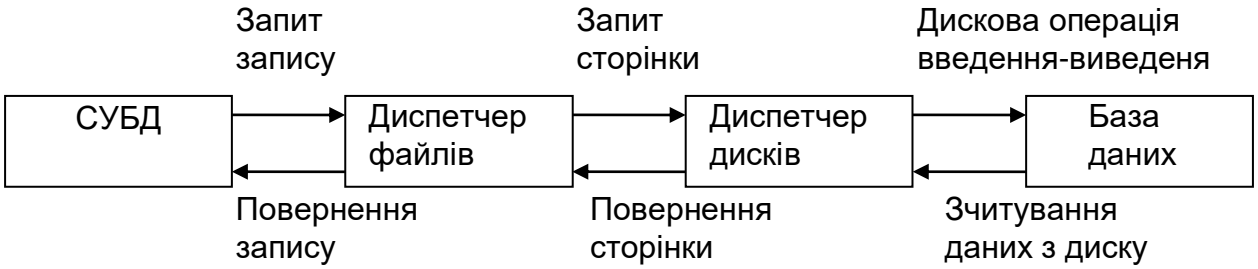


Рисунок 22.1 – Схема доступу до бази даних

Слід зазначити, що організація зберігання даних передбачає використання технології кластеризації даних, що представляє собою найважливіша умова високої продуктивності. Ця технологія забезпечує якомога ближче фізичне розміщення на диску логічно пов'язаних між собою і часто використовуваних даних. Виділяють внутріфайлову і межфайлову кластеризацию. В сучасних СУБД адміністратором БД відповідно до вимог, що пред'являються до продуктивності СУБД, можуть встановлюватися кількох різних типів кластеризації даних з різних файлів.

2 Сторінкова організація даних в СУБД

Через те, що СУБД підпорядковується дещо іншими принципами і стратегіям управління зовнішньою пам'яттю, ніж ті, які використовують більшість призначених для користувача процесів або задач, СУБД змушені брати на себе безпосереднє керування зовнішньою пам'яттю. При цьому простір зовнішньої пам'яті повністю управляється СУБД, а операційне середовище не отримує безпосереднього доступу до цього простору. При розподілі дискового простору розглядаються дві схеми структуризації: фізична,

яка визначає збережені дані (див. рис. 22.2), і логічна, яка визначає деякі логічні структури, пов'язані з концептуальною моделлю даних.

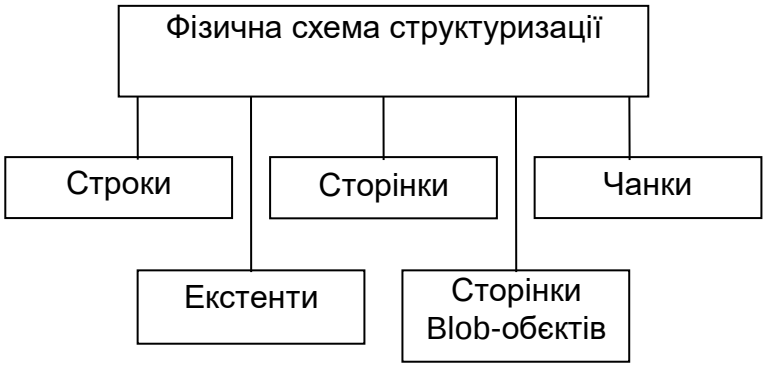


Рисунок 22.2 – Класифікація об'єктів фізичної моделі даних

У даній класифікації використовується ряд понять:

Сторінка даних - основна одиниця здійснення операцій обміну (введення-виведення).

Чанк - являє собою частину диска, фізичний простір на диску, яке ассоціірова-но одному процесу.

Екстент - це безперервна область дискової пам'яті.

Сторінки Blob призначені для зберігання слабоструктурованої інформації, що містить тексти великого обсягу, графічну інформацію, двійкові коди.

Сторінки даних Основними одиницями здійснення операцій обміну є сторінки даних, керовані диспетчером дисків. Всі дані зберігаються посторінково. Т.ч., з точки зору СУБД база даних виглядає як набір записів, які проглядаються за допомогою диспетчера файлів. З точки зору диспетчера файлів база даних виглядає як набір сторінок, які можуть переглядатися за допомогою диспетчера дисків. На одній сторінці зберігаються однорідні дані, наприклад, тільки дані або тільки індекси. Всі сторінки даних мають однакову структуру, представлену на рис. 22.3.

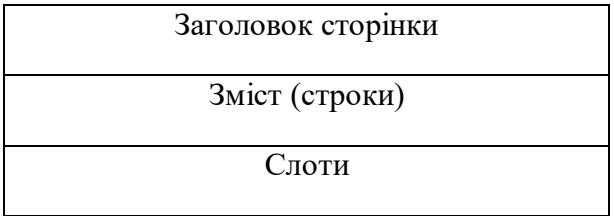


Рисунок 22.3 – Структура сторінки даних

Заголовок сторінки використовується для завдання за допомогою покажчиків логічної послідовності сторінок в даному наборі. Він також містить

інформацію про фізичну дисковому адресі сторінки, яка логічно випливає за даною сторінкою (див. рис. 22.4 - правий верхній кут).

Заголовок						5
Запис1	…	…	…	…	…	…
Запис2	…	…	…	…	…	…
Вільний простір						

Рисунок 22.4 – Визначення логічної послідовності сторінок

Заголовки сторінок і покажчики на наступні сторінки обробляються тільки диспетчером дисків і приховані від диспетчера файлів. Для виконання своїх функцій в розпорядженні диспетчера дисків є каталог (сторінка 0), що містить інформацію про всі наявні на даному диску наборах сторінок разом з покажчиками на перші сторінки цих наборів. За допомогою диспетчера дисків вдається вирішити і проблему збереження логічно пов'язаних сторінок в фізично близько розташованих місцях. Так, вставка деякого запису при наявності вільного простору на сторінці буде проведена на те місце, яке цей запис займає в логічній послідовності записів, із зсувом вниз записів, вже збережених на сторінці, але логічно прямувати за нею. До того ж відповідність фізичної та логічної послідовності досягається також і за рахунок того, що диспетчер дисків розміщує або видаляє сторінки в наборах не по одній, а цілими екстенти. Слоти характеризують розміщення рядків даних на сторінці. На одній сторінці зберігається не більше 255 рядків. Кожен запис має унікальний ідентифікатор, що не змінним під весь час її існування, він має певну довжину і складається з номера сторінки, на якій даний запис знаходиться, і байта зміщення слоту від кінця сторінки, який в свою чергу містить байт зміщення записи від початку сторінки (див. рис. 22.5). Подібна схема ідентифікації записи поєднує в собі швидкість безпосередньої адресації і гнучкість непрямої адресації, тим самим, забезпечуючи необхідну ефективність пошуку інформації. Зазвичай кожен запис зберігається цілком в одній сторінці.

З цього випливає, що максимальна довжина запису будь-якого відношення обмежена розмірами сторінки.

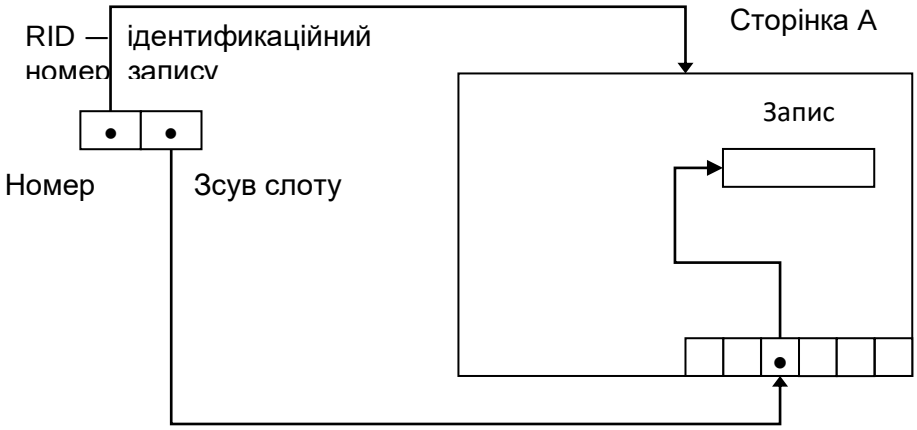


Рисунок 22.5 – Структура ідентифікаційного номера запису

При такій організації зберігання даних виконання процедури впорядкування рядків на сторінках не викликає їх фізичне переміщення, все маніпуляції відбуваються зі слотами. При переповненні сторінок створюється спеціальний вид сторінок, називаємих сторінками переповнення. Рядки, що не вмістилися на основній сторінки, зв'язуються зі своїм продовженням на сторінках переповнення за допомогою посилань-покажчиків "вперед", які містять номер сторінки і номер слота на сторінці. При наступному переповненні сторінки запис знову переміщається таким же чином на нову сторінку переповнення. Проблема розподілу пам'яті на сторінках даних пов'язана з проблемами синхронізації і журналізації. Якщо в ході виконання транзакції видаляються дані з деякою сторінки, то її не можна перевести в статус вільних сторінок до кінця транзакції, оскільки при відкат транзакції віддалені при прямому виконанні транзакції і відновлені при її відкат записи повинні отримати ті ж самі ідентифікатори. Поширеним способом підвищення ефективності СУБД є кластеризація Відношення за значеннями одного або декількох стовпців. Корисним для оптимізації з'єднань є спільна кластеризація декількох відносин.

Таблиці даних Як правило, на одній сторінці даних зберігаються записи тільки однієї таблиці. Існують варіанти з можливістю зберігання на одній сторінці записів кількох таблиць, що іноді дозволяє різко скоротити число обмінів із зовнішньою пам'яттю при виконанні з'єднань. Таблиця моделюється

сукупністю екстентів. Для моделювання кожної таблиці використовується два типи екстентів: перший і наступні. Перший екстент задається при створенні нового об'єкта типу таблиця, його розмір задається при створенні. Мінімальний розмір екстента в кожній системі свій, але в більшості випадків він дорівнює 4 сторінок, максимальний - 2 Гбайтам. Новий екстент створюється після заповнення попереднього і зв'язується з ним спеціальним посиланням, яка розташовується на останній сторінці екстента. Усередині екстента йде облік вільних станиць. Екстенти складаються з чотирьох типів сторінок: сторінки даних, сторінки індексів, бітові сторінки та сторінки Blob-об'єктів (Binary Large Object), які відповідають неструктурованих даними. В сучасних СУБД до цього типу належать неструктуровані текстові дані, картинки, просто набори машинних кодів. Зміна схеми зберігається відносини з додаванням нового стовпця не викликає потреби у фізичній реорганізації відносини. Досить лише змінити інформацію в описувач відносини і розширювати записи тільки при занесенні інформації в новий стовпець. Т.ч., підбиваючи підсумки розгляду загального підходу організації зберігання даних і доступу до них на фізичному рівні, слід зазначити, що ця організація спрямована на те, щоб забезпечити підтримку функціональності СУБД з максимально можливою ефективністю, причому так, щоб користувач не помічав всієї складності і багатосупеневості даної проблеми. Диспетчер дисків приховує всі подробиці фізичної організації введення-виведення від диспетчера файлів, який веде роботу тільки на логічному рівні. На своєму рівні СУБД вже має справу тільки з збереженими записами і файлами. З точки зору СУБД зберігається запис володіє певною структурою, а з точки зору диспетчера файлів - це просто бітова рядок. Описана технологія зберігання даних на фізичному рівні і спрямована на те, щоб забезпечити якомога більш високу ефективність операцій введення-виведення, все-таки її можливостей явно недостатньо. Продуктивність і інші показники сучасних СУБД спираються і на інші технології, які пропонують способи доступу, що відрізняються від тих, які спираються на фізичну послідовність даних.

3 Файлові структури баз даних

Файли і файлові структури, широко застосовуються практично у всіх СУБД для зберігання інформації у зовнішній пам'яті, можна класифікувати наступним чином (див. рис. 22.6).

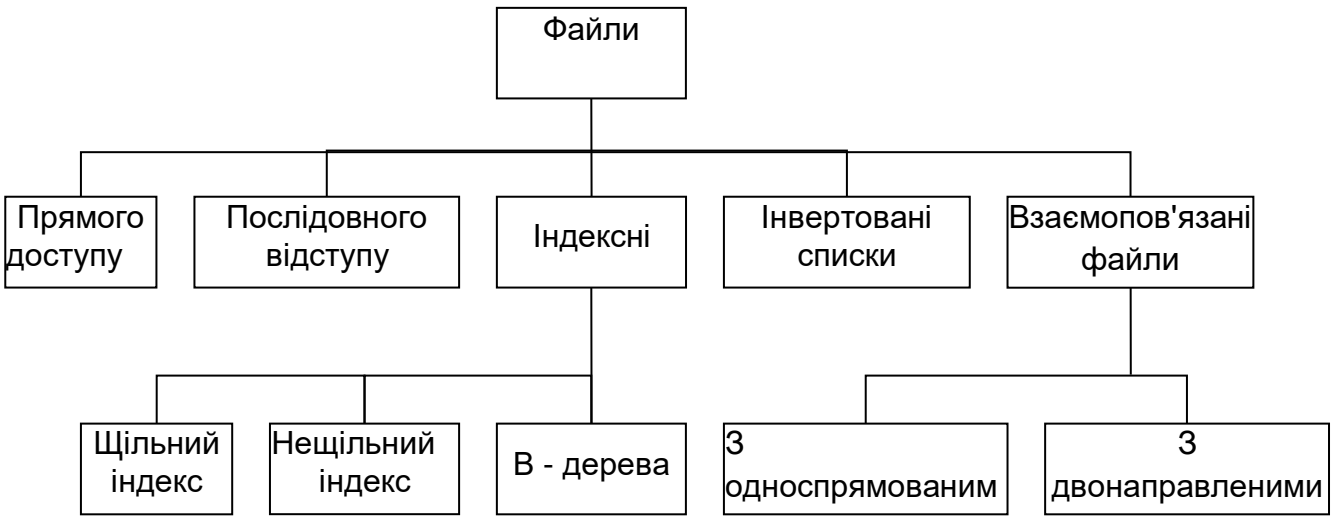


Рисунок 22.6 – Класифікація файлів, використовуваних в системах баз даних

Оскільки файл - це лінійна послідовність записів, то завжди в файлі можна визначити поточну запис, що передує їй і наступну за нею. Для кожного файлу в системі зберігає-ся наступна інформація:

– ім'я файлу;
– тип файлу (наприклад, розширення або інші характеристики);
– розмір запису;
– кількість зайнятих фізичних блоків;
– базовий початковий адресу;
– посилання на сегмент розширення;
– спосіб доступу.

Відомо, що відповідно до методами управління доступом розрізняють пристрої зовнішньої пам'яті довільного і послідовного доступу. На пристроях послідовного доступу можуть бути організовані файли тільки послідовного доступу. Взагалі файли можуть мати постійну або змінну довжину запису. Файли зі змінною довжиною запису завжди є причиною файлами послідовного доступу. Файли з постійною довжиною запису, розташовані на пристроях прямого доступу, є файлами прямого доступу.

Найбільш перспективним у системах баз даних вважається використання файлів прямого доступу, оскільки за допомогою їх забезпечується найбільш швидкий доступ до довільних записам. Для файлів з постійною довжиною запису фізичну адресу розташування потрібного запису може бути обчислений за номером запису. Але доступ по номеру запису в базах даних досить неефективний. Найчастіше в БД необхідний пошук по ключу, для якого необхідно визначити відповідний йому номер запису, а отже, краще пов'язувати між собою не номер запису і фізичну адресу, а значення ключа запису і номер запису файлу. При організації файлів прямого доступу в деяких дуже рідкісних випадках можлива побудова лінійної забезпечує однозначне відповідність функції, яка за значенням ключа однозначно обчислює номер запису файлу. Але найчастіше такий підхід виявляється неприйнятним і тоді доводиться шукати вихід у залученні інших технологій.

4 Хешування

Хешуванням називається технологія швидкого прямого доступу до інформації, що зберігається записи на ос-нове заданого значення деякого поля, яке може бути навіть не ключовим. Проте, надалі будемо вважати, що мова йде про ключове поле. Технологія хешування передбачає наступні моменти:

1) Кожен запис БД розміщується за адресою, обчислюваному шляхом застосування до значення ключа деякої функції згортки (хеш-функції), що виробляє значення меншого розміру. Згортка значення ключа застосовується для того, щоб уникнути неефективного використання дискового простору. Досягається подібний ефект підбором спеціальної функції. Вказана неефективність обов'язково виникне, якщо замість згортки використовувати значення не-посередньо ключового поля, які розкидані по домену, а якщо діапазон значень ключового поля значно ширше діапазону наявних адрес, такий підхід взагалі надає-ся неприйнятним. Наприклад, ключове поле містить 7 значень в діапазоні 0 – 1000. У ре-док використати відповідну хеш-функції отримаємо 7 її значень в діапазоні 0 – 9.

2) Згортка значення ключа використовується і для доступу до запису. Її отримане значення береться в якості адреси початку пошуку, тим самим обмежується час пошуку (кількість додаткового-них кроків) для остаточного отримання адреси. У найпростішому, класичному випадку, згортка ключа використовується як адреса в таблиці, що містить ключі і записи.

У файлі може бути тільки одна структура хешування або, інакше кажучи, одне хеш поле. Найпростішим прикладом загального класу хеш-функцій є тип розподіл / залишок (на деякий просте число, наприклад 13). Слід зазначити, що при хешуванні фізична послідовність записів майже завжди відрізняється від послідовності ключового поля. Основною вимогою до хеш-функції є рівномірний розподіл значення згортки, де одному значенню хеш-функції може відповідати одне значення ключа. Однак часто доводиться допускати зворотне. Якщо для декількох значень ключа виходить одна і та ж згортка, то створюється ланцюжка переповнення. Подібні ситуації називаються колізіями. Значення ключів, які мають одне і те ж значення хеш-функції, називаються синонімами. Головним обмеженням цього методу є фіксований розмір таблиці. Якщо таблиця заповнена занадто сильно або переповнена, то виникає дуже багато ланцюжків переповнення і втрачається головна перевага хешування: доступ до запису майже завжди за одне звернення до таблиці. Розширення таблиці вимагає її повної переробки на основі нової хеш-функції (зі значенням згортки більшого розміру). Отже, при використанні хешування як методу доступу необхідно прийняти два незалежних рішення.

- обрати хеш-функцію;
- обрати метод дозволу колізій.

Стратегії вирішення колізій. Розглянемо деякі досить поширені стратегії вирішення колізій.

1) Метод послідовного перебору При даній стратегії значення хеш-функції визначає не адреса розташування на носії запису, а деяку відправну точку для додаткового перегляду і пошуку. При збереженні деякої записи здійснюється пошук першої вільної для розміщення інформації області зберігання. Для вилучення цього запису доведеться виконати аналогічні дії: використовуючи метод послідовного перебору, пройти від отриманої за

допомогою хеш-функції стартової точки до необхідної записи, і лише потім її витягти.

2) Стратегія розв'язання колізій з областю переповнення При виборі цієї стратегії область зберігання розбивається на дві частини:

- основній область;
- область переповнення.

Для кожного нового запису обчислюється значення хеш-функції, яка визначає адресу її розташування, і запис заноситься в основну область відповідно до отриманого значення третьому хеш-функції. Якщо знову заноситься запис має значення функції хешування, таке ж, яке використовувала інший запис, вже наявна в БД, то новий запис заноситься в область переповнення на перше вільне місце, а в запису-синоніму, яка знаходиться в основний області, робиться посилання на адресу знову розміщеної записи в області переповнення. Якщо ж вже існує посилання в запису-синоніму, яка розташована в основній області, то тоді новий запис розташовується завжди на другому місці в ланцюжку синонімів, що істотно скорочує час розміщення нового запису. При такому алгоритмі час розміщення будь-якої нової записи становить не більше двох звернень до диска з урахуванням того, що номер першої вільної записи в області переповнення зберігається у вигляді системної змінної. При пошуку довільного запису також спочатку обчислюється значення її хеш-функції і зчитується перший запис в ланцюжку синонімів, яка розташована в основний області. Якщо шукана запис не відповідає першій в ланцюжку синонімів, то далі пошук відбувається переміщенням по ланцюжку синонімів, поки не буде виявлена необхідна запис. Швидкість пошуку залежить від довжини ланцюжка синонімів, тому якість хеш-функції визначається максимальною довжиною ланцюжка синонімів. Хорошим результатом може вважатися наявність не більше 10 синонімів в ланцюжку. При видаленні довільного запису спочатку визначається її місце розташування. Якщо видаляється є перший запис в ланцюжку синонімів, то після видалення на її місце в основний області заноситься другий запис в ланцюжку синонімів, при

цьому всі посилання на синоніми зберігаються. Якщо ж видаляється запис знаходиться в середині ланцюжка синонімів, то необхідно провести коригування покажчиків: у записі, що передує видаляється, в ланцюжку ставиться покажчик з видаляється записи. Якщо це останній запис в ланцюжку, то все одно механізм зміни покажчиків такий же, тобто в попередню запис заноситься ознака відсутності наступного запису в ланцюжку, який раніше зберігався в останньому записі.

3) Організація стратегії вільного заміщення При цій стратегії файловий простір не поділяється на області, але для кожного запису додається два покажчика: покажчик на попередню запис в ланцюжку синонімів і покажчик на наступний запис в ланцюжку синонімів. Відсутність відповідного посилання позначається спеціальним символом, наприклад нулем. Для кожного нового запису обчислюється значення хеш-функції, і якщо цю адресу вільний, то запис потрапляє на задане місце і стає першою в ланцюжку синонімів. Якщо адреса, відповідний отриманого значення хеш-функції, зайнятий, то за наявністю посилань визначається, чи є запис, розташована за вказаною адресою, першої в ланцюжку синонімів. Якщо так, то новий запис розташовується на першому вільному місці і для неї встановлюються відповідні посилання: вона стає другою в ланцюжку синонімів, на неї посилається перший запис, а вона посилається на наступну, якщо така є. Якщо запис, яка займає необхідне місце, не є першим записом в ланцюжку синонімів, значить, вона займає дане місце "незаконно" і при появі "законного власника" повинна бути переміщена на нове місце. Механізм переміщення аналогічний занесенню нового запису, яка вже має синонім, занесений в файл. Для цього запису шукається перше вільне місце, і коригуються відповідні посилання: в запису, яка є попередньою в ланцюжку синонімів для переміщуваної записи, заноситься покажчик на нове місце переміщуваної записи, покажчики ж в самій переміщуваної записи залишаються колишні. Після переміщення "незаконної" записи знову вноситься запис займає своє законне місце і стає першим записом в новій ланцюжку синонімів. Механізми видалення записів в чому аналогічні механізмам видалення в стратегії з областю переповнення.

5 Індексування

Хоча технологія хеширования і може дати високу ефективність, але для її реалізації не завжди вдається знайти відповідну функцію, тому при організації доступу до даних широко використовуються індексні файли. Основне призначення індексів полягає в забезпеченні ефективного прямого доступу до запису таблиці по ключу. Розрізняють індексований файл і індексний файл (див. рис. 22.7). Індексований файл - це основний файл, який містить дані відносіння, для якого створено індексний файл.

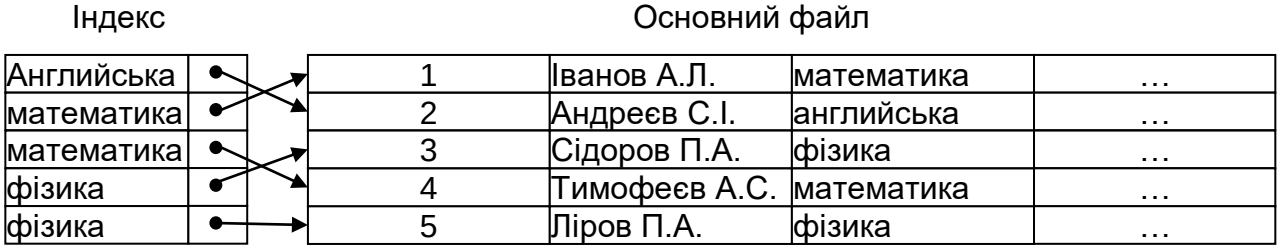


Рисунок 22.7 – Індексований і індексний файли

Індексний файл - це файл особливого типу, в якому кожен запис складається з двох значень даних і покажчика. Дані представляють поле, по якому проводиться індексація, а покажчик здійснює зв'язування з відповідним кортежем індексованого файлу. Якщо індексування здійснюється по ключовому полю, то індекс називається первинним. Такий індекс має властивість унікальності, тобто не містить дублікатів ключа. Зазвичай індекс визначається для одного відносини, і ключем є значення простого або складеного атрибута. Основна перевага використання індексів полягає в значному прискоренні процесу вибірки даних, а основний недолік - в уповільненні процесу оновлення даних. Дійсно, при кожному додаванні нового запису в індексований файл потрібно також додати новий індекс (а це додатковий час) в індексний файл. Тому при виборі поля індексування завжди важливо уточнити, який з двох показників важливіше: швидкість вибірки або швидкість відновлення. Індекси дозволяють:

– здійснювати послідовний доступ до індексувати файли відповідно до значень індексного поля для складання запитів на пошук наборів записів;

– здійснювати прямий доступ до окремих записів індексованого файлу на основі заданого значення індексного поля для складання запитів для заданих значень індексу;

– організувати запити, які не потребують звернення до індексувати файли, а лише які призводять до перевірки наявності даного значення в індексному файлі.

При виконанні багатьох операцій мовного рівня потрібно сортування відносин в відповідності зі значеннями деяких атрибутів, корисною властивістю індексу є забезпечення послідовного перегляду кортежів відносини в діапазоні значень ключа в порядку зростання або зменшення значень ключа. Залежно від організації індексного і основний областей розрізняють два типи файлів: індексного-прямі файли і індексного-послідовні файли.

Індексного-прямі файли У індексного-прямих файлах основна область містить послідовність записів однакової довжини, розташованих в довільному порядку, а індексний запис містить значення первинного ключа і порядковий номер запису в основний області, яка має дане значення первинного ключа. Оскільки індексні файли будуються для первинних ключів, однозначно визначають запис, то в індексного-прямих файлах для кожного запису в основних області існує тільки один запис з індексного області. Такий індекс називається щільним. Всі записи в індексному області впорядковані за значенням ключа, тому можна при-нити більш ефективні способи пошуку в упорядкованому просторі.

Найбільш ефективне алгоритмом пошуку на впорядкованому масиві є бінарний пошук. При цьому весь простір пошуку розбивається навпіл, і так як воно суворо упорядковано, то спочатку визначається, чи не є серединний елемент шуканим, а якщо немає, то дається оцінка в якій половині його треба шукати. Далі встановлена половина також ділиться навпіл і виробляються аналогічні дії, і так до тих пір, поки не буде виявлений шуканий елемент. В даному випадку двійковий алгоритм пошуку застосовується до індексного файлу, а потім шляхом прямої адресації відбувається звернення до основної області вже по конкретному номеру запису.

Операція додавання здійснює запис в кінець основної області. В індексному області при цьому проводиться занесення інформації так, щоб не порушувати впорядкованості. Тому вся індексна область файлу розбивається на блоки і при початковому заповненні в кожному блоці залишається вільна область (відсоток розширення) (див. рис. 22.8). Блок, до якого повинен бути занесений індекс, копіюється в оперативну пам'ять, де проводиться вставка нового запису, і змінений блок записується назад на диск. Природно, в процесі додавання нових записів відсоток розширення постійно зменшується. Коли зникає вільна область, виникає переповнення індексної області. У цьому випадку можливі два рішення: або перебудувати заново індексну область, або організувати область переповнення для індексного області, в якій будуть зберігатися не помістилися в основну область записи. Однак перший спосіб зажадає додаткового часу на перебудову індексної області, а другий збільшить час на доступ до довільної записи і зажадає організації додаткових посилань в блоках на область переповнення. Тому при проектуванні фізичної БД важливо заздалегідь якомога точніше визначити обсяги інформації, що зберігається, спрогнозувати її зростання і передбачити відповідне розширення області зберігання.

Рисунок 22.8 – Організація індексного-прямої адресації

При видаленні запису виникає наступна послідовність дій: запис в основний області позначається як віддалена, в індексному області відповідний індекс знищується фізичних, тобто записи індексного файлу, наступні за видаленим записом, переміщаються на її місце, і блок, в якому зберігався даний індекс, заново записується на диск. При цьому кількість звернень до диска для цієї операції таке ж, як і при додаванні нового запису.

Індексного-послідовні файли Якщо файли підтримуються в відсортованому стані з моменту їх створення, то для роботи з ними може бути використаний інший підхід побудови індексного файлу. Принципи внутрішнього упорядкування та блочності побудови таких файлів дозволяють зменшити кількість збережених індексів за рахунок того, що в індексному файлі не містяться покажчики на всі записи індексованого файлу. Таким чином в цьому випадку індекс виходить нещільним або розрідженим. Одним з переваг нещільні індексів є їх малий розмір у порівнянні з щільними індексами, тому що вони містять менше число записів. Це дозволяє переглядати вміст бази даних з більшою швидкістю. Індексна запис для таких файлів повинна містити: значення ключа першого запису блоку і номер блоку з цим записом. Тепер по заданому значенню первинного ключа в індексному області потрібно відшукати вже потрібний блок. Так як всі записи впорядковані, то значення першого запису блоку дозволяє швидко визначити, в якому блоці знаходиться шукана запис. Всі інші дії відходять в основний області (див. рис. 22.9). При переході до нещільного індексу час доступу зменшується практично в півтора рази. При такому підході новий запис повинна заноситися відразу в тре-буєм блок на потрібне місце. Дане занесення здійснюється в оперативній пам'яті, куди зчитується блок основної пам'яті, який внаслідок впорядкованості записів за значенням ключа повинен прийняти цей запис. Вміст ліченого блоку коригується, і потім він знов записується на диск на старе місце. Для додавання записів вже блок основної області повинен мати вільне місце. При внесенні нового запису індексна область не коригується. Знищення запису відбувається шляхом її фізичного видалення з основної області, при цьому індексна область

зазвичай не коригується, навіть якщо видаляється перший запис блоку. Однак слід зазначити:

– за допомогою одного нещільного індексу не можна виконати перевірку наявності деякого знання;

– в даному доглянутому файлі може бути, принаймні, один нещільний індекс, який організовується по полю, по якому цей файл впорядкований, а решта індекси обов'язково повинні бути щільними.

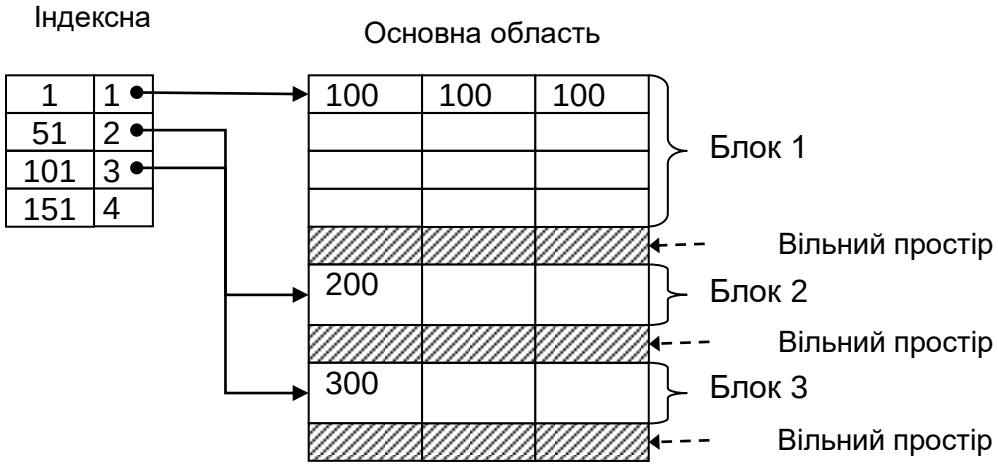


Рисунок 22.9 – Організація індексного-послідовної адресації

Організація індексів у вигляді Б-дерев

Структура типу Б-дерева є окремим випадком бінарного індексу деревовидного типу, яка використовується для побудови найбільш важливих і поширених індексів. Бінарні індекси мають в більшості випадків порівняно високою продуктивністю і тому в даний час їх використання передбачено майже у всіх СУБД, а деякі СУБД працюють тільки на основі такого індексу. Висока продуктивність бінарних індексів забезпечується тим, що при їх використанні вдається уникнути обов'язкового перегляду всього вмісту файлу згідно його фізичної послідовності, яке вимагає великих витрат часу. Справа в тому, що якщо індексований файл має великий розмір, то і його індекс також дуже великий і послідовний перегляд навіть тільки одного індексу займає чимало часу.

Побудова Б-дерев пов'язано з простою ідеєю побудови індексу над уже побудованим індексом. Дійсно, якщо вже побудований щільний (але може бути і нещільний, якщо в індексувати файли проведена кластеризація на основі

індексу) індекс для реальних даних, то сама індексна область може розглядатися як основний файл, над яким треба знову побудувати нещільний індекс. Записи всередині індексу згруповані по сторінках, а сторінки пов'язані в ланцюжок таким чином, щоб логічне впорядкування на основі індексу здійснювалося всередині першої сторінки, потім за допомогою фізичної послідовності записів всередині другої сторінки цієї послідовної ланцюжка і т. д. Таким чином, за допомогою набору послідовностей можна організувати швидкий послідовний доступ до індексованих даними. Потім над новим індексом будується наступний нещільний індекс і так до того моменту, поки не залишиться всього один індексний блок. Ця операція зазвичай застосовується тричі, оскільки створення великої кількості ієрархічних рівнів індексування потрібно для дуже великих файлів. При цьому індекс на кожному з рівнів буде нещільним по відношенню до нижнього індексованих рівню. Т.ч., на самому верхньому рівні такого індексу перебуває тільки один елемент структури (сторінка, що містить безліч записів), який називається кореневим.

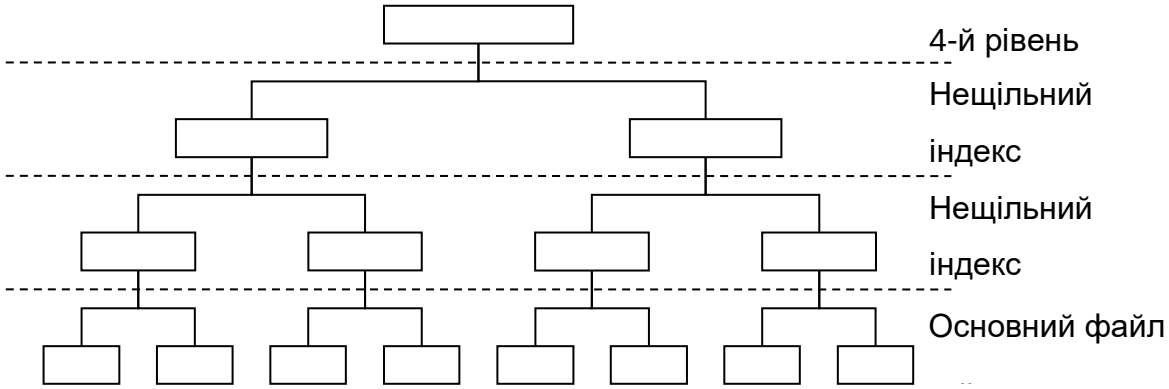


Рисунок 22.10 – Б-дерево

Набір індексів забезпечує швидкий безпосередній доступ до набору послідовностей (а значить, і до даних). В результаті такої побудови вийде деякий дерево (див. рис. 22.10), кожен батьківський блок якого пов'язаний з однаковою кількістю підлеглих блоків, число яких дорівнює числу індексних записів, що розміщуються в одному блоці. Кількість звернень до диска при цьому для пошуку будь-якого запису однаково і дорівнює кількості рівнів в побудованому дереві. Такі дерева називаються збалансованими тому, що шлях від кореня до будь-якого листа в цьому дереві однаковий. Структури типу Б-

дерева відносяться до ієрархічним системам. Незбалансованість роботи ієрархічних структур є небажаним їх властивістю, яке полягає в тому, що елементи з реальними даними можуть виявитися на різних рівнях і на різних відстанях від кореневого елемента. При роботі з такими структурами дуже складно оцінити час пошуку потрібного елемента. Структури типу Б-дерева дозволяють, використовуючи спеціальні, алгоритми, здійснювати збалансовану вставку або видалення значень. К. Дж. Дейт наводить такий короткий алгоритм вставки нового значення V в структуру типу Б-дерева порядку п. Він розрахований на вставку значення тільки лише в набір індексів, але може бути досить просто розширено для вставки запису з даними в набір послідовностей.

На найнижчому рівні набору індексів слід знайти елемент (припустимо, що це елемент N), з яким логічно пов'язане вставляється значення V. Якщо елемент N містить вільний про-странство, то значення V вставляється в нього, і на цьому процес завершується.

В іншому випадку (якщо вільного простору немає, т. Е. Доведеться створити ще один уро-вень) елемент N (припустимо, що він містить 2п індексних записів) розділяється на два елементи - N1 і N2. Позначимо символом S безліч з 2n + 1 значень, в якому 2п вихідних значень і одне нове значення V. Тоді п перших значень цієї логічної (вже впорядкованої) послідовності необхідно помістити в елемент N1, n останніх - в елемент N2, а середнє між ними значення W - в батьківський елемент P на більш високому структурному рівні. Згодом, під час здійснення пошуку значення U і досягненні елемента P, пошук буде перенаправлений в бік елемента N1, якщо U □ W, або в сторону елемента N2, якщо U> W.

Далі цей процес слід повторити для вставки середнього значення W в батьківський елемент P на більш високому структурному рівні.

У гіршому випадку процес поділу елементів структури може тривати аж до кор-невого елемента всієї структури з утворенням нового ієрархічного рівня.

Для видалення деякого значення слід застосувати аналогічний алгоритм, але тільки в зворотному порядку. А для зміни значення його можна видалити, а потім вставити нове.

Питання та завдання до контролю знань студентів

1) Які різновиди об'єктів бази даних для підтримки функціонування СУБД необхідно зберігати у зовнішній пам'яті?

2) Дайте характеристику сторінкової організації даних в СУБД.

3) Як здійснюється зберігання таблиць при сторінкової організації даних в СУБД?

4) Що собою являє технологія хеширования, і для чого вона використовується в струк-турах зберігання баз даних?

5) Які ситуації називаються колізіями? У чому полягає основне призначення індексів?

6) Який індекс називається щільним? Поясніть методології пошуку, вставки і видалення елемента основного файлу при такій організації індексу.

7) Опишіть технологію створення нещільного індексу і використання його при пошуку, вставці і видаленні елемента.

8) Чим забезпечується висока продуктивність бінарних індексів? Поясніть мето-довгий отримання Б-дерева.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 23

з навчальної дисципліни «Бази даних»

Тема Структура та можливості SQL

Мета заняття Дати характеристику двох складових частин мови SQL, необхідних для роботи з базами даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Історичні аспекти розвитку SQL.
2 Структура та можливості SQL.
3 Типи даних мови SQL.

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Історичні аспекти розвитку SQL.

Запропоновані Коддом реляційні мови обробки даних (реляційна алгебра та реляційне счислення) заклали теоретичні основи для розробки комерційних мов реляційних СУБД й є еталоном при тестуванні їхніх логічних можливостей.

Одним з мов, що з'явилися пізніше й використають закладений Коддом теоретичний базис, є SQL (Structured Query Language) - структурована мова запитів. Ця мова являє собою вдалу програмну розробку для маніпулювання даними реляційної моделі. У цей час він є найпоширенішим програмним продуктом такого роду.

Джерела мови SQL можна віднести до 1974 року, коли на чотири роки пізніше публікації Код-будинок своєї основної статті побачило світло визначення мови, що получили назву "Structured English Query Language", або скорочено SEQUEL, на базі якого був реалізований перший прототип реляційної СУБД фірми IBM System R. Згодом перероблена версія цієї мови одержала назву SQL і поступово в силу свого широкого поширення стала стандартом для мов маніпулювання даними в реляційної СУБД. Вихід вихідного варіанта стандарту мови SQL ставиться до 1987 року. Слеует відзначити, що цей варіант стандарту мови мав ряд недоліків і справедливо зазнавав критики. До числа таких недоліків відносили його надмірну надмірність, відсутність у мові найважливіших функцій підтримки цілісності й ряду інших конструкцій. В 1989 році ISO (Міжнародний комітет зі стандартизації), урахувавши висловлені зауваження на адресу попередньої версії стандарту мови, опублікував перший міжнародний стандарт мови SQL, а в 1992 році була прийнята нова, істотно перероблена версія міжнародного стандарту мови SQL, що одержала назву SQL/92 або SQL2. Для того щоб можна було розрізняти ранню й пізню версії мови, версії 1989 року привласнили назва стандарту SQL1. Переважна більшість доступних на ринку СУБД розроблялися з орієнтацією саме на з SQL1, але в даний момент основна частина виробників СУБД внесли зміни у свої продукти так, щоб вони більшою мірою задовольняли стандарту SQL2. В 1999 році з'явився новий стандарт мови, названий SQL3, у який включений цілий ряд розширень, що сприяють якісним серйозним перетворенням стандарту мови. В SQL3 уведені нові стандартні типи даних, користувачеві надається з конструювання складних структурованих типів даних, які дозволяють організувати об’єктно-орієтовну роботу з даними, уведені стандарти на події й тригери, і багато чого іншого. У

цей час ніхто не має наміру вважати процес розвитку стандарту мови SQL завершеним. Робота над стандартом триває. Фахівці докладають значних зусиль за рішенням ряду проблем, наприклад, створенню розширень мови, які дозволять здійснити взаємодія між різнорідними системами.

Сучасний стан справ в області створення й підтримки інформаційних систем характеризується наявністю поки єдиної стандартної мови для роботи з базами даних. Практично всі найбільші розроблювачі СУБД створюють свої продукти з використанням мови SQL і намагаються дотримуватися його стандарту. Додержуватися стандарту прагнуть не тільки розроблювачі СУБД, але і їхні споживачі. На це в тих й інших існують свої досить вагомі причини. З одного боку, жоден серйозний розроблювач, що проектує бази даних, не стане ігнорувати стандарт. Кожен автор інформаційної системи прагне до того, щоб його проект був розрахований на розширюваність і переносимість на інші платформи. Саме тому користувач СУБД, як правило, ігнорує додаткові можливості даного продукту, що не входять у стандарт, тому що система, що володіє властивістю переносимості, краще захищає інтелектуальну працю, вкладена в розробку. З іншого боку, зазначена вище причина визначає поводження й більшості постачальників програмного забезпечення, які в такій ситуації змушені випливати стратегії підтримки стандартів, оскільки в противному випадку користувачі просто не будуть такі продукти купувати.

На закінчення необхідно звернути увагу на деякі аспекти визначення стандарту ISO SQL, які не збігаються з окремими концепціями реляційної теорії Кодда. Так, у стандарті замість таких понять, як відношення, атрибут, кортеж використаються більше звичні для рядового користувача й не потребуючі додаткових пояснень терміни: таблиця, стовпець, рядок. У цій темі матеріал буде викладатися з використанням термінології стандарту мови. Крім того, у мові SQL допускається, що створена в результаті виконання операції select таблиця може містити дублюючі рядки, стовпці таблиці вважаються впорядкованими, її рядки також можуть бути наведені в упорядкований стан.

2 Структура та можливості 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 счислення, 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.

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 (див. табл. 23.1).

Таблиця 23.1 – 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>	Позиція, з якої починається

Функція	Результат
IN <другий рядок>)	входження 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 якої 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

3 Типи даних мови SQL

У мові SQL1 визначений ряд типів даних, які представлені в таблиці 23.2.

Таблиця 23.2 – 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 або 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 (див. табл. 23.3).

Таблиця 23.3 – 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)	Рядок бітів постійної довжини
BIT VARYING (n)	Рядок бітів 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 необов’язковим завданням знаку числа й десятковою крапкою;

− константи із плаваючої 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.

2 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 використаються для зберігання чисел. У нашому прикладі ми використали типи int (ціле число) і float (число із плаваючої коми).

Вони представляють два підтипи числових типів: точні числові типи й наближені числові типи.

Числові типи можуть характеризуватися максимальною довжиною, а типи із плаваючої 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 указати тільки загальну довжину або ж указати як довжину, так і число десяткових розрядів.

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

Ключове слово UNSIGNED указує, 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 використаються для зберігання точних значень із плаваючої коми й звичайно використаються для того, щоб запам'ятовувати грошові значення. Вони мають той же діапазон, що й числа із плаваючої 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 BI й BOOL.
2) SHA.LLINT 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 плаваючої коми зі звичайною точністю. Вони можуть представляти позитивні числа в діапазоні 1,18* 10-38 -3,40* 1038 й аналогічний діапазон негативних чисел.

DOUBLE

Це — числа із плаваючої коми з подвійною точністю. Синонімами DOUBLE є REAL й DOUBLE PRECISION. Вони можуть представляти позитивні числа в діапазоні 2,23*10-308 -1,80* 10308 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 використається 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(l). 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, NUMERI 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 використається 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 указується довжина, наприклад VARCHAR (10). Допускаються значення від 0 до 255.

TEXT, BLOB й їхньої варіації

Типи TEXT використаються для зберігання більше довгих фрагментів тексту, чим допускається типами CHAR й VARCHAR. Абревіатура 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 використається YEAR (4). YEAR (2) представляє значення з 1970 по 2069.

Питання та завдання до контролю знань студентів

1) Які дві складові частини, необхідні для роботи з базами даних, присутні в мові SQL?
2) Поясните їхнє призначення.
3) Які стандартні вбудовані функції використовуються в запитах?
4) Який ряд типів даних визначений у мові SQL?
5) Які додаткові типи даних розширюють можливості мови?
6) Які константи визначені у стандарті мови?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 24

з навчальної дисципліни «Бази даних»

Тема Створення бази даних, таблиць бази даних. Введення даних в таблиці

Мета заняття Розглянути роботу операторів створення та видалення бази даних, створення таблиць та індексів

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Створення та видалення баз даних.

1.1 Введення

1.2 Створення бази даних

1.3 Створення таблиць

1.4 Створення індексів

2 Видалення баз даних, таблиць й індексів

3 Зміна структури таблиць

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Створення та видалення баз даних.

1.1 Введення

Для прикладів ми будемо використати базу даних, про яку йшла мова в методичних рекомендаціях до лабораторних робіт - це база 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)

Для створення баз даних, таблиць й індексів ми будемо використати мову визначення даних. Що це значить? Абревіатура SQL розшифровується як Structured Query Language (мову структурованих запитів), і саме ця мова ми будемо використати для створення реляційних баз даних і виконання запитів до них. Він складається із двох семантично відособлених частин: мови визначення даних (Data Definition Language - скорочено DDL), за допомогою якого створюються структури бази даних, і мови маніпулювання даними (Data Manipulation Language - скорочено DML), за допомогою якого виконуються запити до бази даних.

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 A5СЩ255).

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.

1.2 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;

Переконатися 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 використається 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.

1.3 Створення таблиць

Для створення таблиць у базі даних organization використається оператор SQL

CREATE TABLE.

У своїй типовій формі цей оператор виглядає так:

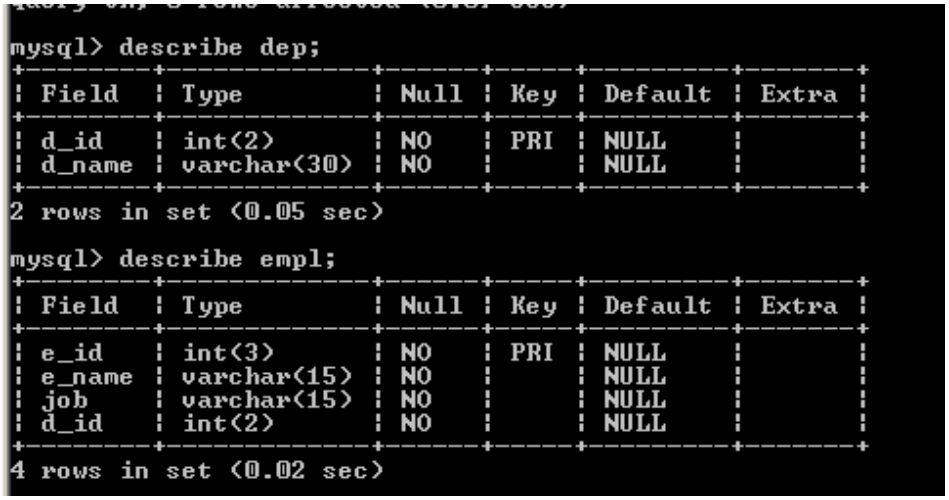
create table *ім'я_таблиці* (*визначення таблиці*) [type=*тип_таблиці*] ;

Як бачите, необхідно почати зі слів create table (створити таблицю), за яких повинне випливати 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_стовпця тип [not null | null] [default *значення*]

[auto_increment] [primary key] [*визначення_посилання*]

або primary key (*ім'я_стовпця_індексу*, . . .)

або key [*ім'я_індексу*] (*ім'я_стовпця_індексу*, . . .)

або index [*ім'я__індексу*] (*ім'я_стовпця_індексу*, . . ,)

або unique [index] [*ім'я_індексу*] (,*ім'я_стовпця_індексу*, . . .)

або fulltext [index] [*ім'я_індексу*] (,*ім'я_стовпця_індексу*, . . .)

або [constraint символ, foreign key [*ім'я_індексу*]

(ім'я_стовпця_індексу, . . .) [*визначення_посилання*]

або CHECK (*вираження*)

Розглянемо опції, зазначені в цій загальній формі.

Ключове слово TEMPORARY використається для створення таблиць, які будуть існувати тільки в поточному сеансі роботи з базою даних й які будуть автоматично вилучені, 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 же 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 випливати 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. Індекси автоматично створюються для стовпців, оголошених як PRIMARY KEY.

4) Оголосити стовпець первинним ключем таблиці за допомогою вираження PRIMARY KEY.

5) Оголосити стовпець зовнішнім ключем, використовуючи, як у попередньому прикладі, вираження REFERENCES.

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

7) Щоб створити первинний ключ, що складається з декількох стовпців, як у прикладі, необхідно вказати вираження PRIMARY KEY, за яким у дужках, через кому, випливають 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.

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

9) UNIQUE використається для вказівки того, що відповідний стовпець повинен містити унікальні значення. Стовпці UNIQUE теж будуть індексовані.

10) FULLTEXT використається для створення повнотекстових індексів на основі стовпців типу TEXT, CHAR або VARCHAR. Повнотекстові індекси можна використати тільки з таблицями MyiSAM.

11) 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.

1.4 Створення індексів

Звичайно всі необхідні індекси створюються при створенні таблиці. Всі стовпці, оголошені як PRIMARY KEY, KEY, UNIQUE або INDEX, будуть індексовані.

Іноді з'ясовується, що багато запитів використають значення не індексованого стовпця, - у цьому випадку за допомогою оператора CREATE INDEX можна додати новий індекс.

Цікаво, що перед виконанням оператор CREATE INDEX перетвориться в оператор ALTER TABLE. Оператор ALTER TABLE можна використати й для створення індексів, і для рішення багатьох інших завдань. Можна, наприклад, додати індекс до таблиці Empl так:

```
create index name on empl(e_name);
```

У результаті буде створений індекс із ім'ям name на основі значень поля name з таблиці empl.

Опцій в оператора create index не занадто багато. Перед ключовим словом index може бути присутнім UNIQUE, що вимагає унікальності обмеження. Перед словом index можна також розмістити ключове слово FULLTEXT, якщо необхідно створити повнотекстовий індекс для таблиці MylSAM.

Ще одна опція ставиться до індексів на основі char й varchar і дозволяє створити індекс по першим декількох символах поля. Це можна зробити, указавши в дужках після імені индексируемого стовпця число символів, які варто використати, наприклад:

```
create index part_name on empl(e_name(5));
```

Це робиться тому, що індекси на основі текстових даних працюють не занадто ефективно в порівнянні з індексами на основі числових, а індексація по першим декількох символах поліпшує продуктивність.

2 Видалення баз даних, таблиць й індексів

Тепер, коли ви знаєте, як створювати бази даних, таблиці й індекси, корисно також довідатися, як їх видалити. Ключовим словом, що підходить для цієї мети, є DROP.

Можна видалити всю базу даних разом з усім її вмістом, надрукувавши наступне (на даному етапі ми не рекомендуємо вам робити цього!):

```
drop database organization;
```

Окрему таблицю можна видалити за допомогою оператора DROP TABLE:

```
drop table assignment;
```

Загальна форма оператора DROP TABLE виглядає так:

drop [temporary] table [if exists*]* *ім'я_таблиці* [, *ім'я_таблиці*,...]

Для видалення тимчасових таблиць використається ключове слово TEMPORARY. За допомогою одного оператора можна видалити й кілька таблиць відразу, указавши список їхніх імен, розділивши імена комами. Необов'язкове вираження IF EXISTS тут працює точно так само, як й в операторі DROP DATABASE.

За допомогою оператора DROP INDEX можна видалити індекс:

drop index part_name on empl;

Як бачите, необхідно вказати й таблицю, індекс якої ви з видалити.

3 Зміна структури таблиць

Крім створення й видалення таблиць, буває потрібно змінити структуру вже існуючої таблиці. Це можна зробити за допомогою оператора ALTER TABLE, Оператор ALTER TABLE має дуже багато варіантів, які можна використати для зміни структури таблиць.

Оператор ALTER TABLE є винятково гнучким, тому він має величезну безліч додаткових ключових слів. Загальна форма оператора з посібника з MySQL виглядає так:

ALTER [IGNORE] TABLE *ім'я_таблиці специфікація*_alter

[,*специфікація*_alter ...]

специфікація_alter

ADD [COLUMN] *визначення*_create

[FIRST | AFTER *ім'я_стовпця*]

або ADD [COLUMN] (*визначення*_create, *визначення*_create, . . .)

або ADD INDEX [ім'я_індексу] (*ім'я_стовпця*_індексу, . . .)

або ADD PRIMARY KEY (*ім'я_стовпця_індексу*, . . .)

або ADD UNIQUE [*ім'я_індексу*] (*ім'я_стовпця_індексу*, . . .)

або ADD FULLTEXT [*ім'я_індексу*] (*ім'я_стовпця_індексу*, . . .)

або ADD [CONSTRAINT *символ*] FOREIGN KEY [*ім'я__індексу*]
(*ім'я_стовпця_індексу*, . . .) [*визначення_посилання*]

або ALTER [COLUMN] *ім'я_ стовпця*

{SET DEFAULT *літерал* | DROP DEFAULT}

або CHANGE [COLUMN] *старе_ім'я_стовпця визначення*_create

[FIRST | AFTER *ім'я_стовпця*]

або MODIFY [COLUMN] *визначення*_create

[FIRST | AFTER *ім'я_стовпця*]

або DROP [COLUMN] *ім'я_стовпця*

або DROP PRIMARY KEY

або DROP INDEX *ім'я_індексу*

або DISABLE KEYS

або ENABLE KEYS

або RENAME [TO] *нове_ім'я_таблиці*

або ORDER BY *ім'я_стовпця*

або *опції_таблиці*

Більшість із цих варіантів самоочевидні або відповідають вираженням оператора CREATE TABLE (як, наприклад, ADD PRIMARY KEY). Ми коротенько обговоримо лише ті опції, які не здаються очевидними.

Вираження CHANGE й MODIFY означають те саме : вони дозволяють змінити визначення стовпця або його місце в таблиці.

DROP COLUMN видаляє стовпець із таблиці, тоді як DROP PRIMARY KEY й DROP INDEX видаляють тільки індекс, пов'язаний із зазначеним стовпцем.

Вираження DISABLE KEYS змушує MySQL зупинити відновлення індексів для таблиці MyiSAM, а вираження ENABLE KEYS знову включає відновлення індексів.

Вираження RENAME дає можливість змінити ім'я таблиці.

Вираження ORDER BY розмістить рядка в зміненій таблиці в зазначеному порядку, подібно вираженню ORDER BY в операторі SELECT, що буде розглядатися пізніше. Цей порядок не буде підтримуватися в міру заповнення таблиці новими даними.

Список опції_таблиці дозволяє вказати такої ж опції таблиці, які припустимі наприкінці оператора CREATE TABLE, розглянутого вище.

Питання та завдання до контролю знань студентів

1) Яке з наступних імен неприпустимо для таблиць MySQL?
   а. employee;
   б. select;
   в. employee . skill;
   г. employeeSkills.
2) Які з наступних тверджень відносно CHAR й VARCHAR коректні?

а. стовпець CHAR, незалежно від його вмісту, завжди займає той самий обсяг дискового простору;

б. значення VARCHAR доповнюються пробілами, коли зберігаються на диску;

в. стовпець CHAR у середньому займає менше місця, чим аналогічний стовпець VARCHAR;

г. стовпець VARCHAR, незалежно від його вмісту, завжди займає той самий обсяг дискового простору.

3) Перш ніж створювати таблиці в базі даних, необхідно:
   а. створити індекси для таблиці;
   б. створити цю базу даних;
   в. створити цю базу даних і вибрати її для використання;
   г. створити всі стовпці таблиці.
4) Який з наступних операторів CREATE TABLE синтаксично коректний?

а. create table department

```
departmenti int not null auto_increment
primary key,
name varchar(20)
type=InnoDB;
```

б. create table department type=InnoDB

```
(departmenti int not null auto_increraent
primary key,
```

name varchar(20));

в. create department (

departmenti int not null auto_increment

primary key,

name varchar(20)

) type=InnoDB;

г.

create table department

(departmenti int not null auto_increment

primary key,

name varchar(20)

) type=InnoDB;

5) Щоб видалити базу даних dbname разом з усім її вмістом, варто ввести

а. drop ail tables on dbname;

б. drop database dbname;

в. drop dbname;

г. delete database dbname;

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 25

з навчальної дисципліни «Бази даних»

Тема Введення даних в таблиці. Додавання, оновлення та видалення рядків таблиці.

Мета заняття Дати характеристику двох складових частин мови SQL, необхідних для роботи з базами даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Додавання даних в таблиці.

2 Видалення даних з таблиць.

3 Оновлення даних в таблицях.

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Додавання даних в таблиці.

Мова обробки даних (DML) включає три операції поновлення: INSERT (вставка), UPDATE (зміна) і DELETE (видалення). Розглянемо синтаксис цих

операторів в дещо спрощеному вигляді в порівнянні зі стандартом SQL.

Оператор введення даних INSERT має наступний синтаксис:

INSERT INTO ім’я_таблиці [(<список стовпців>)]

VALUES (<список значень)

Такий синтаксис дозволяє ввести лише один рядок у таблицю. Наприклад, введемо нового студента до таблиці S2 БД Сесія:

INSERT INTO S2 (ПІБ, Група)

VALUES ('Сидоров П.П.', '02-КТ-21')

Завдання списку стовпців необов'язкове тоді, коли вводиться рядок із завданням значень усіх стовпців. При такому введенні передбачається, що інформація буде вводитися в порядку, в якому вони описані в операторі CREATE TABLE. Так як у прикладі вводиться повний рядок, то можна не задавати список стовпців, обмежитися тільки завданням переліку значень, в цьому випадку оператор введення виглядатиме таким чином:

INSERT INTO S2

VALUES ('Сидоров П.П.', '02-КТ-21')

Якщо стовпець при описі таблиці має ознаку NOT NULL, то оператор INSERT повинен обов'язково містити дані для введення кожного рядка цього стовпця. Тому, якщо в таблиці для всіх стовпців введення обов'язкове, то кожен рядок, що вводиться, повинен містити повний перелік введених значень, а вказівку імен стовпців в цьому випадку можна опустити. У той же час, якщо є хоча б один необов'язковий стовпець, у який не вводиться значення, завдання списку імен стовпців обов'язкове. Між списком імен стовпців та списком значень має бути така відповідність:

− кількість елементів в обох списках має бути однаковою;

− між положенням елементів у списках має бути сувора відповідність, що визначається зліва направо: перший елемент одного списку відповідає першому елементу другого списку тощо;

− типи даних відповідних елементів списків повинні бути однакові та належати до одного й того самого домену.

У набір значень можуть бути включені спеціальні функції та вирази.

Обмеження тут є те, що значення цих функцій повинні бути визначені на момент введення даних. Оператор введення даних дозволяє ввести відразу рядок рядків, якщо їх можна вибрати з іншої таблиці. У разі синтаксис оператора введення визначається так:

INSERT INTO ім’я_таблиці [(<список стовпців>)] SELECT ...

Пропозиція SELECT може бути будь-якого допустимого йому виду. Усі обмеження введення, перелічені вище, повинні виконуватися суворо і в цьому випадку. Допустимо, що є таблиця STUDENT, де вказані основні дані про студентів, які мають наступний склад: STUDENT (ПІБ, Група, Адреса, Телефон, Дата народження). Тоді можна помістити з неї в таблицю S2 кілька рядків одним оператором:

INSERT INTO S2 (ПІБ, Група) SELECT (ПІБ, Група) FROM STUDENT

2 Видалення даних з таблиць

Оператор видалення даних дозволяє видалити один або кілька рядків з таблиці відповідно до умов, які задаються для рядків, що видаляються. Синтаксис оператора DELETE наступний:

DELETE FROM ім'я_таблиці [WHERE умови_відбору]

Умови відбору визначають, які рядки мають бути видалені. Якщо умови відбору не задаються, то з таблиці видаляються всі рядки, що існують у ній. Однак це не означає, що видаляється вся таблиця. Початкова таблиця залишається, але вона залишається порожньою, незаповненою. Наприклад, якщо нам треба видалити результати минулої сесії, то ми можемо видалити всі рядки із відношення S1 наступним оператором:

DELETE FROM S1

В результаті виконання даного оператора в БД збережеться тільки опис цієї таблиці, в яку можна буде помістити нову інформацію. Умови відбору частини WHERE мають той самий вигляд, що й умови фільтрації в операторі SELECT. Ці умови визначають, які рядки вихідного відношення будуть видалені. Наприклад, виняток із якоїсь причини студента Крісс Р.О. з таблиці S2 можна виконати оператором:

DELETE FROM S2 WHERE ПІБ = 'Кріс Р.О. '

У частині WHERE може бути вбудований запит. Однак слід звернути увагу на той факт, що всі операції маніпулювання даними пов'язані з поняттям цілісності бази даних і не здійсненні, навіть якщо синтаксично вони написані правильно. Наприклад, при виконанні операції DELETE, що включає складний підзапит, в підзапиті не можна згадувати таблицю, з якої видаляються рядки, так як буде порушена цілісність БД і тому СУБД відкине такий підзапит.

3 Оновлення даних в таблицях

Операція оновлення даних UPDATE потрібна тоді, коли потрібно змінити вміст бази даних. Цей оператор, як і інші оператори оновлення інформації БД, застосовується до однієї конкретної таблиці і має такий формат:

UPDATE ім'я таблиці

SET ім'я_стовпця1 = нове_значення1 [, ім'я_стовпця2 = нове значення2...]

[WHERE умова_відбору]

Тут у пропозиції UPDATE вказується ім'я таблиці, що оновлюється, у пропозиції SET вказуються імена стовпців і нові дані. Нові дані повинні бути сумісні з тими даними, які вони мають замінити. Частина WHERE є необов'язковою, як і в операторі DELETE. Вона відіграє ту саму роль, що й у операторі DELETE, — дозволяє відібрати рядки, яких буде застосована операція модифікації. Якщо умова відбору не визначається, то операція модифікації буде застосована до всіх рядків таблиці. Розглянемо операцію поновлення даних таблиці бази даних НДР. Припустимо, що вирішено усі нарахування фахівцям збільшити на 10%. Операція оновлення інформації у зв'язку з цим буде виглядати так:

UPDATE R3 SET Нарахування = Нарахування * 1.1

У тому випадку, коли модифікацію інформації необхідно проводити вибірково, потрібно використання пропозиції WHERE. Припустимо, що у БД Сесія слід зробити зміна даних, оскільки студентка Думська М.Т. перездала іспит з фізики та отримала оцінку "відмінно" замість "задовільно". Для вирішення поставленого завдання необхідно виконати таку операцію:

UPDATE S1 SET S1. Оцінка = 5

WHERE S1.ПІБ = 'Думська М.Т.' AND S1.Дисципліна = 'ФІЗИКА'

Операція модифікації, як і операція видалення, може використовувати складні підзапити.

Питання та завдання до контролю знань студентів

1) Інструкція UPDATE – це:
   а) вибірка даних з бази
   б) вставка даних до таблиці
   в) відновлення (зміна) даних у таблиці
   г) видалення даних з таблиці
   д) знищення таблиці.
2) Інструкція DELETE – це:
   а) вибірка даних з бази
   б) вставка даних до таблиці
   в) відновлення (зміна) даних у таблиці
   г) видалення даних з таблиці
   д) знищення таблиці.
3) Інструкція INSERT – це:
   а) вибірка даних з бази
   б) вставка даних до таблиці
   в) відновлення (зміна) даних у таблиці
   г) видалення даних з таблиці
   д) знищення таблиці.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 26

з навчальної дисципліни «Бази даних»

Тема Формування простих запитів MySQL: вибір даних з довільної таблиці.

Мета заняття Пояснити принцип формування простих запитів за допомогою оператора SELECT

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Прості запити MySQL

1.1 Можливості оператора SELECT

1.2 Відбір стовпців

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

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 довідаєтеся, як витягати дані

з бази даних.

У даній лекції ми 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.

У цій лекції порозуміється, як витягати дані з однієї таблиці. Ми розглянемо наступні питання.

1) Огляд можливостей оператора SELECT.
2) Прості запити.
3) Відбір 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.

Спочатку ми розглянемо загальну форму оператора SELECT.

1.2 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]
[LIMITмежі];
```

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 тім 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 Відбір стовпців

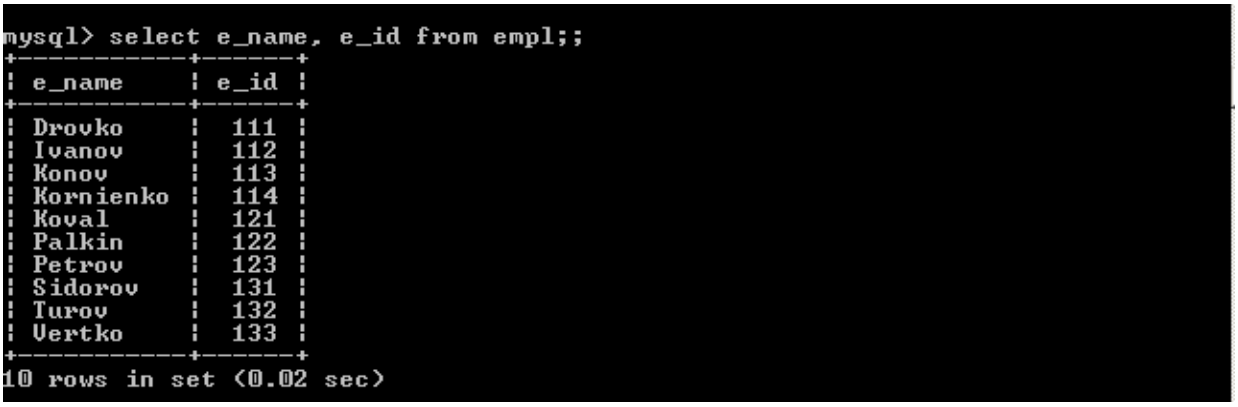
```
mysql> load data infile 'd:\\empl.txt' into table empl
Query OK, 10 rows affected (0.05 sec)
Records: 10  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from empl;
+------+-----------+-----------------+------+
| e_id | e_name    | job             | d_id |
+------+-----------+-----------------+------+
|  111 | Drovko    | AdministratorDB |   12 |
|  112 | Ivanov    | System admin    |   11 |
|  113 | Konov     | ProgramTs       |   14 |
|  114 | Kornienko | ProgramTs       |   14 |
|  121 | Koval     | ProgramTs       |   13 |
|  122 | Palkin    | ProgramTs       |   14 |
|  123 | Petrov    | ProgramTs       |   13 |
|  131 | Sidorov   | AdministratorDB |   12 |
|  132 | Turov     | System admin    |   11 |
|  133 | Vertko    | ProgramTs       |   13 |
+------+-----------+-----------------+------+
10 rows in set (0.00 sec)
```

Першим обмеженням, що ми можемо використати, є вказівка набору стовпців, що повертають. 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 із зазначені у вигляді списку значень, розділених комами.

Наприклад, що випливає з вибирає тільки значення полів e_id й f_name з таблиці empl:

select e_name, e_id from empl;

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



Як бачите, запит повернув значення тільки двох зазначених нами стовпців, Зверніть увагу на те, що у висновку 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, уикористовуючи для цього нотацію виду база_данных. таблиця.

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

select organization. empl.e_name from empl;

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

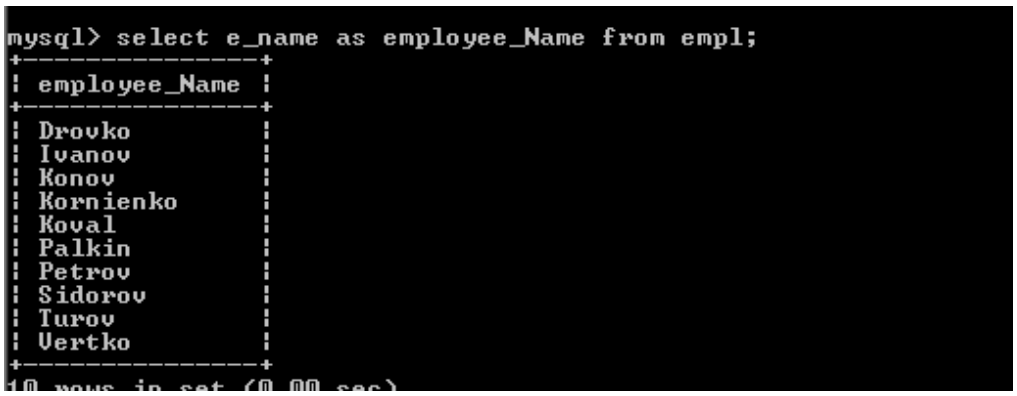
Тепер ми повинні обговорити концепцію псевдонімів для імен стовпців і таблиць.

В операторі SELECT можна перейменувати стовпці або призначити ім'я утримуючим 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:



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

У цьому конкретному прикладі користь від використання псевдоніма не занадто велика. Ви зможете оцінити переваги використання псевдонімів тоді, коли ми почнемо 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 корисної, 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;
```

і

```
select  e.e_name
from  emple;
```

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.

Питання та завдання до контролю знань студентів

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

2 Який ряд типів даних визначений у мові SQL1?

3 Дайте характеристику різних категорій операторів, які містить у собі мова SQL.

4 Поясните порядок проходження фраз в операторі SELECT. Які фрази в операторі є обов'язковими, а які - немає?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 27

з навчальної дисципліни «Бази даних»

Тема Запити та вбудовані функції MySQL

Мета заняття Розглянути роботу опцій оператора SELECT, за допомогою яких можна робити відбір стовпців та рядків таблиці, видалення повторень в рядках таблиці

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Відбір рядків за допомогою WHERE.

2 Видалення повторень за допомогою DISTINCT.

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Відбір рядків за допомогоюWHERE

Ми навчилися вибирати всі дані з таблиці або з конкретних її стовпців. Тепер ми з'ясуємо, як вибирати потрібні рядки. Це може придатися, 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 цікавимося.

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

3) Варіанти > (більше), < (менше), >= (більше або дорівнює) і <= (менше або дорівнює).

4) IS NULL й IS NOT NULL які використаються 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 використаються 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 операторів 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 виразу 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.

2 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     |
| Programer        |
| Programer        |
| Programer        |
| Programer        |
| Programer        |
| AdministratorDB  |
| System admin     |
| Programer        |
+------------------+
10 rows in set (0.00 sec)
```

Як бачите, значення Programer 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     |
| Programer        |
+------------------+
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 оманний. 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.

Питання та завдання до контролю знань студентів

1) Дайте характеристику різних категорій операторів, які містить у собі мова SQL.

2) Поясните порядок проходження фраз в операторі SELECT. Які фрази в операторі є обов'язковими, а які - немає?

3) Поясните порядок використання в стандарті мови SQL агрегатних функцій. Які обмеження на них накладаються?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 28

з навчальної дисципліни «Бази даних»

Тема Підсумовування й угруповання даних.

Мета заняття Розглянути роботу опцій оператора SELECT, за допомогою яких можна групування даних таблиці та обмеження результатів пошуку даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Використання GROUP BY

2 Сортування результатів пошуку за допомогою ORDER BY

3 Обмеження результатів пошуку за допомогою LIMIT

4 Використання агрегатних функцій

5 Опції оператора SELECT. Доповнення

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 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 згадуємо 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 | Programer       |
|        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 убутному 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 | Programer       |
|        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 починаючі користувачі SQL часто плутають вираження WHERE й HAVING. Вираження WHERE використається в запиті для перевірки умов, що стосуються окремих рядків, а умови HAVING застосовуються до груп.

2 Сортування 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. Сортувати при цьому можна як по зростанню (позначається ASC), так і по убуванню (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.

3 Обмеження результатів пошуку за допомогою LIMIT

Вираження LIMI використається для обмеження числа й діапазону рядків, що повертають запитом. Розглянемо, наприклад, що випливає 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 змішання (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 повертав рядка 0-4, а в другому вибиралися рядки 5-7. Якщо як другий параметр указати -1, запит поверне рядка, починаючи з рядка, певної зсувом, до кінця таблиці.

Вираження LIMIT звичайно використається 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.

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). Основні агрегані функції представлені в таблиці 28.1.

Таблиця 28.1 – Агрегатні функції

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 під запит, що повертає не значення, а рядок, але користь від такого під запита досить обмежена, тому відповідний приклад ми не приводимо.

5 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 JOI]

[SQL_SMALL_RESULT] [SQL_BI_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], . . .]

[LIMI [зсув,] рядка | рядка OFFSET зсув]

[PROCEDURE ім'я_процедуры(список_аргументів)]

[FOR UPDATE Й LOCK IN SHARE MODE]]

Більшість із зазначених виражень нам уже знайомо. А тепер коротенько розглянемо ті з них, які ми ще не обговорювали.

1) Вираження STRAIGHT JOI 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 JOI у вираженні WHERE, як було зроблено в розглянутому вище прикладі. Це - розширення ANSI SQL.

2) Опції SQL_SMALL_RESULT, SQL_BI_RESULT й SQL_BUFFER_RESULT призначені для настроювання параметрів оптимізації. SQL_SMALL_ RESULT й SQL_BI_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 кэшировать результати. (Ще одне розширення ANSI SQL.)

4) SQL_CALC_FOUND_ROWS передбачається використати з вираженням LIMI. Ця опція змушує MySQL з'ясувати, скільки рядків повинне повернутися у випадку відсутності вираження LIMI. 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 LIMI.

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 використається 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 використає 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

Питання та завдання до контролю знань студентів

1) У чому полягає суть операції угруповання? Які умови її використання?
2) 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;

3) Який з наступних запитів вибере всіх програмістів з таблиціempl?

а) 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;

4) 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;

5) Не допускається використати псевдоніми

а) для стовпців;

б) для таблиць;

в) у вираженні WHERE;

г) у вираженні SELECT.

6) Якщо необхідно за допомогою запиту повернути рядка 15-20,

коректним вираженням LIMIT буде

а) LIMIT 15, 20

б) LIMIT 14, 19

в) LIMIT 14, 6

г) LIMIT 15, 5

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 29

з навчальної дисципліни «Бази даних»

Тема Створення під запитів.

Мета заняття Показати на прикладах правила створення під запитів похідних таблиць, під запитів з одним значенням, під запитів в логічних виразах

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Створення підзапитів

1.1 Підзапити похідних таблиць

1.2 Під запити з одним значенням

1.3 Під запити в логічних виразах

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Створення підзапитів

Під запит - це запит усередині іншого запиту, тобто запит, у якому використаються результати іншого запиту. Іноді під запити називають вкладеними запитами. Під запити є новою можливістю в MySQL 4.1. Цієї можливості користувачі чекали давно. Під запити 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 під запитів:

– під запити похідних таблиць;

– під запити-вирази.

Під запити-вирази застосовуються у вираженні WHERE оператора SELECT, Вони бувають двох типів:

– під запит, що повертають одне значення або рядок;

– під запит, використовувані в логічному вираженні.

Ми розглянемо приклади під запитів обох зазначених типів.

1.1 Підзапити похідних таблиць

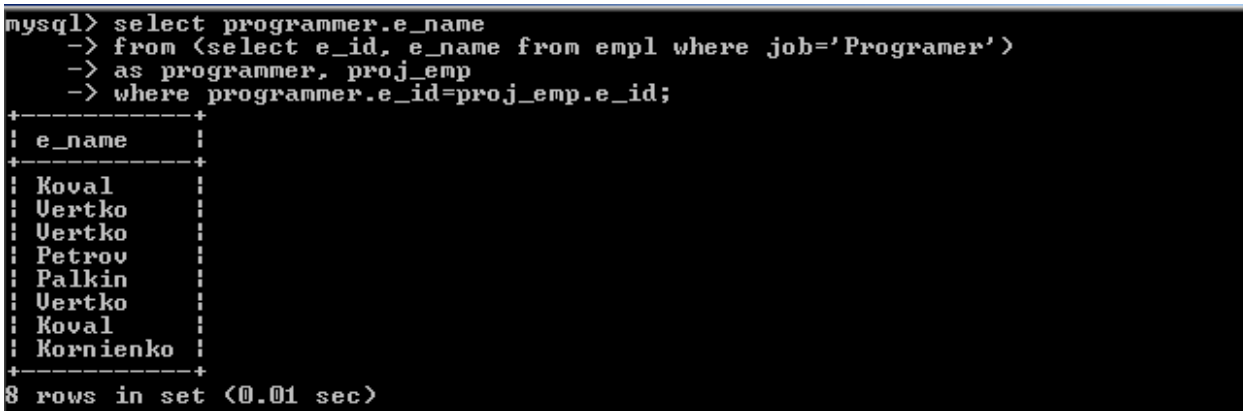
Під запити похідних таблиць дають можливість указати запит у вираженні FROM іншого запиту. По суті це дозволяє створити тимчасову таблицю й додати її в запит. Наприклад, розглянемо наступний простий запит:

```
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 (selecte_id,   e_name from empl where job=' Programer')
as programmer,
proj_emp
where programmer .e_id = proj_emp .e_id;
```

У цьому випадку ми використаємо під запит (selecte_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 д будь-який іншої. У цьому випадку ми використаємо цю таблицю для того, щоб з'ясувати, хто із програмістів працював над виконанням проектів, і одержуємо наступний результат:



1.2 Підзапити з одним значенням

Як й у попередньому розділі, ми починаємо із простого запиту

select max (royalty_share) from assignment;

Він повертає одне значення, що представляє собою максимальну процентну ставку, що працівник одержить за виконання завдання по проекті. 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 під запитів з одним значенням.

Можна піти далі й використати даний запит у рамках іншого.

Під запити з одним значенням повертають одне значення стовпця й звичайно використаються для порівняння. Розглянемо, наприклад, що випливає запит:

select e.e_id, e.e_narne

from empl e, proj_emp a

where e.e_id = a.e_id

and a. royalty_share = (select max (royalty_share) from proj_emp);

Тут ми намагаємося знайти той, кого можна назвати "ломовим конем" компанії: хто з службовців одержав більше всіх грошей за виконання завдання по проекті? От результат виконання такого запиту:

```
mysql> select e.e_id, e.e_name
    -> from empl  e,  proj_emp  a
    -> where e.e_id=a.e_id
    -> and a.royalty_share=(select max(royalty_share) from proj_emp);
+------+--------+
| e_id | e_name |
+------+--------+
|  112 | Ivanov |
+------+--------+
1 row in set (0.01 sec)
```

Можна також створити під запит, що повертає не значення, а рядок, але користь від такого під запита досить обмежена, тому відповідний приклад ми не приводимо.

1.3 Підзапити в логічних виразах

Під запити в логічних виразах використаються 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 використається для перевірки приналежності до деякої безлічі 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 JOI. 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 from empl
    -> where e_id not in(select e_id from proj_emp);
+---------+
| e_name  |
+---------+
| Konov   |
| Sidorov |
+---------+
2 rows 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  |   12 |
| Ivanov  |   11 |
| Sidorov |   12 |
| Turov   |   11 |
+---------+------+
4 rows in set (0.02 sec)
```

Ключове слово EXISTS працює лише небагато по-іншому в порівнянні зі словом IN. При використанні EXISTS ми насправді використаємо в під запиті дані деякого зовнішнього запиту. Такий запит іноді називають зв'язаним під запитом.

Розглянемо наступний приклад:

select e.e_name, e.e_id from empl e where not exists

(select * from proj_emp

where e_id = e. e_id) ;

Тут нам потрібно одержати список службовців, які ніколи не працювали над проектами.

```
mysql> select e.e_name, e.e_id from empl e where not exists
    -> (select * from proj_emp
    -> where e_id=e.e_id);
+---------+------+
| e_name  | e_id |
+---------+------+
| Konov   |  113 |
| Sidorov |  131 |
+---------+------+
2 rows in set (0.00 sec)
```

У під запиті ми шукаємо рядки таблиці proj_emp, 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 під запиту, 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 JOI, як

це було зроблено вище. Використання LEFT JOI, крім того, буде більше ефективним, і, таким чином, що відповідає запит буде виконаний швидше.

Ключові слова ALL, ANY й SOME використаються для порівняння з набором значень, повернутих під запитом. Припустимо, що Ivanov, що, якщо ви помнете, є найбільш завантаженим програмістом, поставить завдання довести, що ніхто не працює більше програмістів. Щоб установити цей факт, вона створить наступний запит:

```
select e.e_name
from empl  e, proj_emp  a where e.e_id = a.e_id and a.royalty_share>all
(select a.royalty_share
from proj_emp  a, empl  e
where e.e_id = a.e_idande.job='Programer');
```

Тут під запит повертає список процентних ставок, які були нараховані за виконання завдань програмістами в компанії. Потім запит шукає співробітників, які працювали над завданнями більше, ніж ці програмісти, використовуючи для цього a. royalty_share> ALL (процентні ставки програмістів). Для вас не буде сюрпризом, що цей запит не поверне ні одного рядка, доводячи той факт, що в даній компанії ніхто не працює так важко, як програмісти.

Питання та завдання до контролю знань студентів

1) Яку послідовність дій припускає процес створення такого багатокомпонентного об'єкта, як база даних? Поясните синтаксис необхідних структур.

2) Використовуючи базу даних Сесія, одержите:

– список студентів - претендентів на відрахування, уважаючи, що у відношенні S1 перебувають остаточні результати сесії, і тому відрахуванню підлягають всі студенти, які не здали або не здавали два й більше з покладених іспитів у сесію;

– список студентів, переведених на наступний курс;

– відрахувати студентів за результатами поточної сесії.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 30

з навчальної дисципліни «Бази даних»

Тема Відбір даних з декількох таблиць

Мета заняття Розглянути види об’єднання таблиць БД

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Об’єднання декількох таблиць

2 Множинні операції реляційної алгебри

3 Відкриті з'єднання

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Об’єднання декількох таблиць

Будь-який запит, що вибирає дані з більш ніж однієї таблиці, виконує яке-небудь об'єднання. Перелічимо основні характеристики об'єднань:

− операнди об'єднання (тобто ті дві таблиці, які подаються на вхід) звичайно називають першою таблицею й другою таблицею, але, маючи справу із зовнішніми об'єднаннями, де порядок проходження вхідних таблиць важливий, їх називають лівою таблицею й правою таблицею відповідно;

− таблиці завжди поєднуються порядково при виконанні всіляких умов, певних у вашому запиті;

− ті рядки, які не відповідають заданим умовам, можуть бути як включені в об'єднання, так і виключені з нього, залежно від типу цього об'єднання;

− значення зв'язаних стовпців звичайно перевіряють на рівність (=), але ви також можете порівнювати ці значення, застосовуючи інші оператори порівняння (<>, <, <=, >, >=);

− *об'єднанням по рівності* називається таке об'єднання, в умові якого застосовується оператор рівності (=), щоб групувати рядки, що мають рівні значення в певних стовпцях, називаних зв'язаними;

− об'єднання по рівності - це окремий випадок більше загального тэта-об'єднання, при якому значення зв'язаних стовпців порівнюють із застосуванням не тільки оператора рівності, але й, можливо, будь-якого іншого оператора порівняння;

− зв'язані (або єднальні, як їх ще називають) стовпці будь-якого об'єднання найчастіше виявляються зв'язаними ключовими стовпцями, але ви самі, будуючи своє об'єднання, можете зв'язати будь-які стовпці, якщо їхні типи даних сумісні (все це не має відносини до так називаного перехресного об'єднання, при якому не повинне бути ніяких виділених зв'язаних стовпців);

− щоб убезпечити себе від побудови безглуздих об'єднань вибирайте єднальні стовпці так, щоб вони були визначені на одному домені (умова об'єднання, що зустрічається часто, включає який-небудь зовнішній ключ однієї дочірньої таблиці й пов'язаний із цим зовнішнім ключем первинний ключ іншої материнської таблиці;

− якщо який-небудь ключ є складовим (тобто містить кілька стовпців), можна зробити єднальними всі стовпці цього ключа;

− зв'язані (єднальні) стовпці не обов'язково повинні мати те саме ім'я (за винятком випадку природного об'єднання).

Для того, щоб об'єднати інформацію з декількох вихідних таблиць в результуючій таблиці, необхідно виконати операцію з'єднання, при якій об'єднання інформації із двох таблиць відбувається за допомогою утворення

пар зв'язаних рядків, обраних з кожної таблиці. Таблицям можна привласнити імена-псевдоніми, що буває корисно для здійснення операції з'єднання таблиці із самої собою й у ряді інших ситуацій. Якщо в операторі SELECT зазначено більше одного імені таблиці, неявно мається на увазі, що над перерахованими таблицями здійснюється операція декартового добутку. Найпростіший запит SELECT такого роду без необов'язкових частин виглядає в такий спосіб:
SELECT * FROM R1, R2і відповідає декартовому добутку таблиць R1 й R2.

Вираз SELECT R1.A, R2.B FROM R1, R2;відповідає проекції декартового добутку двох таблиць на два стовпці А з таблиці R1 й В з таблиці R2. Розглянемо базу даних, у якій зберігається інформація про вироблені виплати фахівцям за пророблену роботу з певних етапів НДР. Нехай вона складається із трьох відносин Rl, R2 й R3. Будемо вважати, що вони представлені таблицями Rl, R2 й R3 відповідно.

R1= (ПІБ, Відділ); R2 = (Відділ, Етап); R3= (ПІБ, Етап, Нарахування).

R1

ПІБ	**Відділ**
Семенов Т.Т.	03
Просов СМ.	03
Мехова И.И.	03
Немцов Я.Ю.	04
Яров И.М.	04

R2

Відділ	**Етап**
03	Этап_1
03	Этап_2
03	Этап_3
04	Этап_3
04	Этап_4

R3

ПІБ	**Етап**	**Нарахування (руб)**
Семенов ТТ.	Этап_1	1000
Просов С.М.	Этап_1	2000
Мехова И.И.	Этап_1	500
Семенов Т.Т.	Этап_2	500
Просов С.М.	Этап_2	500
Мехова И.И.	Этап_2	1000
Просов С.М.	Этап_3	1000
Мехова И.И.	Этап_3	1000
Чемцов Я.Ю.	Этап_3	2000
Чемцов Я.Ю.	Этап_4	2000
Яров И.М.	Этап_4	3000

Для запитів буде використатися й розширена база даних Сесія,

представлена таблицями: **S1 = (ПІБ, Дисципліна, Оцінка)** - утримуючої відомості про результати сесії; **S2 = (ПІБ, Група)** - утримуючої відомості про склад груп; **S3 = (Група, Дисципліна)**– утримуючий перелік іспитів, що підлягають здачі.

S1

ПІБ	**Дисципліна**	**Оцінка**
Мур С.М.	Физика	4
Цуканов Т.Т.	Физика	5
Думская М.Т.	Физика	3
Дрозд Г.Р.	Физика	4
Мур С.М.	История	4
Цуканов Т.Т.	История	5
Думская М.Т.	История	3
Цуканов Т.Т.	Математика	5
Думская М.Т.	Математика	4
Дрозд Г.Р.	Математика	5
Петрова С.О.	Электротехника	5
Часов И.И.	Электротехника	4
Иванова Я.С.	Электротехника	5
Крисс Р.О.	Электротехника	3
Часов И.И.	Иностр_язык	5
Иванова Я.С.	Иностр_язык	4
Часов И.И.	Экономика	4
Иванова Я.С.	Экономика	4
Крисс Р.О.	Экономика	5
Фирсова Л.Р.	Экономика	3

S2

ПІБ	**Група**
Мур С.М.	02-КТ-21
Цуканов Т.Т.	02-КТ-21
Думская М.Т.	02-КТ-21
Дрозд Г.Р.	02-КТ-21
Петрова С.О.	О2-КТ-12
Часов И.И.	02-КТ-12
Иванова Я.С.	О2-КТ-12
Крисе Р.О.	02-КТ-12
ФирсоваЛ.Р.	02-КТ-12

S3

Група	**Дисципліна**
02-КТ-21	Физика
02-КТ-21	История
02-КТ-21	Математика
02-КТ-12	Экономика
02-КТ-12	Электротехника
02-КТ-12	Иностр_язык

*Запит 1*БД НДР. Вивести список співробітників відділу 03, які брали участь у виконанні Етапу_3. Запит буде виглядати в такий спосіб:

```
SELECT R3.ПІБ, R3.Етап
FROM R1, R3
WHERE R1.Відділ = '03' AND
R1.ПІБ = R3.ПІБ AND
R3.Етап = 'Етап_3'
```

Результат запиту:

ПІБ	**Етап**
Просов С.М.	Этап_3
Мехова И.И.	Этап_3

Запит 2 Вивести групи, у яких по одній дисципліні на іспитах отримано більше однієї п'ятірки. Запит буде виглядати в такий спосіб:

```
SELECT S2.Група
FROM S1, S2
WHERE S1.ПІБ = S2.ПІБ AND
S1.Оцінка = 5
GROUP BY S2.Група , S1.Дисципліна
HAVING count(*)>1
```

Результатом виконання розділу HAVING є згрупована таблиця, що містить тільки ті групи рядків, для яких результат обчислення умови пошуку є TRUE. Результат запиту:

Група
02-КТ-21
02-КТ-12

Запит 3 Приклад запиту, де буде використаний предикат NOT EXISTS. Його можливості зручно проілюструвати в тексті багатотабличного запиту. Вивести список тих студентів, хто повинен був здавати екзамен з історії, але поки ще не здавав. Запит буде виглядати в такий спосіб:

```
SELECT ПІБ FROM S2 a, S3
WHERE S2.Група=S3.Група AND
Дисципліна = 'Історія' AND
NOT EXISTS (SELECT ПІБ FROM SI
                    WHERE ПІБ = a.ПІБ AND
                    Дисципліна = 'Історія')
```

Результат запиту:

ПІБ
Дрозд Г.Р.

Нагадаємо, що предикат EXIST істинний, коли підзапит не порожній, тобто містить хоча б один кортеж, у противному випадку предикат EXIST хибний. Предикат NOT EXIST –істинний тільки тоді, коли підзапит порожній.

Обробка такого запиту полягає в тому, що для кожного студента, що навчається в групі, студентам якої необхідно здавати екзамен з історії, перевіряється істинність предиката NOT EXIST. Істинність його встановлюється по факті присутності у внутрішньому запиті значень. Якщо підзапит порожній, то даний студент ще не здавав екзамен з історії, предикат NOT EXIST має значення TRUE, і ПІБ студента міститься в результуючу таблицю висновку.

2 Множинні операції реляційної алгебри

Мова SQL дозволяє використати такі бінарні реляційні операції над відношеннями (таблицями), як *об'єднання* (UNION), *перетинання* (INTERSECT), *різниця* (EXCEPT). На таблиці, що беруть участь у таких множинних операціях, накладаються певні обмеження, що забезпечують їхня сумісність по з'єднанню:

− таблиці повинні мати однакові схеми;

− значення у відповідних стовпцях таблиць повинні належати тому самому домену.

Перше обмеження є обов'язковим, і його контроль забезпечується засобами мови. Стежити за тим, щоб вихідні таблиці задовольняли другому обмеженню, і щоб не відбувалося з'єднання "книг" й "черевик", повинен сам користувач. Оператори мають наступну структуру:

Оператор [ALL] [CORRESPONDING [BY {colomn1 [,...]}]].

Фраза CORRESPONDING BY задає операцію для зазначених у ній стовпців. Якщо ж в операторі слово BY опущене, то операція задається для загальних для двох таблиць стовпців. Присутність в операторі ключового слова ALL визначає ситуацію, коли дублюючі рядки з результуючої таблиці не виключаються. Запит на виконання, скажемо, оператора об'єднання повинен будуватися за наступною схемою:

(SELECT - запит)

UNION

(SELECT - запит)

UNION

(SELECT - запит)

Запит на об'єднання може поєднувати будь-яке число вихідних запитів. Всі запити, що беруть участь у зазначених операціях, не повинні містити виразів, тобто полів, що обчислюють. Жоден з вихідних запитів в операції UNION не повинен містити пропозиції впорядкування результату ORDER BY, однак результат об'єднання може бути впорядкований, для цього пропозиція ORDER BY із вказівкою списку стовпців упорядкування записується після тексту останнього вихідного SELECT-запиту. Порядок запису всіх зазначених операторів однаковий. Приведемо простий приклад, що розглядався при вивченні операцій реляційної алгебри, де використалася ситуація із двома вихідними відносинами, що представляють склад виробів:

***r* —виріб 1**

Код_ деталі	Назва	Вага
01	A	1
03	B	2
04	C	3

***s* — виріб2**

Код_ деталі	Назва	Вага
02	D	2
03	B	2
04	C	3

Запит 4 Які типи деталей входять до складу обох виробів? Для досягнення цієї мети необхідно виконати операцію об'єднання, де запит буде виглядати в такий спосіб:

(SELECT *FROM r)

UNION CORRESPONDING

(SELECT *FROM s)

Оскільки обидві таблиці мають всі загальні стовпці, то результуюче відношення, що містить всі деталі, що входять до складу обох виробів, буде виглядати так:

Код_ деталі	Назва	Вага
01	A	1
02	D	2
03	B	2
04	C	3

За замовчуванням при виконанні запиту на об'єднання дублікати кортежів

завжди виключаються. Тому, якщо найдуться деталі, які входять до складу обох виробів, то вони однаково в результуючий список потраплять тільки один раз. В SQL той самий запит можна виконати різними способами й цим запитом не є виключенням. Той же результат можна одержати простою зміною фрази WHERE першої частини вихідного запиту, з'єднавши локальні умови логічною операцією АБО й виключивши дублікати кортежів.

3 Відкриті з'єднання

При виконанні запитів часто необхідно проводити з'єднання даних різних таблиць. Одна з можливостей реалізації з'єднань полягає у використанні умов, що задають у частині WHERE оператора SELECT. У цьому випадку із двох таблиць комбінуються рядки шляхом утворення по заданих умовах зчеплених пар рядків, по одній з кожної таблиці, для яких ці умови були визначені й щирі. Як умова зчеплення рядків можна використати, наприклад, факт рівності значень стовпців, що зіставляють. Якщо рядок однієї з таблиць не знаходить собі відповідності в іншій, то вона не попадає в результуючий набір даних. Таке з'єднання називається внутрішнім. Стандарт SQL2 розширив поняття умовного з'єднання. У ньому передбачене утворення відкритих з'єднань, при яких у результуючу таблицю містяться також рядки, що не задовольняють умові з'єднання. Необхідність виконання подібних з'єднань часто зустрічається в дійсності. Наприклад, буває необхідно в результуючу таблицю помістити всі рядки однієї з вихідних таблиць, а якщо для якогось її рядка не перебуває відповідного рядка в другій таблиці, то на місце цього другого рядка містяться невизначені значення. Існують три типи відкритих з'єднань:

− LEFT JOIN - ліве відкрите з'єднання; у результат входять всі рядки лівого операнду-таблиці, а частини результуючих рядків, для яких не було відповідних значень у правої таблице-операнде, доповнюються значеннями NULL;

− RIGHT JOIN - праве відкрите з'єднання; у результуючу таблицю включаються всі рядки правої таблиці-операнду, а відсутні частини з лівої таблиці-операнду заповнюються NULL-значеннями;

− FULL JOIN - повне відкрите з'єднання; виконуються одночасно й ліві й праве відкриті з'єднання: у результуючу таблицю включаються всі рядки й лівої й правої таблиць. У тому випадку, якщо який-небудь рядок з якоїсь таблиці не може бути зчеплена з рядком з іншої таблиці, те відсутні значення визначаються як NULL-значення.

Для того, щоб можна було відрізняти відкриті з'єднання від внутрішнього з'єднання, останнє позначається словом INNER.

Запит 5 БД НДР. Вивести таблицю, де будуть представлені всі фахівці відділів і для кожного фахівця будуть перераховані всі етапи, у виконанні яких він міг брати участь. По кожному етапі для кожного фахівця необхідно проставити величину виплачених за роботу сум. Якщо ж з якоїсь причини нарахування не були зроблені, то їх визначити як NULL-значення. Операція зовнішнього об'єднання може використатися для формування джерел у пропозиції FROM, тому реалізація такого запиту можлива за допомогою наступної SQL-конструкції:

```
SELECT R3.ФИО, R3.3Tan, R3.Нарахування
FROM (R1 INNER JOI R2) LEFT JOI R3 USING(ФИО, Етап)
```

У даному запиті у фразі FROM зроблене спочатку внутрішнє з'єднання таблиць R1 й R2, а потім отримана таблиця використається у відкритому лівому з'єднанні з таблицею R3 по стовпцях ФИО й Етап. Результат виконання такого запиту:

ПІБ	**Етап**	**Нарахування (руб)**
Семенов Т.Т.	Этап_1	1000
Просов С.М.	Этап_1	2000
Мехова И.И.	Этап_1	500
Семенов Т.Т.	Этап_2	500
Просов СМ.	Этап_2	500
Мехова И.И.	Этап_2	1000
Семенов Т.Т.	Этап_3	NULL
Просов С.М.	Этап_3	1000
Мехова И.И.	Этап_3	1000
Чемцов Я.Ю.	Этап_3	2000
Яров И.М.	Этап_3	NULL
Чемцов Я.Ю.	Этап_4	2000
Яров И.М.	Этап_4	3000

Питання та завдання до контролю знань студентів

1) Приведіть приклад необхідності організації вкладених запитів. Якими засобами для їхньої реалізації розташовує оператор SELECT?

2) У багатьох випадках для одержання відповіді на запит необхідно об'єднати інформацію з декількох вихідних таблиць. Як це може бути реалізоване в мові SQL?

3) Якими структурами представлені в мові SQL оператори маніпулювання даними, що дозволяють модифікувати існуючу базу даних?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 31

з навчальної дисципліни «Бази даних»

Тема Типи таблиць MySQL.

Мета заняття Розглянути типи таблиці ISAM, MyISAM та InnoDB

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Типи таблиць MySQL

1.1 Таблиці ISAM

1.2 Таблиці MyISAM

1.3 Таблиці InnoDB

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Типи таблиць MySQL

Проектування системи керування базою даних, подібно багатьом іншим завданням проектування, має на увазі безліч компромісних рішень. Наприклад, багато завдань бази даних повинні забезпечуватися рішеннями, що припускають безпека транзакцій, але такі рішення вимагають більше часу для

їхнього виконання, більше дискового простору й пам'яті. Творці MySQL делегували вибір деяких з компромісних рішень вам, розроблювачеві бази даних, запропонувавши на вибір кілька типів таблиць. Залежно від додатка ви можете вибрати один з типів таблиць, що забезпечують безпеку транзакцій, або ж за допомогою іншого типу забезпечити більше високу продуктивність, не гарантуючи безпеку транзакцій. Так чи інакше, необхідно знати, які компроміси ви при цьому допускаєте.

Ви вже могли звернути увагу, що типи таблиць ми називали механізмами зберігання. Це відбиває той факт, що деякі з типів таблиць використають більші самостійні фрагменти програмного коду зі своїми системами кешування, індексації, блокування й доступу до диска. Це відбиває також суть призначення бази

Приклади, які ми розглядали, вивчаючи запити SQL, виконувались автономно, але в багатьох додатках це зовсім не відповідає реальному положенню справ. Представте на хвилину, що ви маєте справу з базою даних, що містить інформацію про банківські рахунки. Якщо необхідно перевести 1000 доларів з одного рахунку на іншій, вам буде потрібно як мінімум два запити SQL: один, щоб зняти 1000 доларів з одного рахунку, і ще один, щоб додати 1000 доларів до другого рахунку. Існує небезпека (наприклад, у випадку відключення електроживлення), що один запит завершить свою роботу, а другий - немає. Було б дуже бажано, щоб у подібних випадках завершитися успішно могли або обоє з пари запитів, або жоден з них, щоб дані в базі даних залишилися погодженими.

Щоб забезпечити безпека транзакцій, таблиці повинні передбачати можливість вказівки того, що деякий набір запитів повинен трактуватися як одна неподільна одиниця - транзакція. Транзакція повинна або завершитися повністю, або база даних повинна виконати відкіт, тобто повернутися до того ж стану, що мало місце до початку виконання транзакції.

До типів таблиць, доступних в MySQL, ставляться наступні:

– ISAM
– MyISAM

– InnoDB
– BerkeleyDB (BDB)
– MERGE
– HEAP

Ми розглянемо кожний із цих типів по черзі, але максимум уваги приділимо найбільше часто використовуваним типам: MyISAM й InnoDB. Безпеку транзакцій забезпечують таблиці InnoDB й BerkeleyDB. Інші (ISAM, MyISAM, MERGE н REAP) безпеку транзакцій не забезпечують.

1.1 Таблиці ISAM

Таблиці ISAM включені в MySQL винятково для підтримки сумісності. Їхні функціональні можливості повністю підтримуються таблицями MyISAM, тому таблиці ISAM тут докладно розглядатися не будуть. Їхню підтримку планується припинити в MySQL 5.0;

За допомогою наступного коду можна створити таблицю ISAM, якщо з якої-небудь причини вам знадобиться:

```
create table asset (asse tiint not null,
                    description varchar(255) ) type=ISAM;
```

Таблиці ISAM пропонують швидкий, але не захищений з погляду транзакцій, механізм зберігання даних. Практично все, що можна сказати про таблиці ISAM, буде вірно й для таблиць MyISAM, але стосовні до більше старого стандарту таблиці ISAM мають ряд обмежень.

Поліпшення, запропоновані в таблицях MyISAM, включають наступне.

Мобільність таблиць. Таблиці, збережені на диску або іншому носії, можна завантажити в іншу систему, що підтримує MySQL, незалежно від платформи. Для таблиць ISAM це не так.

Підтримка дуже великих таблиць. Розміри таблиці ISAM мають тверде обмеження - 4 Гбайт. MyISAM дозволяє створювати таблиці настільки більші, наскільки більшими їм дозволяє бути операційна система, у якій виконується MySQL. Це може виявитися важливим невеликому числу користувачів, але це означає, що до вибору операційної (і: файлової) системи варто віднестися

досить уважно. Багато файлові системи мають обмеження 2 Гбайт для розміру файлу. (Зверніть увагу на те, що це обмеження можна насправді обійти, використовуючи таблиці MERGE.)

Більше ефективне використання дискового простору. Скорочений об'єм порожнього простору й зменшена фрагментація.

Менші обмеження на ключі. Таблиці ISAM допускають використання 16 ключів на таблицю й максимальну довжину ключа за замовчуванням 256 байт. Таблиці MyISAM допускають 64 ключа на таблицю й максимальну довжину ключа за замовчуванням 1024 байт.

Таблиці ISAM повинні сприйматися як небажані. Вони усе ще існують, але їх не слід використати для нових проектів. Якщо у вас є які-небудь таблиці ISAM, настійно рекомендується розглянути можливість їхнього перетворення у формат MyISAM. Це зажадає дуже незначних зусиль, але забезпечить досить істотні переваги.

1.2 Таблиці MyISAM

Багато хто працюють із MySQL протягом багатьох лет, не підозрюючи, що ця система допускає використання різних типів таблиць. Ці люди використають таблиці MyISAM, тому що вони використалися за замовчуванням у всіх останніх версіях.

Таблиці MyISAM забезпечують дуже швидкий, але не захищений з погляду транзакцій, механізм зберігання даних. Вони забезпечують високу продуктивність у більшості ситуацій навіть при наявності помилок у структурі бази даних, а в руках кваліфікованого адміністратора вони можуть ефективно обслуговувати дуже великі й інтенсивно використовувані бази даних. За допомогою наступного коду можна задати таблицю MyISAM:

```
create table article (articlel int not null auto_increment primary key,
                      title varchar(255), bodytext );
```

Останній рядок міг би виглядати й так:)

```
type=MyISAM;
```

але результат від цього не змінився б.

Таблиці MyISAM можуть бути одного з типів: динамічні, статичні або стислі. Таблиця автоматично стає динамічної або статичною залежно від визначення її стовпців. Стислі таблиці повинні створюватися спеціально за допомогою засобу myisampack.

Таблиці з рядками фіксованої довжини будуть статичними, а таблиці з рядками змінної довжини - динамічними але як з'ясувати, рядка якої довжини має таблиця - фіксованої або змінної?

Тип char і всі числові типи мають фіксовану довжину. Розмір стовпців varchar, text й blob може варіювати залежно від їхнього вмісту. Таблиця зі стовпцями тільки числових типів і типу char буде створена як статична, а таблиця, що містить хоча б один стовпець типу varchar, text або blob, буде динамічною.

Статична таблиця має цілий ряд переваг. Вона забезпечує більше швидкий пошук у порівнянні з динамічною або стислою таблицею. Для бази даних дуже легко витягти певний запис на основі індексу, коли кожен запис перебуває на певній відстані від початку файлу. Дані простіше кешувати. Менш імовірно виникнення серйозних ушкоджень у випадку відмови системи - можливості відновлення даних, як правило, легко забезпечують відновлення всіх рядків, окрім одного ушкодженого.

Недоліком статичних таблиць є те, що розміщення реальних даних у стовпцях фіксованого розміру не сприяє економії дискового простору. Це може бути не занадто великою ціною для даних, що варіюються у невеликих межах (наприклад, для імен користувачів), але може виявитися неприйнятним для даних, які сильно варіюють у розмірах. Якщо ви приймете оцінку зверху в 80 символів для імен службовців, то повинні бути готові витратити зайві 75 байт щораз , коли буде потрібно зберегти ім'я Сміт.

Динамічні таблиці в рамках MySQL вимагають більше складних методів керування. У цьому випадку й кешування, і пошук, і відновлення записів для механізму зберігання виявляються не такими простими завданнями. Причиною почасти є те, що дані варіюють у розмірах, але також і те, що вони можуть виявитися фрагментованими. Якщо рядок змінюється й стає більше, частина

даних залишиться на вихідному місці, а частина може зберегтися у вигляді нового фрагмента де-небудь в іншому місці файлу. Це значить, що сегмент файлу, поміщений у кеш операційною системою, не обов'язково буде містити всі частини відповідного рядка. Можуть виникнути труднощі й при відновленні ушкоджень, оскільки у випадку втрати фрагментів або посилань буде зовсім не очевидно, яким рядкам повинні належати розпізнані частини даних.

Щоб відновити або дефрагментувати таблицю MyISAM, можна використати засіб myisamchk або команду MySQL REPAIR TABLE. Щоб дефрагментувати, але не відновлювати таблицю, можна використати команду MySQL OPTIMIZE TABLE.

Стиск таблиць MyISAM

Хоча таблиці стають статичними або динамічними без вашої спеціальної вказівки (але відповідно до розроблювальними вас структурними рішеннями), автоматично таблиці не стискуються. Щоб стиснути таблицю, необхідно використати програму myisampack, що запускається з командного рядка. (Існує версія цієї програми для таблиць ISAM, вона називається pack_isam, і по необхідності ви можете скористатися нею.)

Стиск здається корисним рішенням, але насправді воно виявляється прийнятним не завжди, оскільки стислі таблиці стають доступними тільки для читання. Якщо буде потрібно змінити, модифікувати або додати дані, прийдеться відновити таблицю в колишньому виді, ввести зміни, а потім стиснути неї знову.

Стиск, виконуваний програмою myisampack, містить у собі як безпосередньо стиск (кодування по методуХаффмана), так і множину оптимізуючих дій, метою яких є зменшення розміру стовпців (наприклад, перетворення типів у менш ємні й стовпців - у перечислимі). Оскількикожен запис стискується окремо, додаткові зусилля, які потрібні для декомпресії запису, відносно невеликі. На повільних пристроях це може навіть компенсуватися зменшенням обсягу даних, які потрібно зчитувати з диска.

Повнотекстовий пошук у таблицях MyISAM

Однією з можливостей, що у цей час виявляється доступною тільки при роботі з таблицями MyISAM, є повнотекстовий пошук й індексація. Звичайні індекси дуже добре підходять для пошуку рядків, коли потрібно зрівняти задане значення зі значеннями в таблиці, але нерідко потрібно знайти слова або рядки в текстових блоках. Саме в таких випадках можливість повнотекстового пошуку виявляється дуже до речі.

Наступний запит SQL створить таблицю MyISAM з повнотекстовим індексом:

```
create table article (articlel int not null auto_incrernent primary key,
                      title varchar (255),
                      body text,
                      fulltext (title,body) );
```

А цей запит поверне всі рядки, що містять слово ' merger':

```
select title from article
where match(title, body) against ('merger') ;
```

Підтримуються й більше складні можливості пошуку. Наступний запит витягне рядки, що містять кожне зі слів 'merge', 'acquisition', 'acquire' або 'takeover':

```
select title from article
where match (title, body) against ('merge acquisition acquire take over');
```

Зверніть увагу на те, що запиту буде відповідати будь-який запис, що містить як мінімум одне із зазначених слів. Ми не шукаємо рядок або запис, що містить всі наведені слова. Для здійснення цього виду пошуку ми можемо використати модифікатор IN BOOLEAN MODE, що обговорюється нижче.

Ми повинні були шукати 'acquire' й 'acquisitions' окремо, тому що MySQL у цей час не підтримує морфологічний пошук. Морфологічний пошук - це технологія пошуку, застосовувана в багатьох системах повнотекстового пошуку, що розпізнають слова з однією основою. Так, слово'acquire' є основою множини інших слів, наприклад, 'acquires', 'acquired' й 'acquisition'.

Кожній знайденій відповідності приписується коефіцієнт релевантності, і результати автоматично сортуються в цьому порядку. Коефіцієнти

релевантності для записів можна побачити. Наступний запит виведе несортований список коефіцієнтів релевантності для всіх рядків. Записи з нульовим значенням коефіцієнта не мають ніякої подібності із запитаними даними й тому витягнуті не будуть.

```
select title, match (title, body)
against ('merge acquisition acquire take over')
as relevance
from article;
```

Зверніть увагу на те, що тут relevance - це не ключове слово, а псевдонім для match (title, body) against ('mergeacquisitionacquiretakeover'). Ми використали його для того, щоб результат був більше акуратним.

Більше корисним виявляється наступний запит, що повертає заголовки статей і коефіцієнти релевантності для відповідних документів. Через те, що умова MATCH зазначена у виразі WHERE, результати будуть відсортовані, а невідповідні рядки - зігноровані.

```
select title,  match   (title, body)
against   ('merge acquisition acquire take over')
as relevance
from article
where match   (title, body)   against   ('merge acquisition acquire take over');
```

Існують слова, які ви не зможете знайти. Для поліпшення продуктивності деякі слова виключаються з індексів або ігноруються при пошуку.

Короткі слова не індексуються. За замовчуванням слова, що складаються менш чим із чотирьох символів, ігноруються. Іноді це породжує проблеми (найбільш відомий приклад - Slashdot.org), оскільки трибуквені скорочення часто є важливою складовою технічних текстів. Можна обійти це обмеження, змінивши значення змінної ft_min_word_len, але після цього вам доведеться перешикувати свої індекси.

У повнотекстових індексах використаються стоп-слова, тобто слова, що не мають семантичного значення. Загалом кажучи, ці слова важливі з погляду конструкції пропозиції, але навряд чи виявляться важливими частинами вмісту:

Наприклад, артиклі, союзи й тому подібні слова, звичайно, необхідні, але звичайно вони не занадто корисні для пошуку. Забезпечуючи стандартні списки,

MySQL дає вам можливість указати й свій власний список *стоп-слів* для будь-якої мови, на якому виконується індексування.

Щодо повнотекстової індексації необхідно ще сказати наступне. Це - складне завдання, і для неї існують цілком певні межі продуктивності. Коли таблиці MySQL стають більшими (скажемо, містять більше 1000000 рядків), швидкість повнотекстового пошуку істотно знижується. Для малих додатків це не породжує серйозних проблем, але ви не повинні забувати про це, якщо вам доводиться мати справа з більшими додатками.

Слова, які часто зустрічаються у ваших даних, при пошуку не використаються. Якщо таблиця містить статті інформаційного бюлетеня компанії Acme PTY Ltd., те цілком імовірно, що багато хто з них будуть містити слово 'Acme'. Пошук по цьому слову повинен генерувати дуже велика результуюча безліч, у чому звичайно не занадто багато користі. Якщо 50% або більше значна частина ваших записів містить деяке слово, то при обчисленні коефіцієнта релевантності це слово буде розцінюватися як що не має значення.

Повнотекстовий пошук у режимі BOOLEAN

Можна забезпечити більший контроль над результатами пошуку за допомогою модифікатора IN BOOLEAN MODE.

Наступний запит виявить тільки записи, що містять слово 'linux' й строку"0penSource", але не слово 'desktop'.Слова 'Java' й 'Oracle' необов'язкові, але при оцінці релевантность присутність у записі слова 'Java ' поліпшить оцінку, тоді як присутність 'Oracle' - погіршить. Порядок слів у рядку пошуку або запису не має значення.

```
select title
from article
wher match (title, body)
against ('+linux +"OpenSource"  -desktopJava-Oracle'
IN  BOOLEAN  MODE);
```

Повний набір операторів розглянутого режиму показаний у таблиці31.1.

Таблиця 31.1 – Набір операторів

Оператор	Значення
+	Слово є обов'язковим
-	Слово не повинне бути присутнім
<	Слово є менш важливим
>	Слово є більше важливим
( )	Групують слова в підвираз
~	Слово може бути присутнім, але впливає на оцінку релевантності
*	Суфікс, що має групове значення. Наприклад, merge не буде відповідати merger, але merge* відповідає й merge, і merger. Може використатися тільки наприкінці слова

Для виконання пошуку в режимі BOOLEAN не обов'язково створювати повнотекстові індекси. Можна шукати й у неіндексованих таблицях, але відповідний процес буде протікати дуже повільно.

Ще одна невелика різниця між повнотекстовим пошуком і пошуком у режимі BOOLEAN полягає в тім, що при пошуку в режимі BOOLEAN слова, щочастозустрічаються слова, не ігноруються. Тут правило п'ятдесяти відсотків не застосовно. При пошуку статей інформаційного бюлетеня Acme PTY Ltd. перший з наступних двох запитів швидше за все повернув би майже всі рядки, тоді як другий - нічого.

```
select title from article
where match (title,body) against ('Acme' IN BOOLEAN MODE);
select title from article
wher ematch (title,body) against ('Acme');
```

1.3 Таблиці InnoDB

Наступним типом таблиць, що ми будемо обговорювати, є тип InnoDB. Даний тип являє собою механізм зберігання, що забезпечує швидку роботу й безпеку транзакцій. Таблиці InnoDB пропонують наступні можливості.

Транзакції. (Докладніше про їх мова йтиме пізніше.)

Блокування на рівні рядків. Це означає, що при виконанні запиту недоступної для інших користувачів буде тільки рядок, використовувана в

даному запиті. Більшість інших механізмів зберігання (за винятком BDB) пропонує блокування на рівні таблиць - поки один процес обновляє таблицю, таблиця не доступна іншим процесам.

Підтримка зовнішніх ключів. Приклади, у яких використаються зовнішні ключі, не будуть працювати з іншими типами таблиць.

Погоджене неблокуюче читання в операторах SELECT. (Ця ідея запозичена з Oracle.)

Механізм InnoDB має свої власні опції настроювання, окремий каталог і свої особливості зберігання даних. У той час як механізм MyISAM для зберігання кожної таблиці використає окремий файл, механізм InnoDB зберігає всі таблиці й індекси в спеціальному просторі таблиць, і це означає, що вміст однієї таблиці може розмішатися в декількох файлах. Це дає можливість механізму InnoDB створювати дуже більші таблиці, які не зобов'язані підкорятися яким-небудь обмеженням на розміри файлів, що накладають операційною системою. Однак, варто враховувати те, що для зберігання того самого обсягу корисної інформації таблиці InnoDB використають значно більше дискового простору, чим таблиці MyISAM.

Незважаючи на те що механізм InnoDB розроблявся окремо від більшості інших складових MySQL, він також доступний у рамках подвійної ліцензійної схеми.

Механізм InnoDB використають багато більші й дуже відомих Web-вузлів, такі, наприклад, як, Slashdot (www.slashdot.org), Google (www.google. com) і Yahoo!Finance (http: //finance, yahoo.com). Цей механізм особливо добре підходить для роботи з більшими обсягами даних в умовах, що потребують швидкого доступу й безпеки транзакцій.

InnoDB є однієї з найшвидших систем, що забезпечують безпеку транзакцій, але забезпечення цієї безпеки вимагає жертв. Для більшості реальних сценаріїв таблиці MyISAM будуть швидше, але різниця, загалом , не буде занадто вражаючою.

Система InnoDB створена окремою компанією, InnoBaseOy, що має власний Web-вузол. Саму свіжу інформацію про систему InnoDB ви знайдете за адресою www. innodb. com.

Питання та завдання до контролю знань студентів

1) Нам будуть потрібні транзакції й таблиці, які можна легко перенестиз однієї системи в іншу. У цьому випадку варто використати:
   а) ISAM;
   б) MyISAM;
   в) InnoDB;
   г) BOB.
2) Необхідно створити тимчасову таблицю для швидкого пошуку. У цьому випадкуварто використати:
   а) ISAM;
   б) MyISAM;
   в) MERGE;
   г) HEAR
3) Нам будуть потрібні дуже більші таблиці. У цьому випадку не треба використати:
   а) ISAM;
   б) MERGE;
   в) InnoDB;
   г) жоден з перерахованих вище варіантів.
4) Нам буде потрібно повнотекстовий пошук. У цьому випадку треба використати:
   а) MyISAM;
   б) InnoDB;
   в) BOB;
   г) HEAP.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 32

з навчальної дисципліни «Бази даних»

Тема Визначення транзакцій. Управління транзакціями в середовищі MySQL.

Мета заняття Ввести поняття транзакції, розглянути організацію транзакцій в MySQL

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Поняття транзакцій

2 Використання транзакцій в MySQL

3 Модель транзакцій InnoDB

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Поняття транзакцій

У контексті системи керування базами даних *транзакція* - це послідовність зв'язаних інструкцій, які повинні розглядатися як одна неподільна одиниця. Це значить, що або всі дії, запропоновані транзакцією, повинні бути виконані, або всі ці дії повинні залишитися невиконаними. Для

позначення такої вимоги використається поняття атомарність. Транзакція є атомарною, оскільки не може бути розбита на частині, - вона або цілком виконується, або цілком відкидається.

Це доводиться враховувати при розгляді питань паралельного доступу користувачів, процесів або потоків, а також питань відновлення даних. Сучасні комп'ютери часто бувають доступні одночасно декільком користувачам, звичайно допускають одночасне виконання не однієї, а відразу декількох програм і підтримують програмне забезпечення типу Web-сервера, що породжує безліч процесів або потоків, виконуваних одночасно. Кожному із цих користувачів, програм, процесів або потоків може знадобитися доступ до сервера бази даних.

Численні одночасно працюючі потоки не повинні заважати один одному. Якщо виникає помилка, при відновленні база даних повинна враховувати й транзакції. Це означає або повернення бази даних до стану, у якому вона перебувало до виникнення помилки, або завершення транзакції. Взагалі, краще втратити всю транзакцію й відповідні зміни, чим відновлювати стан, що частково змінився в результаті незавершеної послідовності модифікацій, оскільки -у такому випадку стан бази даних може виявитися неузгодженим.

Цю ідею можна виразити й більш формально, але найпростіше пояснити її на прикладі. Розглянемо просту (але потенційно дуже важливу) базу даних, у якій зберігаються записи про банківські рахунки. Кожен рахунок має, як мінімум, унікальний ідентифікатор і баланс (значення, що відповідає залишку на рахунку).

Відповідну таблицю можна створити так:

```
create table account
(number int not null auto_increment primary key,
      balance float) type * InnoDB;
```

Ви, напевно, догадаєтеся самі, що за допомогою наступних запитів ми створюємо нові рахунки:

```
insert into account (balance) values (0.0);
insert into account (balance) values (1000.0);
```

insert into account (balance) values (2000.0);

Можливостей для виникнення помилок тут не багато, і результатом повинне бути наступне:

number	balance
1	0
2	1000
3	2000

Ми не робимо ніяких застережень, що стосуються номерів рахунків, генеруємих командою auto_increment стовпця, тому не має ніякого значення порядок, у якому ці запити будуть виконані. Це не буде мати значення, наприклад, і в тому випадку, коли відповідні запити виконуються одночасно трьома різними клієнтами, і немає можливості вгадати, який із цих запитів буде оброблений першим.

Такий підхід може породжувати проблеми у випадку більше складних запитів або наборів запитів. Розглянемо наступну пару операторів, що ставлять своєю метою депонувати 500 доларів на рахунок 2:

```
# спочатку перевіримо баланс
select balance from account where number = 2;
# запит поверне значення J1000
# тепер збережемо нове значення балансу
update account set balance "* 1500 where number = 2;
```

Ці запити зв'язані, тому повинні виконуватися разом. Якщо інші клієнти можуть змінити залишок на цьому рахунку між нашою перевіркою балансу й наступною його модифікацією, ми можемо одержувати не той результат, якого очікуємо.

Якщо два клієнти виконують пари подібних запитів одночасно, заключний результат буде залежати від часу їхнього виконання. Якщо ми намагаємося депонувати 500 доларів за допомогою наших запитів, а інший клієнт намагається депонувати 100 доларів за допомогою наступної пари запитів, кінцевим результатом може бути баланс в 1100 або 1500 доларів - ні то ні інше правильним результатом не є:

```
# спочатку перевіримо баланс
select balance from account where number=2;
# запит поверне значення $1000
# тепер збережемо нове значення балансу
update account set balance - 1100 where number = 2;
```

Ясно, що таке положення справ зовсім небажано, але в цьому випадку проблема вирішується досить легко. Перетворивши обновлювані значення з абсолютних у відносні, ми одержимо окремі неподільні одиниці. Наступний запит буде виконаний коректно, незалежно від порядку, у якому обробляються одночасно представлені запити:

```
update account set balance o= balance + 500 where number = 2;
```

Окремий оператор update в MySQL завжди буде атомарним. Він не може бути перерваний іншим запитом або виконаний наполовину. Він або успішно завершиться, або генерує помилку.

Більше складні сценарії можуть породжувати й більше складні проблеми. Розглянемо наступну пару запитів, завданням яких є переміщення 1000 доларів з рахунку 2 на рахунок 1:

```
update account set balance " balance -  1000 where number = 2;
update account set balance = balance + 1000 where number = 1;
```

Тут обоє відновлення є відносними, але з погляду правильності результату дуже важливо, щоб ці два запити виконувалися разом. Загальна кількість грошей у системі після виконання запитів повинне залишитися тим же. Гроші переміщаються з одного рахунку на інший, а не створюються або знищуються. Якщо між виконанням першого й другого запитів відбудеться збій електроживлення, наші дані втратять погодженість.

У простих випадках рішенням може бути розміщення запитів в одному операторі SQL. У цьому випадку можна було б написати наступне:

```
updateaccountassource, accountasdest
set source.balance = source.balance - 1000,
dest.balance = dest.balance + 1000
where source.number = 2 and dest.number =1;
```

За допомогою двох псевдонімів (source й dest) для таблиці рахунків ми змогли створити одне атомарне відновлення, що або буде виконано, або цілком відкинуте. Тепер нам не потрібно турбуватися про можливості зупинки сервера між різними частинами наших операцій.

У цьому випадку проблемою є тільки зручність сприйняття. Об'єднаний запит трудніше читати й налагодити, чим наш перший варіант.

Але в багатьох випадках об'єднати всі зв'язані запити в один буває просто неможливо. У таких випадках і тоді, коли ви хочете мати код, більше зручний для читання, рішенням повинне бути використання синтаксису транзакцій MySQL. Позначивши набір операторів як транзакцію, ви інформуєте базу даних про те, що ці оператори є зв'язаними неподільною множиною, з якою варто обходитися так само, як з атомарною одиницею: або всі оператори повинні бути виконані, або жоден з них не повинен мати ніякого ефекту. Щоб виконати зазначені два запити як одну транзакцію, використається наступна послідовність операторів SQL:

```
start transaction;
update account set balance = balance - 1000 where number = 2;
update account set balance = balance + 1000 where number = l;
commit;
```

Важливою властивістю транзакцій є те, що вони залишаються невидимими для інших сеансів доти , поки не будуть завершені й зафіксовані. Ніякий інший потік не зможе прочитати неузгоджені дані таблиць, поки вони перебувають у процесі відновлення.

Інша перевага транзакцій полягає в тім, що частково виконані транзакції допускають відкіт. При спробі виконати відкіт транзакції до її фіксації всі зміни, виконані запитами, що формують транзакцію, будуть скасовані.

У нашому прикладі перекладу коштів можна використати додатковий оператор SELECT для перевірки того, що з вихідного рахунку не буде знято більше грошей, чим він містить, а також використати ключове слово ROLLBACK, щоб якщо буде потреба скасувати транзакцію. Відповідна послідовність операторів може виглядати так:

```
start transaction;
update account set balance = balance - 1000 where number = 2;
update account set balance = balance + 1000 where number = 1;
select balance from account where number = 2;
#       цей запит повідомляє про негативний баланс для рахунку №2!
#       краще виконати відкат
rollback;
```

Команда ROLLBACK скасовує транзакцію й всі зміни, які при цьому могли бути зроблені. Транзакція, що не була зафіксована, а скасована, не вносить ніяких змін у дані. А оскільки її часткові результати були недоступні іншим сеансам, усе буде виглядати так, начебто її не існувало взагалі.

2 Використання транзакцій в MySQL

Щоб використати транзакції, необхідно використати тип таблиць, що гарантує безпеку транзакцій, тобто InnoDB або BDB. При цьому для одержання необхідного результату можна використати кілька варіантів синтаксису.

Оператор START TRANSACTION має синоніми BEGIN й BEGIN WORK. Можна віддати перевагу якусь із зазначених форм, щоб код був сумісний з іншими використовуваними вами базами даних, але, оскільки START TRANSACTION пропонується стандартом SQL-99, рекомендується використати саме його.

Режим автоматичної фіксації

Як правило, MySQL працює в режимі автоматичної фіксації транзакцій (режим autocommit). Кожен виконуваний запит насправді ізолюється у вигляді транзакції. Ви можете інтерпретувати це, як автоматичне додавання команд START TRANSACTION й COMMIT до кожного із запитів. Пари наступних простих запитів

```
update account set balance = balance - 1000 where number = 2;
update account set balance = balance + 1000 where number = 1;
```

виконується так, начебто вони записані у вигляді

```
start transaction;
```

update account set balance = balance - 1000 where number = 2;

commit;

start transaction;

update account set balance = balance + 1000 where number = 1;

commit;

Зверніть увагу на те, що якщо ви надрукуєте вручну

start transaction;

нічого зафіксовано не буде доти, поки ви вручну не введете

commit;

Автоматичну фіксацію можна скасувати за допомогою команди SET:

set autocommit=0;

А як ви вже, напевно, догадалися, що випливає команда поверне MySQL у режим автоматичної фіксації транзакцій:

set autocommit=l;

Змінна autocomrait є локальною для кожного сеансу, тому зміна зазначеного режиму буде впливати тільки на запити вашого сеансу до його завершення.

Якщо виключити режим autocommit, то для запуску транзакції не потрібно викликати START TRANSACTION. Дуже важливо при цьому не забувати час від часу викликати COMMIT, щоб фіксувати зміни, які ви вносите в базу даних.

Незалежно від того, включений або виключений режим autocommit, у деяких ситуаціях зміни будуть фіксуватися автоматично. Якщо ви використаєте тип таблиць, що не забезпечує безпеку транзакцій (наприклад, MyISAM), всі ваші зміни будуть негайно зафіксовані, незалежно від установки autocommit. Ви можете групувати свої оператори за допомогою START TRANSACTION й COMMIT, але у випадку таблиць, що не забезпечують безпеку транзакцій, це не буде мати ніякого значення. Можна навіть викликати ROLLBACK, і це не породить помилки - тільки ніяк не вплине на зміни, які вже внесені у відповідну таблицю. Це може придатися при перевірці коду або завантаженню дамп-файлу на сервери з різними типами таблиць.

Для таблиць, що забезпечують безпеку транзакцій, деякі дії (крім безпосереднього уведення COMMIT) автоматично викликають запуск COMMIT. Наприклад, вимога блокування неявно викликає фіксацію всіх ще невиконаних запитів.

3 Модель транзакцій InnoDB

Системи керування базами даних з підтримкою транзакцій служать, загалом , одним цілям, але розрізняються підходами до рішення відповідних завдань. Щоб ізолювати транзакції, InnoDB використає дрібномасштабний механізм блокування на рівні рядків. Це значить, що різні транзакції можуть виконуватися для однієї й тієї ж таблиці одночасно, якщо вони здійснюють тільки читання або використають різні рядки для запису.

Незафіксовані зміни блокують для інших потоків тільки порушені рядки, а не всю таблицю. Це забезпечує високу продуктивність InnoDB і ряд інших характеристик, які звичайно очікуються від сучасної системи керування реляційними базами даних (СУРБД). Однієї з таких характеристик є відповідність ACI.

Відповідність вимогам ACI

Важливим терміном у базах даних, що ми не ще визначили, є абревіатура ACI (Atomicity (атомарність), Consistency (цілісність), Isolation (ізольованість) і Durability (стійкість)). Багато говорилося про те, що з таблицями MylSAM система MySQL не проходить "тест на ACI". З таблицями InnoDB система MySQL відповідає вимогам ACI.

Атомарність означає, що транзакції є атомарними й неподільними. Або всі пропоновані транзакцією зміни зберігаються в базі даних, або не зберігається жодне з них. У випадку зовнішньої помилки ідеальним рішенням було б, мабуть, завершення всіх уже, що почалися транзакцій, однак прийнятний також повний відкіт назад.

Цілісність означає, що операції перетворюють базу даних з одного припустимого стану в інше. Не повинне бути проміжних стадій, коли дані виявляються неузгодженими. База даних повинна відкидати дії, що порушують

обмеження, що забезпечують її цілісність. Якщо ви працюєте з банківськими рахунками клієнтів банку, повинне бути неможливим створення рахунку для неіснуючого клієнта й точно так само повинне бути неможливим видалення клієнта з таблиці клієнтів, якщо в таблиці рахунків ще є рахунки, пов'язані з ним.

Ізольованість означає, що транзакції не впливають одна на іншу в процесі їхнього виконання. Кожна транзакція повинна мати можливість уважати себе єдиним об'єктом, наділеним повноваженнями читати й змінювати дані. На практиці це звичайно не так, але для досягнення відповідної ілюзії використається блокування. Залежно від бази даних й її настроювань, на практиці ви можете мати різні рівні ізольованості (більш докладно це обговорюється нижче, у розділі "Ізольованість транзакцій").

Стійкість означає, що після того, як транзакція зафіксована в базі даних, її результати стають постійними. Цю вимогу легко задовольнити в простій програмі, але в складній структурі СУРБД, що використає блокування й багатоваріантність для організації багатокористувацького доступу й кешування з метою підвищення продуктивності, це є дійсним мінним полем. Крім того, стійкість припускає, що повинна існувати можливість відновлення поточного стану бази даних у випадку відмови системи. Якщо під час пересилання транзакції від клієнта до бази даних і запису цієї транзакції на диск відбудеться збій електроживлення, відмова жорсткого диска або якась інша катастрофа, ми повинні мати можливість на основі резервної копії й файлів реєстрації подій привести базу даних назад до стану, що передували катастрофі, а можливо, і обробити транзакції, які вже були зареєстровані, але ще не виконані або не зафіксовані.

При використанні таблиць InnoDB (або таблиць BerkeleyDB) MySQL відповідає вимогам ACI. Використання синтаксису транзакцій забезпечує атомарність. Обмеження транзакцій і зовнішніх ключів дають цілісність. Ви можете вибрати рівень взаємної ізольованості транзакцій. Бінарна реєстрація й засоби відновлення забезпечують стійкість. (Використовуючи реплікацію, ви,

можете створити винятково надійну систему, на яку не будуть впливати відмови її окремих елементів.)

Питання та завдання до контролю знань студентів

1) Розглянемо наступний запит повнотекстового пошуку:

```
select title
from article
where match (title, body)
against (' +php +pdfpdflib>tutorial ~reference'
IN  BOOLEAN  MODE);
```

Які з наступних тверджень щодо цього запиту є помилковими?

а) важливість терміне 'tutorial1 для пошуку підвищена;

б) оцінка результату, що містить 'reference', підвищується;

в) результат пошуку повинен містити 'php';

г) результат пошуку, що містить ' pdflib', буде оцінюватися більше високо, чим результат, що не містить зазначений рядок.

2) При виключеному режимі autocommit транзакції будуть фіксуватися

а) при виклику COMMI;

б) при запиті блокування;

в) при виконанні умов пп. а) і б);

г) при жодному з вищевказаних умов.

3) Атомарність означає, що

а) або виконується весь уміст транзакції, або не виконується нічого;

б) операції перетворять базу даних з одного погодженого стану в інше;

в) транзакції не впливають одна на іншу;

г) результати зафіксованої транзакції повинні бути перманентними.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 33

з навчальної дисципліни «Бази даних»

Тема Блокування в середовищі MySQL. Управління блокуваннями.

Мета заняття Ввести поняття блокування таблиць та управління блокуваннями в MySQL

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Використання блокування.

2 Ізольованість транзакцій.

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Використання блокування

Альтернативним способом одержання деяких можливостей транзакцій є ручне блокування й розблокування таблиць.

Якщо необхідно записати код для депозиту банківського рахунку у вигляді двох окремих транзакцій, це можна зробити так:

lock tables account write;

```
select balance from account where number = 2;
update account set balance = 1500 where number = 2;
unlock tables;
```

Виклик LOCK TABLES означає спробу блокувати всі таблиці, перераховані в даному операторі, щоб поточний потік міг працювати з ними, не побоюючись зовнішніх впливів. Виклик UNLOCK TABLES скасовує всі блокування, призначені даним потоком. Скасувати блокування просто. При цьому варто враховувати тільки те, що заблокувавши таблицю, ви повинні відкрити її якнайшвидше , щоб зменшити колізії для інших потоків. Блокування - більше складна проблема.

Необхідно запросити всі необхідні блокування відразу. У попередньому прикладі було потрібна одне блокування, але якщо потрібен доступ до безлічі таблиць або навіть до декількох псевдонімів однієї таблиці, прийде додати їх у той же виклик. Наприклад:

```
lock tables account write, account as a read,
other table low_priority write;
```

Виклик LOCK TABLES знімає всі встановлені до цього блокування, так що спроби нагромадити потрібні блокування за допомогою декількох операторів не мають змісту - усе раніше певні блокування будуть скасовані й ви одержите тільки ті, які будуть зазначені в заключному операторі.

Є два основних типи блокування: блокування для читання й блокування для запису. Якщо потрібен доступ до таблиці для запису в неї й ви не можете дозволити іншим потокам використати таблицю в той же час, блокування для запису заборонить будь-якому іншому потоку як читання таблиці, так і запис у неї, поки ви це блокування не знімете. Блокування для читання є менш строгим. Якщо необхідно тільки прочитати дані таблиці, немає нічого небезпечного в дозволі іншим потокам читати дані з таблиці одночасно. Блокування для читання забороняє іншим потокам тільки запис у таблицю, поки ваш потік здійснює блокування.

Блокування для запису може також позначатися як low_priority. У будь-якій системі, що використає блокування, включаючи MySQL, є правила, Що

Дозволяють вирішити, хто одержить блокування першим при конфлікті запитів. Як правило, MySQL надає пріоритет запитам блокування для запису перед запитами блокування для читання, щоб модифікація даних виконувалася якнайшвидше . Якщо таке поводження вас не влаштовує, можна встановити низький пріоритет блокування для запису, як це зроблено для таблиці othertable у попередньому прикладі. Тут, однак, таїться підводний камінь. Щораз при запиті блокування вам, можливо, прийдеться чекати, поки вона буде надана. Низькопріорітетне блокування буде надано тільки тоді, коли не виявиться інших потоків, що запросили блокування для читання або для запису з тією же таблицею. На дуже завантаженому сервері цього може не відбутися ніколи.

Швидше за все, ви не занадто часто будете вручну контролювати блокування так, як показано тут, але є певні ситуації, коли рекомендується саме ручний контроль. Якщо у вас є додаток, що вимагає дуже високої продуктивності, але рідко використає транзакції, ви можете вибрати який-небудь швидкий тип таблиць, що не забезпечує безпеку транзакцій, і скористатися блокуванням для рішення проблем із транзакціями.

Іншою типовою ситуацією, коли вам може знадобитися LOCK TABLES, є безпосередня робота з файлами даних MySQL. Наприклад, щоб бути впевненими, що файли на диску залишаються погодженими і незміненими, поки виконується їхнє резервне копіювання, необхідно встановити для них блокування.

Саме головне, що варто пам'ятати про блокування, - це те, що зняти блокування потрібно якнайшвидше , оскільки інші системи й користувачі будуть змушені чекати. Деякі завдання, що вимагають блокування під час їхнього виконання (наприклад, реіндексація або резервне копіювання більших файлів), можуть віднімати дуже багато часу.

2 Ізольованість транзакцій

Таблиці InnoDB пропонують чотири різних рівні ізольованості транзакцій. У порядку від найдужчого до слабкого, рівні ізольованості можуть бути наступними:

– упорядкування (serializable);

– повторюване читання (repeatableread);

– читання підтвердженого (readcommitted);

– читання непідтвердженого (readuncommitted).

Як і для багатьох інших опцій, тут теж доводиться вибирати між стійкістю й продуктивністю.

Упорядкування (serializable) - це ідеал з погляду чистоти й стійкості. З упорядкуванням читання й запис у базі даних повинні здаватися виконуваними по черзі, коли записувані зміни вносяться повністю до початку наступного читання. При цьому транзакції не завжди будуть виконуватися в неперемежованій послідовності - багато транзакцій не заважають одна інший, але у випадку колізії будуть заважати. Блокування й очікування разом з непродуктивними зусиллями на пророкування можливих колізій перетворюють упорядкування в самий повільний режим ізольованості. Якщо ви захочете використати цей режим, скористайтеся наступною командою:

set transaction isolation level serializable;

Для InnoDB рівнем ізольованості за замовчуванням є *повторюване читання* (repeatableread). У цьому режимі ізольованості кожна транзакція працює в ізольованій версії таблиці, де кожен рядок залишається в тім виді, у якому вона перебувала перед початком транзакції. При читанні будь-якого рядка гарантується повторюваність результату. Якщо викликати

select * from account where nurnber=l;

на початку транзакції й виконувати той же запит у рамках транзакції пізніше, ви одержите той самий результат. Проте існує небезпека так називаного *фантомного читання*. Можливо, що інша транзакція, що була тільки що зафіксована, додала в таблицю нові рядки. Якщо виконати той самий запит з деякою умовою двічі, наприклад

select * from account where balance>1000;

то цілком можливо, що в другий раз ви одержите нові рядки - фантомні.

На практиці фантомне читання в MySQL повинне спостерігатися винятково рідко. Для рішення цієї проблеми механізм InnoDB використає алгоритм,

називаний *блокуванням наступного ключа*, але стовпець, що використається у відповідній умові, повинен бути індексований. Ви, імовірно, уже знаєте, що InnoDB пропонує блокування на рівні рядків. Коли транзакція використає рядок, вона блокує її, щоб транзакція могла ізолюватися від інших. Разом з таким блокуванням рядків блокування наступного ключа заблокує й порожнечі між рядками, знайдені в індексі. У результаті такого підходу до дозволу проблеми фантомного читання лише невеликому числу систем дійсно може знадобитися режим упорядкування для ізольованості.

Якщо встановити для сервера режим *читання підтвердженого* (readcommitted), транзакції вже не будуть занадто ізольованими. Якщо виконати запит і повторити його пізніше в рамках тієї ж транзакції, у другий раз ви одержите інший результат, якщо за цей час інша транзакція змінить дані й буде зафіксована. Якщо вам буде потрібно встановити цей режим, що відповідає команда повинна виглядати так:

set transaction isolation level read committed;

На самому слабкому рівні ізольованості, у режимі *читання непідтвердженого* (readuncommitted), очевидно вже не тільки те, що транзакції більше не ізольовані, не забезпечують цілісність й, таким чином, відповідність ACI, але й те, що транзакцій по суті взагалі мати неможливо, У цьому режимі транзакції можуть читати зміни, які вносять інші транзакції, до того, як ці зміни будуть підтверджені (тобто зафіксовані). Це називають *"брудним" читанням*. Ви можете допустити це тільки при винятково незвичайних умовах, наприклад, коли ви знаєте, що всі активні потоки будуть або тільки читати, або тільки записувати дані, але не те й інше одночасно. Щоб установити режим читання непідтвердженого, використайте наступну команду:

set transaction isolation level read uncommitted;

У таблиці 33.1 приводяться характеристики кожного з розглянутих вище режимів ізольованості.

Таблиця 33.1 – Характерисики режимів ізольованості

	"Брудне" читання	Неповторне читання	Фантомне читання
Читання непідтвердженого (read uncommitted)	Можливо	Можливо	Можливо
Читання підтвердженого (read committed)	Неможливо	Можливо	Можливо
Повторюване читання (repeatable read)	Неможливо	Неможливо	Можливо (але малоймовірно)
Упорядкування (seriatizable)	Неможливо	Неможливо	Неможливо

Питання та завдання до контролю знань студентів

1) Ізольованість означає, що
   а) або виконується весь уміст транзакції, або не виконується нічого;
   б) операції перетворять базу даних з одного погодженого стану в інше;
   в) транзакції не впливають одна на іншу;
   г) результати зафіксованої транзакції повинні бути перманентними.
2) Стійкість означає, що
   а) або виконується весь уміст транзакції, або не виконується нічого;
   б) операції перетворять базу даних з одного погодженого стану в інше;
   в) транзакції не впливають одна на іншу;
   г) результати зафіксованої транзакції повинні бути перманентними.
3) У режимі повторюваного читання
   а) можливо "брудне" читання;
   б) можливо неповторювальне читання;
   в) можливо фантомне читання;
   г) неможливо нічого із зазначеного в пп. а)~в).

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 34

з навчальної дисципліни «Бази даних»

Тема Поняття функції. Функції користувача.

Мета заняття Розглянути існуючі типи вбудованих функцій та сворення користувацьких функцій в MySQL

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Рядкові функції
2 Числові функції
3 Функції дати
4 Розширені функції
5 Користувацькі функції в MySQL

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Рядкові функції

MySQL підтримує багато вбудованих функцій, які беруть один або кілька вхідних даних і повертають один вихід. У MySQL ми маємо різні типи функцій:

1) Рядкові функції

2) Числові функції

3) Функції дати

4) Розширені функції

Вони детально пояснюються.

Функції, які використовуються для обробки рядків, відомі як функції strings, деякі приклади та використання рядкових функцій у MySQL:

CHAR_LENGTH(рядок): Це рядкова функція, яка використовується для обчислення довжини рядків. Щоб зрозуміти це, розглянемо приклад, у нас є таблиця з назвою emp_data, ми знайдемо довжину стовпця імен її співробітників за допомогою команди:

```
mysql> SELECT emp_name, CHAR_LENGTH(emp_name) FROM emp_data;
+----------+-----------------------+
| emp_name | CHAR_LENGTH(emp_name) |
+----------+-----------------------+
| Paul     |                     4 |
| Sophia   |                     6 |
| John     |                     4 |
+----------+-----------------------+
3 rows in set (0.01 sec)

mysql>
```

UPPER (рядок): Це рядкова функція, яка використовується для перетворення алфавітів у верхніх літерах. Щоб зрозуміти це, ми знову розглянемо попередній приклад таблиці emp_data:

```
mysql> SELECT emp_name, UPPER(emp_name) FROM emp_data;
+----------+-----------------+
| emp_name | UPPER(emp_name) |
+----------+-----------------+
| Paul     | PAUL            |
| Sophia   | SOPHIA          |
| John     | JOHN            |
+----------+-----------------+
3 rows in set (0.00 sec)

mysql>
```

SUBSTR(): Він використовується для віднімання символів з рядка, ви повинні згадати початкову та кінцеву точки віднімання символів. Щоб зрозуміти це, візьмемо рядок «слухняний» і віднімемо «ient» частина за допомогою команди:

```
mysql> SELECT SUBSTR("Obedient", 1, 4);
+---------------------------+
| SUBSTR("Obedient", 1, 4) |
+---------------------------+
| Obed                      |
+---------------------------+
1 row in set (0.00 sec)

mysql>
```

У наведеному вище прикладі цифри «1» і «4» позначали першу і четверту літери слова «слухняний».»

2 Числові функції

Ці функції використовуються для обробки числових даних, є багато функцій, деякі з них наведені нижче:

SUM(): Використовується для виконання додавання над наведеними даними, щоб зрозуміти це, розглянемо приклад, передаємо дані стовпця total_price таблиці, Customer_data та обчисліть його суму за допомогою функція:

```
mysql> SELECT customer_name ,SUM(total_price) FROM Customer_data
GROUP BY customer_name;
+---------------+------------------+
| customer_name | SUM(total_price) |
+---------------+------------------+
| John          |               13 |
| Paul          |               43 |
+---------------+------------------+
2 rows in set (0.00 sec)

mysql>
```

МАКС(): Використовується для того, щоб дізнатися максимальне значення з заданих числових даних, наприклад, у таблиці Customer_data ми дізнаємося максимальну загальну вартість за допомогою MAX(), запускаємо команду:

```
mysql> SELECT customer_name, MAX(total_price) FROM Customer_data
GROUP BY customer_name;
+---------------+------------------+
| customer_name | MAX(total_price) |
+---------------+------------------+
| John          |                7 |
| Paul          |               23 |
+---------------+------------------+
2 rows in set (0.00 sec)

mysql>
```

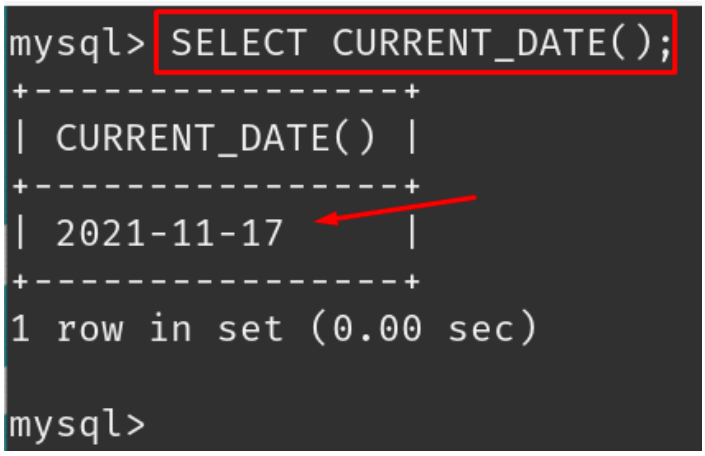
POWER(): Він використовується для визначення ступеня цілих чисел шляхом визначення основного цілого числа та числа, за яким потрібно обчислити його потужність. Давайте розглянемо приклад, ми знайдемо ступінь 10 на 6 за допомогою функції POWER():

```
mysql> SELECT POWER (10,6);
+--------------+
| POWER (10,6) |
+--------------+
|      1000000 |
+--------------+
1 row in set (0.00 sec)

mysql>
```

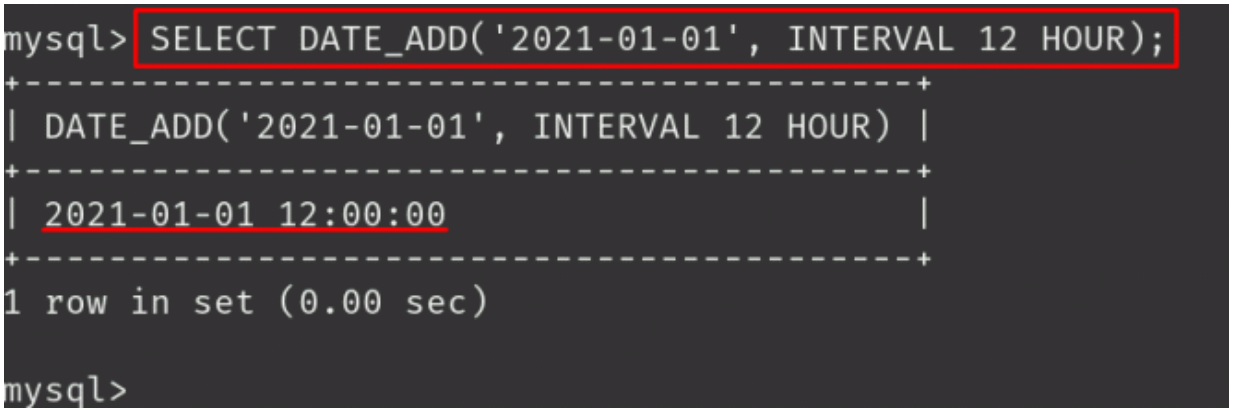
3 Функції DATE

Ці функції використовуються для вилучення або обробки дати та часу в MySQL. Деякі приклади таких функцій пояснюються тут.

ПОТОЧНА ДАТА(): Він використовується для вилучення поточної дати в MySQL, для кращого розуміння виконайте таку команду:



ТИЖЕНЬ(): Він використовується для відображення тижня року, ми передаємо дату функції, і вона скаже вам, у якому тижні року існує ця дата. Щоб зрозуміти це, ми передаємо поточну дату за допомогою функції now():

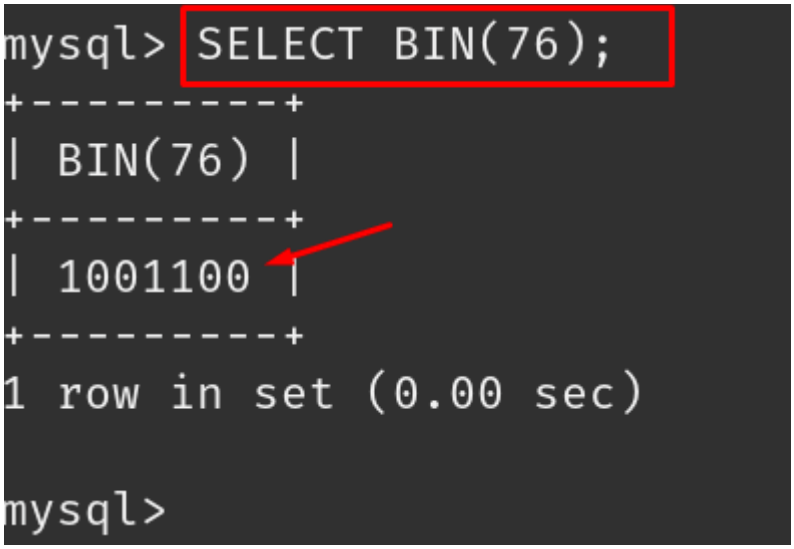


CURRENT_TIMESTAMP(): Він використовується для відображення поточного часу та дати в MySQL, давайте зрозуміємо це, виконавши таку команду:

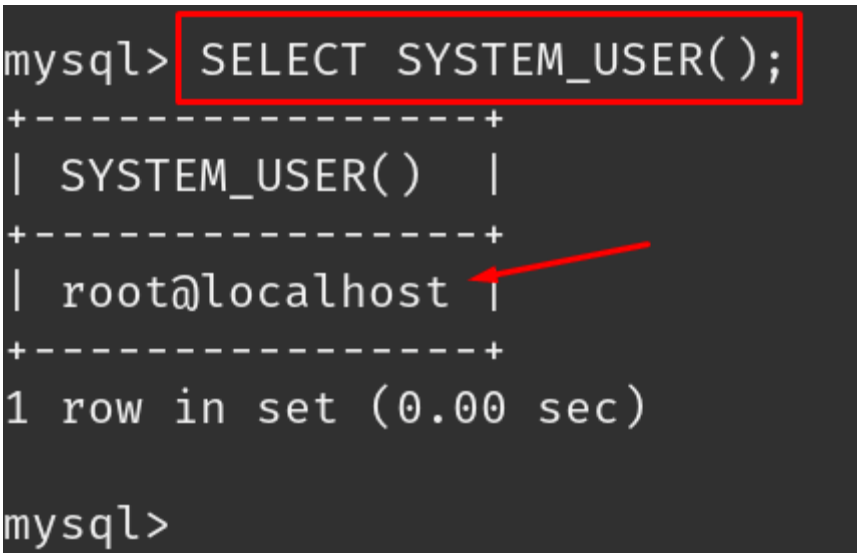
4 Розширені функції

Є деякі інші функції, які використовуються для різних функцій, наприклад для перетворення чисел у двійкові. Деякі з цих функцій пояснені.

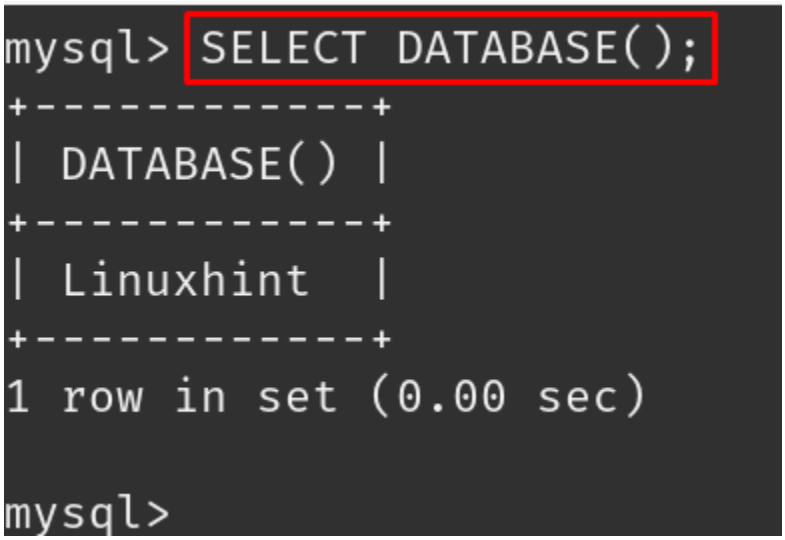
BIN(): Він використовується для перетворення числа в двійковий, щоб зрозуміти використання цієї функції, розглянемо приклад:



SYSTEM_USER(): Він використовується для вилучення інформації про ім'я користувача системи або ім'я хоста, наприклад, ми запускаємо команду за допомогою функції в MySQL:



БАЗА ДАНИХ(): Він використовується для вилучення інформації про назву бази даних, яка використовується, наприклад, запускаємо команду:



5 Користувацькі функції в MySQL

Ми також можемо створювати спеціальні функції, як інші мови програмування, щоб уникнути повторення одного і того ж коду з тією ж метою. У MySQL загальний синтаксис створення налаштованої функції:

```
mysql> DELIMITER $$
mysql> CREATE FUNCTION Clock()
    -> RETURNS TIME
    -> BEGIN
    -> RETURN NOW();
    -> END$$
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql>
```

Пояснення наведеного вище синтаксису:

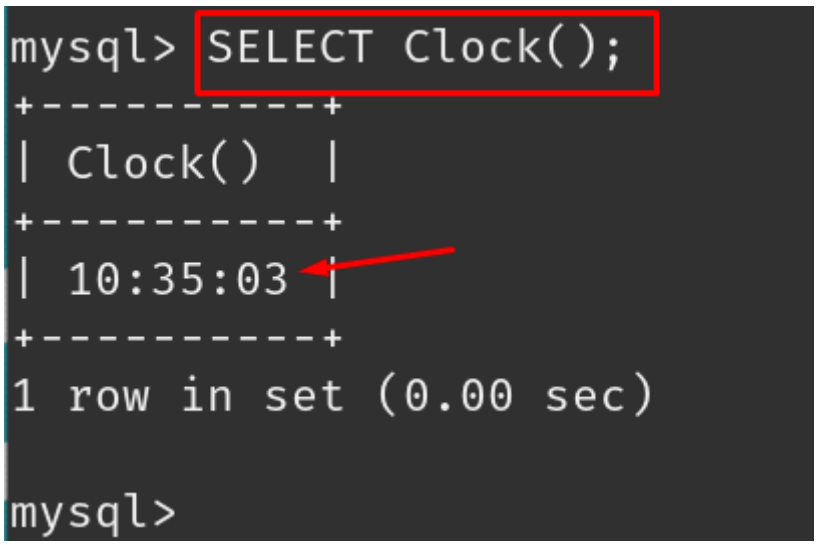
Якщо ви не використовуєте DELIMITER, MySQL використовує його за замовчуванням.

Використовуйте речення CREATE FUNCTION, щоб створити функцію та призначити їй відповідне ім’я.

Всередині дужок оголошуються його параметри, які потрібно передати у функції, а також їхні типи даних.

Код повинен бути написаний між блоками BEGIN і END.

Щоб зрозуміти це, ми створюємо функцію відображення поточного часу замість того, щоб використовувати NOW() знову і знову, ми будемо викликати Clock():



Питання та завдання до контролю знань студентів

1) Функція, повертає рядок, що є результатом об'єднання двох або більше строкових значень.

а) CAST

б) CHOOSE

в) CEILING

г) EOMONTH

д) CONCAT.

2) DTOS(), STR(), CHR(), VAL() Ці функції слід віднести до…

а) символьних функцій;

б) функцій перетворення типів

в) функцій дати

г) функцій визначення типів

д) числових функцій

3) MS SQL. Функція DATEADD…

а) повертає текстові назви, що відповідають заданій даті

б) повертає визначений фрагмент із заданої дати

в) повертає поточну дату та час, що перетворені в формат універсального синхронізованого часу

г) додає до дати деяку величину

д) повертає рік.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 35

з навчальної дисципліни «Бази даних»

Тема Збережені процедури.

Мета заняття Визначити понятя процедури, типи процедур та принципи їх створення в MySQL

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Поняття процедури

2 Збережені процедури в середовищі MS SQL Server. Типи процедур.

3 Створення, зміна та видалення збережених процедур

4 Виконання процедури

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Поняття процедури

Збережені процедури представляють собою групи пов'язаних між собою операторів SQL, застосування яких робить роботу програміста більш легкою і гнучкою, оскільки виконати збережену процедуру часто виявляється набагато

простіше, ніж послідовність окремих операторів SQL. Збережені процедури представляють собою набір команд, що складається з одного або декількох операторів SQL або функцій і зберігається в базі даних в відкомпілювався вигляді. Виконання в базі даних збережених процедур замість окремих операторів SQL дає користувачеві наступні переваги:

– необхідні оператори вже містяться в базі даних;

– всі вони пройшли етап синтаксичного аналізу і знаходяться в виконуваному форматі; перед виконанням процедури, що SQL Server генерує для неї план виконання, виконує її оптимізацію та компіляцію;

– процедури, що підтримують модульне програмування, так як дозволяють розбивати великі завдання на самостійні, більш дрібні і зручні в управлінні частини;

– процедури, що можуть викликати інші процедури, що і функції;

– процедури, що можуть бути викликані з прикладних програм інших типів;

– як правило, збережені процедури виконуються швидше, ніж послідовність окремих операторів;

– процедури, що простіше використовувати: вони можуть складатися з десятків і сотень команд, але для їх запуску достатньо вказати лише ім'я потрібної збереженої процедури. Це дозволяє зменшити розмір запиту, що посилається від клієнта на сервер, а значить, і навантаження на мережу.

Зберігання процедур в тому ж місці, де вони виконуються, забезпечує зменшення обсягу переданих по мережі даних та підвищує загальну продуктивність системи. Застосування збережених процедур спрощує супровід програмних комплексів і внесення змін до них. Зазвичай всі обмеження цілісності у вигляді правил і алгоритмів обробки даних реалізуються на сервері баз даних і доступні кінцевому додатком у вигляді набору процедур, які і представляють інтерфейс обробки даних. Для забезпечення цілісності даних, а також з метою безпеки, додаток зазвичай не отримує прямого доступу до даних - вся робота з ними ведеться шляхом виклику тих чи інших процедур.

Подібний підхід робить вельми простий модифікацію алгоритмів обробки даних, негайно ж стають доступними для всіх користувачів мережі, і забезпечує можливість розширення системи без внесення змін до сам додаток: досить змінити збережену процедуру на сервері баз даних. Розробнику не потрібно перекомпілювати додаток, створювати його копії, а також інструктувати користувачів про необхідність роботи з новою версією. Користувачі взагалі можуть не підозрювати про те, що в систему внесені зміни.

Збережені процедури існують незалежно від таблиць або будь-яких інших об'єктів баз даних. Вони викликаються клієнтською програмою, інший збереженої процедурою або тригером. Розробник може керувати правами доступу до інформації, що зберігається процедурою, дозволяючи або забороняючи її виконання. Змінювати код процедури, що дозволяється тільки її власнику або члену фіксованою ролі бази даних. При необхідності можна передати права володіння нею від одного користувача до іншого.

2 Збережені процедури в середовищі MS SQL Server. Типи процедур.

При роботі з SQL Server користувачі можуть створювати власні процедури, що реалізують ті чи інші дії. Збережені процедури є повноцінними об'єктами бази даних, а тому кожна з них зберігається в конкретній базі даних. Безпосередній виклик збереженої процедури можливий, тільки якщо він здійснюється в контексті тієї бази даних, де знаходиться процедура.

Типи процедур.

У SQL Server є кілька типів збережених процедур.

– Системні збережені процедури призначені для виконання різних адміністративних дій. Практично всі дії з адміністрування сервера виконуються з їх допомогою. Можна сказати, що системні збережені процедуриявляются інтерфейсом, що забезпечує роботу з системними таблицями, яка, в кінцевому рахунку, зводиться до зміни, додаванню, видалення і вибірці даних з системних таблиць як для користувача, так і системних баз даних. Сістемниехранімие процедури мають префікс sp_, зберігаються в системній базі даних і можуть бути викликані в контексті будь-якої іншої бази даних.

– Користувальницькі збережені процедури реалізують ті чи інші дії. Збережені процедури - повноцінний об'єкт бази даних. Внаслідок цього кожна збережена процедура розташовується в конкретній базі даних, де і виконується.

– Тимчасові збережені процедури існують лише деякий час, після чого автоматично знищуються сервером. Вони діляться на локальні і глобальні. Локальні тимчасові процедури, що можуть бути викликані тільки з того з'єднання, в якому створені. При створенні такої процедури їй необхідно дати ім'я, що починається з одного символу #. Як і всі тимчасові об'єкти, збережені процедури цього типу автоматично видаляються при відключенні користувача, перезапуску або зупинці сервера. Глобальні тимчасові процедури, що доступні для будь-яких з'єднань сервера, на якому є така ж процедура. Для її визначення достатньо дати їй ім'я, яке починається з символів ##. Видаляються ці процедури при перезапуску або зупинці сервера, а також при закритті з'єднання, в контексті якого вони були створені.

3 Створення, зміна та видалення збережених процедур

Створення збереженої процедури передбачає вирішення наступних завдань:

– визначення типу створюваної процедури, що: тимчасова або призначена для користувача. Крім цього, можна створити свою власну системну збережену процедуру, призначивши їй ім'я з префіксом sp_ і помістивши її в системну базу даних. Така процедура буде доступна в контексті будь-якої бази даних локального сервера;

– планування прав доступу. При створенні збереженої процедури слід враховувати, що вона буде мати ті ж права доступу до об'єктів бази даних, що і створив її користувач;

– визначення параметрів процедури. Подібно процедурам, які входять до складу більшості мов програмування, збережені процедури можуть мати вхідними та вихідними параметрами;

– розробка коду збереженої процедури. Код процедури може містити послідовність будь-яких команд SQL, включаючи виклик інших процедур.

Створення нової та зміна наявної процедури, що здійснюється за допомогою наступної команди:

<Визначення_процедури> :: =

{CREATE | ALTER} PROC [EDURE] ім’я_процедури

[; Номер]

[{@ Ім’я_параметра тип_даних} [VARYING]

[= Default] [OUTPUT]] [, ... n]

[WITH {RECOMPILE | ENCRYPTION | RECOMPILE,

ENCRYPTION}]

[FOR REPLICATION]

AS

sql_оператор [... n]

Розглянемо параметри даної команди.

Використовуючи префікси sp_, #, ##, створювану процедуру можна визначити як системної або тимчасовою. Як видно з синтаксису команди, не допускається вказувати ім'я власника, якому належатиме створювана процедура, а також ім'я бази даних, де вона повинна бути розміщена. Таким чином, щоб розмістити створювану збережену процедуру в конкретній базі даних, необхідно виконати команду CREATE PROCEDURE в контексті цієї бази даних. При зверненні з тіла збереженої процедури до об'єктів тієї ж бази даних можна використовувати укорочені імена, т. Е. Без вказівки імені бази даних. Коли ж потрібно звернутися до об'єктів, розташованих в інших базах даних, вказівка імені бази даних обов'язково.

Номер в імені - це ідентифікаційний номер збереженої процедури, однозначно визначає її в групі процедур. Для зручності управління процедурами логічно однотипні процедури, можна групувати, привласнюючи їм однакові імена, але різні ідентифікаційні номери.

Для передачі вхідних і вихідних даних в створюваній збереженій процедурі можуть використовуватися параметри, імена яких, як і імена локальних змінних, повинні починатися з символу @. В одній збереженій процедурі можна задати безліч параметрів, розділених комами. У тілі

процедури не повинні застосовуватися локальні змінні, чиї імена збігаються з іменами параметрів цієї процедури.

Для визначення типу даних, який буде мати відповідний параметр збереженої процедури, годяться будь-які типи даних SQL, включаючи певні користувачем. Однак тип даних CURSOR може бути використаний тільки як вихідний параметр збереженої процедури, тобто із зазначенням ключового слова OUTPUT.

Наявність ключового слова OUTPUT означає, що відповідний параметр призначений для повернення даних з збереженої процедури. Однак це зовсім не означає, що параметр не підходить для передачі значень в збережену процедуру. Вказівка ключового слова OUTPUT наказує сервера при виході з процедури, що привласнити поточне значення параметра локальної змінної, яка була вказана при виклику процедури в якості значення параметра. Відзначимо, що при вказівці ключового слова OUTPUT значення відповідного параметра при виклику процедури може бути поставлено лише за допомогою локальної змінної. Не дозволяється використання будь-яких виразів або констант, припустимий для звичайних параметрів.

Ключове слово VARYING застосовується спільно з параметром OUTPUT, що має тип CURSOR. Воно визначає, що вихідним параметром буде результуюча безліч.

Ключове слово DEFAULT є значення, яке буде приймати відповідний параметр за замовчуванням. Таким чином, при виклику процедури можна не вказувати явно значення відповідного параметра.

Так як сервер кеширує план виконання запиту і компілює код, при наступному виклику процедури будуть використовуватися вже готові значення. Однак в деяких випадках все ж потрібно виконувати перекомпіляцію коду процедури. Вказівка ключового слова RECOMPILE наказує системі створювати план виконання процедури при кожному її виклику.

Параметр FOR REPLICATION затребуваний при реплікації даних і включенні створюваної процедури, як статті в публікацію.

Ключове слово ENCRYPTION наказує сервера виконати шифрування коду збереженої процедури, що може забезпечити захист від використання авторських алгоритмів, що реалізують роботу збереженої процедури.

Ключове слово AS розміщується на початку власне тіла збереженої процедури, тобто набору команд SQL, за допомогою яких і буде реалізовуватися ту чи іншу дію. У тілі процедури можуть застосовуватися практично всі команди SQL, оголошуватися транзакції, встановлюватися блокування і викликатися інші процедури, що зберігаються. Вихід з процедури, що можна здійснити за допомогою команди RETURN.

Видалення збереженої процедури здійснюється командою:

DROP PROCEDURE {ім'я_процедури} [... n]

4 Виконання процедури

Для виконання процедури використовується команда:

[[EXEC [UTE] імя_процедури [; номер]

[[@ Імя_параметра =] {значення | @ Ім'я_змінної}

[OUTPUT] | [DEFAULT]] [, ... n]

Якщо виклик збереженої процедури не є єдиною командою в пакеті, то присутність команди EXECUTE обов'язково. Більш того, ця команда потрібна для виклику процедури з тіла іншої процедури або тригера.

Використання ключового слова OUTPUT при виклику процедури дозволяється тільки для параметрів, які були оголошені при створенні процедури з ключовим словом OUTPUT.

Коли ж при виклику процедури для параметра вказується ключове слово DEFAULT, то буде використано значення за замовчуванням. Природно, вказане слово DEFAULT дозволяється тільки для тих параметрів, для яких визначено значення за замовчуванням.

З синтаксису команди EXECUTE видно, що імена параметрів можуть бути опущені при виклику процедури. Однак в цьому випадку користувач повинен вказувати значення для параметрів в тому ж порядку, в якому вони перераховувалися при створенні процедури. Присвоїти параметру значення за

замовчуванням, просто пропустивши його при перерахуванні можна. Якщо ж потрібно опустити параметри, для яких визначено значення за замовчуванням, досить явної вказівки імен параметрів при виклику збереженої процедури. Більш того, таким способом можна перераховувати параметри і їх значення в довільному порядку.

Відзначимо, що при виклику процедури вказуються або імена параметрів зі значеннями, або тільки значення без імені параметра. Їх комбінування не допускається.

Приклад 1. Процедура без параметрів. Розробити процедуру для отримання назв і вартості товарів, придбаних Івановим.

```
CREATE PROC my_proc1
AS
SELECT Товар.Назва,
Товар.Ціна * Сделка.Кількість
AS Вартість, Клієнт.Прізвище
FROM Клієнт INNER JOIN
(Товар INNER JOIN Угода
ON Товар.КодТовару = Угода.КодТовару)
ON Клієнт.КодКлієнта = Угода.КодКлієнта
WHERE Клієнт.Прізвище = 'Іванов'
```

Процедура для отримання назв і вартості товарів, придбаних Івановим.

Для звернення до процедури можна використовувати команди:

EXEC my_proc1 або my_proc1

Процедура повертає набір даних.

Приклад 2. Процедура без параметрів. Створити процедуру для зменшення ціни товару першого сорту на 10%.

```
CREATE PROC my_proc2
AS
UPDATE Товар SET Ціна = Ціна * 0.9
WHERE Сорт = 'перший'
```

Процедура для зменшення ціни товару першого сорту на 10%.

Для звернення до процедури можна використовувати команди:

EXEC my_proc2 або my_proc2

Процедура не повертає ніяких даних.

Приклад 3. Процедура з вхідним параметром. Створити процедуру для отримання назв і вартості товарів, які придбав заданий клієнт.

```
CREATE PROC my_proc3
@k VARCHAR (20)
AS
SELECT Товар.Назва,
Товар.Ціна * Сделка.Кількість
AS Вартість, Клієнт.Прізвище
FROM Клієнт INNER JOIN
(Товар INNER JOIN Угода
ON Товар.КодТовару = Сделка.КодТовару)
ON Клієнт.КодКлієнта = Сделка.КодКлієнта
WHERE Клієнт.Прізвище=@k
```

Процедура для отримання назв і вартості товарів, які придбав заданий клієнт.

Для звернення до процедури можна використовувати команди:

EXEC my_proc3 'Іванов' або

my_proc3 @ k = 'Іванов'

Приклад 12.4. Процедура з вхідними параметрами. Створити процедуру для зменшення ціни товару заданого типу відповідно до зазначеного%.

```
CREATE PROC my_proc4
@t VARCHAR (20), @p FLOAT
AS
UPDATE Товар SET Ціна = Ціна * (1 @ p)
WHERE Тип = @ t
```

Процедура для зменшення ціни товару заданого типу відповідно до зазначеного%.

Для звернення до процедури можна використовувати команди:

EXEC my_proc4 'Вафлі', 0.05 або

EXEC my_proc4 @ t = 'Вафлі', @ p = 0.05

Приклад 5. Процедура з вхідними параметрами і значеннями за замовчуванням. Створити процедуру для зменшення ціни товару заданого типу відповідно до зазначеного%.

CREATE PROC my_proc5

@t VARCHAR (20) = 'Конфети`,

@p FLOAT = 0.1

AS

UPDATE Товар SET Ціна = Ціна * (1 @ p)

WHERE Тип = @ t

Процедура з вхідними параметрами і значеннями за замовчуванням. Створити процедуру для зменшення ціни товару заданого типу відповідно до зазначеного%.

Для звернення до процедури можна використовувати команди:

EXEC my_proc5 'Вафлі', 0.05 або

EXEC my_proc5 @ t = 'Вафлі', @ p = 0.05 або

EXEC my_proc5 @ p = 0.05

У цьому випадку зменшується ціна цукерок (значення типу не вказано при виклику процедури і береться за замовчуванням).

EXEC my_proc5

В останньому випадку обидва параметри (і тип, і відсотки) не вказані при виклику процедури, їх значення беруться за замовчуванням.

Приклад 6. Процедура з вхідними та вихідними параметрами. Створити процедуру для визначення загальної вартості товарів, проданих за конкретний місяць.

CREATE PROC my_proc6

@m INT,

@s FLOAT OUTPUT

AS

SELECT @ s = Sum (Товар.Ціна * Угода.Кількість)

```
FROM Товар INNER JOIN Угода
ON Товар.КодТовару = Угода.КодТовару
GROUP BY Month (Угода.Дата)
HAVING Month (Угода.Дата) = @ m
```

Процедура з вхідними та вихідними параметрами. Створити процедуру для визначення загальної вартості товарів, проданих за конкретний місяць.

Для звернення до процедури можна використовувати команди:

```
DECLARE @st FLOAT
EXEC my_proc6 1, @ st OUTPUT
SELECT @st
```

Цей блок команд дозволяє визначити вартість товарів, проданих в січні (вхідний параметр місяць вказано рівним 1).

Створити процедуру для визначення загальної кількості товарів, придбаних фірмою, в якій працює заданий співробітник.

Спочатку розробимо процедуру для визначення фірми, де працює співробітник.

```
CREATE PROC my_proc7
@n VARCHAR (20),
@f VARCHAR (20) OUTPUT
AS
SELECT @ f = Фірма
FROM Клієнт
WHERE Прізвище = @ n
```

Приклад 7. Використання вкладених процедур. Створити процедуру для визначення загальної кількості товарів, придбаних фірмою, в якій працює заданий співробітник.

Потім створимо процедуру, підраховують загальну кількість товару, який закуплений цікавить нас фірмою.

```
CREATE PROC my_proc8
@fam VARCHAR (20),
@kol INT OUTPUT
```

```
AS
DECLARE @firm VARCHAR (20)
EXEC my_proc7 @ fam, @ firm OUTPUT
SELECT @ kol = Sum (Угода.Кількість)
FROM Клієнт INNER JOIN Угода
ON Клієнт.КодКлієнта = Угода.КодКлієнта
GROUP BY Клієнт.Фірма
HAVING Клієнт.Фірма=@firm
```

Створення процедури для визначення загальної кількості товарів, придбаних фірмою, в якій працює заданий співробітник.

Виклик процедури здійснюється за допомогою команди:

```
DECLARE @k INT
EXEC my_proc8 'Іванов', @ k OUTPUT
SELECT @k
```

Питання та завдання до контролю знань студентів

1) Де зберігається збережена процедура?
2) Може збережена процедура викликати іншу збережену процедуру?
3) Яка SQL команда викликає збережену процедуру?
4) Какие існують типи параметрів процедури?
5) Чим різняться процедури з параметрами IN, OUT та INOUT?
6) Який синтаксис змінної в MySQL?
7) У чому відмінність процедур і функцій, що зберігаються?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 36

з навчальної дисципліни «Бази даних»

Тема Визначення тригера в стандарті мови SQL. Реалізація тригерів в середовищі MySQL

Мета заняття Визначити понятя тригера, типи тригерів та принципи їх створення в MySQL

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Визначення тригера у стандарті мови SQL.

2 Реалізація тригерів серед MS SQL Server.

3 Типи тригерів.

4 Програмування тригера.

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BHV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Визначення тригера у стандарті мови SQL

Тригери є одним з різновидів процедур, що зберігаються. Їхнє виконання відбувається при виконанні для таблиці будь-якого оператора мови

маніпулювання даними (DML). Тригери використовуються для перевірки цілісності даних, а також відкату транзакцій.

Тригер – це відкомпільована SQL-процедура, виконання якої обумовлено настанням певних подій усередині реляційної бази даних. Застосування тригерів здебільшого зручне для користувачів бази даних. І все-таки їх використання часто пов'язані з додатковими витратами ресурсів на операції вводу/вывода. У тому випадку, коли тих же результатів (з набагато меншими непродуктивними витратами ресурсів) можна досягти за допомогою процедур або прикладних програм, що зберігаються, застосування тригерів недоцільно.

Тригери – спеціальний інструмент SQL-сервера, використовуваний підтримки цілісності даних у базі даних. За допомогою обмежень цілісності, правил і значень за умовчанням не завжди можна досягти потрібного рівня функціональності. Часто потрібно реалізувати складні алгоритми перевірки даних, що гарантують їхню достовірність і реальність. З іншого боку, іноді необхідно відслідковувати зміни значень таблиці, щоб належним чином змінити пов'язані дані. Тригери можна розглядати як свого роду фільтри, що набувають чинності після виконання всіх операцій відповідно до правил, стандартних значень і т.д.

Тригер являє собою спеціальний тип процедур, що зберігаються, що запускаються сервером автоматично при спробі зміни даних у таблицях, з якими тригери пов'язані. Кожен тригер прив'язується до конкретної таблиці. Усі вироблені їм модифікації даних розглядаються як транзакція. У разі виявлення помилки чи порушення цілісності даних відбувається відкат цієї транзакції. Тим самим, внесення змін забороняється. Скасовуються також зміни, зроблені тригером.

Створює тригер лише власник бази даних. Це обмеження дозволяє уникнути випадкової зміни структури таблиць, способів зв'язку з ними інших об'єктів тощо.

Тригер є дуже корисним і в той же час небезпечним засобом. При неправильній логіці його роботи можна легко знищити цілу базу даних, тому тригери необхідно дуже ретельно налагоджувати.

На відміну від звичайної підпрограми, тригер виконується неявно у разі виникнення тригерного події, причому він немає аргументів. Приведення в дію іноді називають запуском тригера. За допомогою тригерів досягаються такі цілі:

– перевірка коректності введених даних та виконання складних обмежень цілісності даних, які важко, якщо взагалі можливо, підтримувати за допомогою обмежень цілісності, встановлених для таблиці;

– видача попереджень, що нагадують необхідність виконання деяких дій при оновленні таблиці, реалізованому певним чином;

– накопичення аудиторської інформації за допомогою фіксації відомостей про внесені зміни та тих осіб, які їх виконали;

– підтримка реплікації.

Основний формат команди CREATE TRIGGER показаний нижче:

<Визначення_тригера>::=

CREATE TRIGGER имя_тригера

BEFORE | AFTER <тригерна подія>

ON <ім'я_таблиці>

[REFERENCING

<список_старих_або_нових_псевдонімів>]

[FOR EACH {ROW | STATEMENT}]

[WHEN(умова_тригера)]

<тіло_тригера>

тригерні події складаються із вставки, видалення та оновлення рядків у таблиці. У разі для тригерного події можна вказати конкретні імена стовпців таблиці. Час запуску тригера визначається за допомогою ключових слів BEFORE (тригер запускається до виконання пов'язаних з ним подій) або AFTER (після їх виконання).

Дії, що виконуються тригером, задаються для кожного рядка (FOR EACH ROW), охопленого цією подією, або тільки один раз для кожної події (FOR EACH STATEMENT).

Позначення <список_старих_або_нових_псевдонімів> відноситься до таких компонентів, як старий або новий рядок (OLD/NEW) або стара або нова таблиця (OLD TABLE/NEW TABLE). Зрозуміло, що старі значення не застосовні до подій вставки, а нові – для подій видалення.

За умови правильного використання тригери можуть стати дуже сильним механізмом. Основна їхня перевага полягає в тому, що стандартні функції зберігаються всередині бази даних та узгоджено активізуються при кожному її оновленні. Це може суттєво спростити програми. Проте слід згадати і про властиві тригеру недоліки:

– складність: при переміщенні деяких функцій до бази даних ускладнюються завдання її проектування, реалізації та адміністрування;

– прихована функціональність: перенесення частини функцій у базу даних та збереження їх у вигляді одного або декількох тригерів іноді призводить до приховання від користувача деяких функціональних можливостей. Хоча це певною мірою спрощує його роботу, але, на жаль, може стати причиною незапланованих, потенційно небажаних та шкідливих побічних ефектів, оскільки в цьому випадку користувач не може контролювати всі процеси, що відбуваються в базі даних;

– вплив на продуктивність: перед виконанням кожної команди зі зміни стану бази даних СУБД має перевірити тригерну умову з метою з'ясування необхідності запуску тригера цієї команди. Виконання подібних обчислень позначається загальної продуктивності СУБД, а моменти пікової навантаження її зниження може бути особливо помітним. Очевидно, що при зростанні кількості тригерів збільшуються накладні витрати, пов'язані з такими операціями.

Неправильно написані тригери можуть призвести до серйозних проблем, таких, наприклад, як поява "мертвих" блокувань. Тригери здатні довго блокувати безліч ресурсів, тому слід звернути особливу увагу на зведення до мінімуму конфліктів доступу.

2 Реалізація тригерів серед MS SQL Server

У реалізації СУБД MS SQL Server використовується наступний оператор створення чи зміни тригера:

```
<Визначення_тригера>::=
{CREATE | ALTER} TRIGGER имя_тригера
ON {ім'я_таблиці | ім'я_перегляду }
[WITH ENCRYPTION]
{
{ { FOR | AFTER | INSTEAD OF }
{ [DELETE] [,] [INSERT] [,] [UPDATE]}
[WITH APPEND]
[ NOT FOR REPLICATION ]
AS
   sql_оператор[...n]
} |
{ {FOR | AFTER | INSTEAD OF } { [INSERT] [,]
  [UPDATE] }
[ WITH APPEND]
[ NOT FOR REPLICATION]
AS
{ IF UPDATE(ім'я_стовпця)
[{AND | OR} UPDATE(ім'я_стовпця)] [...n]
|
IF (COLUMNS_UPDATES(){оператор_біт_обробки}
  біт_маска_зміни)
{оператор_біт_порівняння }біт_маска [...n]}
sql_оператор [...n]
}
}
```

Тригер може бути створений тільки в поточній базі даних, але допускається звернення всередині тригера до інших баз даних, у тому числі і на віддаленому сервері.

Розглянемо призначення аргументів із команди CREATE | ALTER TRIGGER

Ім'я тригера має бути унікальним у межах бази даних. Додатково можна вказати ім'я власника.

При вказівці аргументу WITH ENCRYPTION сервер виконує шифрування коду тригера, щоб ніхто, включаючи адміністратора, не міг отримати доступ до нього і прочитати його. Шифрування часто використовується для приховання авторських алгоритмів обробки даних, які є інтелектуальною власністю програміста чи комерційною таємницею.

3 Типи тригерів

У SQL Server існує два параметри, що визначають поведінку тригерів:

1) AFTER. Тригер виконується після успішного виконання команд, що викликали його. Якщо команди з будь-якої причини не можуть бути успішно завершені, тригер не виконується. Слід зазначити, що зміни даних в результаті виконання запиту користувача та виконання тригера здійснюється в тілі однієї транзакції: якщо відбудеться відкат тригера, то будуть відхилені і зміни користувача. Можна визначити кілька AFTER-тригерів кожної операції (INSERT, UPDATE, DELETE). Якщо для таблиці передбачено виконання декількох AFTER-тригерів, то за допомогою системної процедури sp_settriggerorder можна вказати, який з них буде виконуватися першим, а який останнім. За замовчуванням SQL Server всетригери є AFTER-тригерами.

2) INSTEAD OF. Тригер викликається замість виконання команд. На відміну від AFTER-тригера INSTEAD OF-тригер може бути визначений як таблиці, так перегляду. Для кожної операції INSERT, UPDATE, DELETE можна визначити лише один INSTEAD OF-тригер.

Тригери розрізняють на кшталт команд, куди вони реагують.

Існує три типи тригерів:

1) INSERT TRIGGER – запускаються під час спроби вставити дані за допомогою команди INSERT.

2) UPDATE TRIGGER – запускаються під час спроби зміни даних за допомогою команди UPDATE.

3) DELETE TRIGGER – запускаються при спробі видалення даних за допомогою DELETE.

Конструкції [DELETE] [,] [INSERT] [,] [UPDATE] та FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] визначають, яку команду реагуватиме тригер. При його створенні має бути вказана хоча б одна команда. Допускається створення тригера, реагує на дві чи всі три команди.

Аргумент WITH APPEND дозволяє створювати кілька тригерів кожного типу.

Під час створення тригера з аргументом NOT FOR REPLICATION забороняється його запуск під час виконання модифікації таблиць механізмами реплікації.

Конструкція AS sql_оператор[...n] визначає набір SQL-операторів та команд, які будуть виконані при запуску тригера.

Зазначимо, що всередині тригера не допускається виконання низки операцій, таких, наприклад, як:

– створення, зміна та видалення бази даних;

– відновлення резервної копії бази даних або журналу транзакцій.

Виконання цих команд не дозволено, оскільки вони можуть бути скасовані у разі відкату транзакції, у якій виконується тригер. Ця заборона навряд чи може якимось чином позначитися на функціональності створюваних тригерів. Важко знайти таку ситуацію, коли, наприклад, після зміни рядка таблиці потрібно буде виконати відновлення резервної копії журналу транзакцій.

4 Програмування тригера

Під час виконання команд додавання, зміни та видалення записів сервер створює дві спеціальні таблиці: inserted і deleted . Вони містять списки рядків,

які будуть вставлені або видалені після завершення транзакції. Структура таблиць вміщена і вирівняна ідентичною структурою таблиць, для якої визначається тригер. Для кожного тригера створюється свій комплект таблиць, внесених і deleted, тому ніякий інший тригер не зможе отримати доступ до них. Залежно від типу операції, що викликала виконання тригера, вміст таблиць inserted і deleted може бути різним:

– команда INSERT – у таблиці inserted містяться усі рядки, які користувач намагається вставити у таблицю; у таблиці deletedне буде жодного рядка; після завершення тригера всі рядки з таблиці, введені, перемістяться у вихідну таблицю;

– команда DELETE – у таблиці deleted будуть утримуватись всі рядки, які користувач спробує видалити; тригер може перевірити кожен рядок та визначити, чи дозволено його видалення; в таблиці введено не виявиться жодного рядка;

– команда UPDATE – при її виконанні у таблиці deleted знаходяться старі значення рядків, які будуть видалені при успішному завершенні тригера. Нові значення рядків містяться в таблиці введені. Ці рядки додадуться до вихідної таблиці після успішного виконання тригера.

Для отримання інформації про кількість рядків, яка буде змінена при успішному завершенні тригера, можна використовувати функцію @@ROWCOUNT; вона повертає кількість рядків, опрацьованих останньою командою. Слід наголосити, що тригер запускається не при спробі змінити конкретний рядок, а в момент виконання команди зміни. Одна така команда впливає на рядки рядків, тому тригер повинен обробляти всі ці рядки.

Якщо тригер виявив, що з 100 рядків, що вставляються, змінюються або видаляються, тільки один не задовольняє тим чи іншим умовам, то ніякий рядок не буде вставлено, змінено або видалено. Така поведінка обумовлена вимогами транзакції – мають бути виконані або всі модифікації, або жодної.

Тригер виконується як неявно певна транзакція, тому всередині тригера допускається застосування команд керування транзакціями. Зокрема, при виявленні порушення обмежень цілісності для переривання виконання тригера

та скасування всіх змін, які намагався виконати користувач, необхідно використовувати команду ROLLBACK TRANSACTION.

Для отримання списку стовпців, змінених під час виконання команд INSERT або UPDATE, які викликали виконання тригера, можна використовувати функцію COLUMNS_UPDATED(). Вона повертає двійкове число, кожен біт якого, починаючи з молодшого, відповідає одному стовпцю таблиці (у порядку проходження стовпців при створенні таблиці). Якщо біт встановлено значення "1", то відповідний стовпець було змінено. Крім того, факт зміни стовпця визначає функцію UPDATE (ім'я_стовпця).

Для видалення тригера використовується команда:

DROP TRIGGER {ім'я_тригера} [,...n]

Наведемо приклади використання тригерів.

Приклад 14.1. Використання тригера для реалізації обмежень на значення. У доданій до таблиці Угода запису кількість проданого товару має бути не меншою, ніж його залишок з таблиці Склад.

Команда вставки запису в таблицю Угода може бути, наприклад, такою:

```
INSERT INTO Угода
VALUES (3,1,-299,'01/08/2002')
```

Створюваний тригер повинен відреагувати на її виконання наступним чином: необхідно скасувати команду, якщо в таблиціСклад величина залишку товару виявилася меншою за кількість товару, що продається, з введеним кодом (у прикладі код товару=3). У записі, що вставляється, кількість товару вказується зі знаком "+", якщо товар поставляється, і зі знаком "-", якщо він продається. Представлений тригер налаштований на обробку тільки одного запису, що додається.

```
CREATE TRIGGER Тригер_ins
ON Угода FOR INSERT
AS
IF @@ROWCOUNT=1
BEGIN
  IF NOT EXISTS(SELECT *
```

```
  FROM inserted
  WHERE -inserted.кількість<=ALL(SELECT
    Склад.
 FROM Склад,Угода
 WHERE Склад.КодТовара=
  Угода.КодТовара))
 BEGIN
  ROLLBACK TRAN
 PRINT
  'Скасування поставки: товару на складі немає'
 END
END
```

Приклад 14.1. Використання тригера для реалізації обмежень на значення.

Приклад 14.2. Використання тригера для збирання статистичних даних.

Створити тригер для обробки операції вставки запису в таблицю Угода, наприклад, такої команди:

```
INSERT INTO Угода
VALUES (3,1,200,'01/08/2002')
```

поставляється товар із кодом 3 від клієнта з кодом 1 у кількості 200 одиниць.

Під час продажу чи одержання товару необхідно відповідним чином змінити кількість його складського запасу. Якщо товару на складі немає, необхідно додати відповідний запис до таблиці Склад. Тригер обробляє лише один рядок, що додається.

```
ALTER TRIGGER Тригер_ins
ON Угода FOR INSERT
AS
DECLARE @x INT, @y INT
IF @@ROWCOUNT=1
--в таблицю Угода додається запис
```

```
--про постачання товару
BEGIN
--кількість проданого товару має бути не
--менше, ніж його залишок з таблиці Склад
IF NOT EXISTS(SELECT *
          FROM inserted
          WHERE -inserted.кількість<
  =ALL(SELECT Склад.Залишок)
        FROM Склад,Угода
        WHERE Склад.КодТовара=
           Угода.КодТовара))
    BEGIN
      ROLLBACK TRAN
       PRINT 'відкат товару немає'
     END
--якщо запису про поставлений товар ще немає,
--додається відповідний запис
--у таблицю Склад
  IF NOT EXISTS ( SELECT *
          FROM Склад C, inserted i
          WHERE C.КодТовара=i.КодТовара)
      INSERT INTO Склад (КодТовара, Залишок)
  ELSE
--якщо запис про товар вже був у таблиці
--Склад, то визначається код та кількість
--товару доданої до таблиці Угода запису
  BEGIN
    SELECT @y=i.КодТовара, @x=i.Кількість
    FROM Угода C, inserted i
    WHERE C.КодТовара=i.КодТовара
--і проводиться зміни кількості товару в
```

```
-таблиці Склад
        UPDATE Склад
        SET Залишок=залишок+@x
        WHERE КодТовара=@y
    END
END
```

Приклад 14.2. Використання тригера для збирання статистичних даних.

Приклад 14.3. Створити тригер для обробки операції видалення запису з таблиці Угода, наприклад, такої команди:

```
DELETE FROM Угода WHERE КодУгоди=4
```

Для товару, код якого вказано при видаленні запису, необхідно відкоригувати залишок на складі. Тригер обробляє лише один запис, що видаляється.

```
CREATE TRIGGER Тригер_del
ON Угода FOR DELETE
AS
IF @@ROWCOUNT=1 -- видалено один запис
BEGIN
  DECLARE @y INT, @x INT
--визначається код та кількість товару з
--видаленої з таблиці Склад запису
  SELECT @y=КодТовара, @x=Кількість
  FROM deleted
--у таблиці Склад коригується кількість
--товару
    UPDATE Склад
    SET Залишок=Залишок-@x
    WHERE КодТовара=@y
END
```

Приклад 14.3. Тригер для обробки операції видалення запису з таблиці

Приклад 14.4. Створити тригер для обробки операції зміни запису в таблиці Угода, наприклад, такою командою:

UPDATE Угода SET кількість = кількість-10

WHERE КодТовара=3

у всіх угодах з товаром, що має код, що дорівнює 3, зменшити кількість товару на 10 одиниць.

Вказана команда може призвести до зміни відразу кількох записів у таблиці Угода. Тому покажемо, як створити тригер, який обробляє не один запис. Для кожного зміненого запису необхідно для старого (до зміни) коду товару зменшити залишок товару на складі на величину старого (до зміни) кількості товару та для нового (після зміни) коду товару збільшити його залишок на складі на величину нового (після зміни) значення. Щоб обробити всі змінені записи, введемо курсори, в яких збережемо всі старі (з таблиці deleted) і нові значення (з таблиці inserted).

Питання та завдання до контролю знань студентів

1) Що таке тригер? Принцип його роботи.
2) Які існують режими зв'язку, створені за допомогою зовнішнього ключа?
3) У чому різниця між породженою таблицею та таблицею, що породжує?
4) Чи може бути використаний механізм тригерів для перевірки коректності введення. Наприклад: рядок, що містить e-mail, повинен обов'язково містити символ '@'?
5) Що означає поняття подія, що тригується?
6) За допомогою якої синтаксичної конструкції задається одноразове виконання тригера?
7) За допомогою якої синтаксичної конструкції можна звернутися до рядків, що видаляються?
8) За допомогою якої синтаксичної конструкції можна звернутися до рядків, що додаються?

9) За допомогою якої синтаксичної конструкції можна звернутися до рядків, що модифікуються?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 37

з навчальної дисципліни «Бази даних»

Тема Управління правами користувачів

Мета заняття Ознайомитись з командами адміністрування, а саме у правління користувачми та правами користувачів. Надання та скасування привілеїв користувачам.

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Управління користувачами серед MS SQL Server.

2 Адміністрація системи безпеки.

3 Керування доступом до даних. Визначення привілеїв у стандарті мови.

4 Надання привілеїв користувачам

5 Скасування наданих користувачам привілеїв

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Управління користувачами серед MS SQL Server

Стабільна система управління користувачами – обов'язкова умова безпеки даних, що зберігаються у будь-якій реляційній СУБД. У мові SQL немає єдиної стандартної команди, призначеної до створення користувачів бази даних – кожна реалізація робить це по-своєму. В одних реалізаціях ці спеціальні команди мають певну схожість, тоді як в інших їхній синтаксис має суттєві відмінності. Однак незалежно від конкретної реалізації всі основні засади однакові.

Розглянемо питання створення користувачів серед MS SQL Server.

Після проектування логічної структури бази даних, зв'язків між таблицями, обмежень цілісності та інших структур необхідно визначити коло користувачів, які матимуть доступ до бази даних.

У системі SQL-сервер організовано дворівневе налаштування обмеження доступу до даних. На першому рівні необхідно створити так званий обліковий запис користувача (login), що дозволяє йому підключитися до сервера, але не дає автоматичного доступу до баз даних. На другому рівні для кожної бази даних SQL-сервера на підставі облікового запису необхідно створити запис користувача. На основі прав, виданих користувачеві як користувачеві бази даних (user), його реєстраційне ім'я (login) отримує доступ до відповідної бази даних. У різних базах даних login одного і того ж користувача може мати однакові або різні імена користувача з різними правами доступу. Інакше висловлюючись, з допомогою облікового запису користувача здійснюється підключення до SQL-серверу, після чого визначаються його рівні доступу кожної бази даних окремо.

У системі SQL-сервер є додаткові об'єкти – ролі, які визначають рівень доступу до об'єктів SQL-сервера. Вони поділені на дві групи: призначені для облікових записів користувача сервера та використовуються для обмеження доступу до об'єктів бази даних.

Отже, на рівні сервера система безпеки оперує такими поняттями:

– аутентифікація;
– обліковий запис;
– вбудовані ролі сервера.

На рівні бази даних застосовуються такі поняття:

– користувач бази даних;

– фіксована роль бази даних;

– користувальницька роль бази даних.

Режими аутентифікації

SQL Server пропонує два режими автентифікації користувачів:

– режим аутентифікації засобами Windows NT/2000;

– змішаний режим аутентифікації (Windows NT Authentication and SQL Server Authentication).

2 Адміністрація системи безпеки

Для створення користувача в середовищі MS SQL Server слід зробити наступні кроки:

1) Створити обліковий запис користувача в базі даних, вказавши для нього пароль і прийняте за промовчанням ім'я бази даних (процедура sp_addlogin).

2) Додати цього користувача до всіх необхідних баз даних (процедура sp_adduser).

3) Надати йому у кожній базі даних відповідні привілеї (команда GRANT).

Створення нового облікового запису може бути проведене за допомогою системної процедури, що зберігається:

sp_addlogin [@login=] 'обліковий_запис' [, [@password=] 'пароль'] [, [@defdb=] 'база_даних_за промовчанням']

Після завершення автентифікації та отримання ідентифікатора облікового запису (login ID), користувач вважається зареєстрованим, і йому надається доступ до сервера. Для кожної бази даних, до об'єктів якої він має намір отримати доступ, обліковий запис користувача (login) асоціюється з користувачем (user) конкретної бази даних, що здійснюється за допомогою процедури:

sp_adduser [@loginame=] 'обліковий_запис' [, [@name_in_db=] 'ім'я_користувача'] [, [@grpname=] 'ім'я_ролі']

Відобразити обліковий запис Windows NT в ім'я користувача дозволяє процедура, що зберігається:

sp_grantdbaccess [@login=] 'обліковий_запис' [, [@name_in_db=]‘имя_користувача’]

Користувач, який створює об'єкт у базі даних (таблицю, процедуру, що зберігається, перегляд), стає його власником. Власник об'єкта (database object owner dbo) має всі права доступу до створеного ним об'єкта. Щоб користувач міг створити об'єкт, власник бази даних (dbo) має надати відповідні права. Повне ім'я створюваного об'єкта включає в себе ім'я користувача, що його створив.

Власник об'єкта не має спеціального пароля або особливих прав доступу. Він неявно має повний доступ, але має надати доступ іншим користувачам.

SQL Server дозволяє передавати права володіння від одного користувача іншому за допомогою процедури:

sp_changeobjectowner [@objname=] 'ім'я_об'єкта' [@newowner=] 'ім'я_власника'

Роль дозволяє об'єднати одну групу користувачів, виконують однакові функції.

У SQL Server реалізовано два види стандартних ролей: лише на рівні сервера і лише на рівні баз даних. При установці SQL Server створюються фіксовані ролі сервера (наприклад, sysadmin з правом виконання будь-яких функцій SQL-сервера) і фіксовані ролі бази даних (наприклад, db_owner з правом повного доступу до бази даних або db_accessadmin з правом додавання та видалення користувачів). Серед фіксованих ролей бази даних існує роль public, яка має спеціальне призначення, оскільки її членами є всі користувачі, які мають доступ до бази даних.

Можна включити будь-який обліковий запис SQL Server (login) або обліковий запис Windows NT будь-яку роль сервера.

Ролі бази даних дозволяють об'єднувати користувачів в одну адміністративну одиницю і працювати з нею як зі звичайним користувачем. Можна призначити права доступу до об'єктів бази даних для конкретної ролі, при цьому всі члени цієї ролі автоматично наділяються однаковими правами.

До ролі бази даних можна включити користувачів SQL Server, ролі SQL Server, користувачів Windows NT.

Різні дії щодо ролі здійснюються за допомогою спеціальних процедур:

1) Створення нової ролі:

sp_addrole [@rolename=] 'ім'я_ролі' [, [@ownername=] 'ім'я_власника']

2) Додавання користувача до ролі:

sp_addrolemember [@rolename=] 'ім'я_ролі', [@membername=] 'ім'я_користувача'

3) Видалення користувача з ролі:

sp_droprolemember [@rolename=] 'ім'я_ролі', [@membername=] 'ім'я_користувача'

4) Видалення ролі:

sp_droprole [@rolename=] 'ім'я_ролі'

3 Керування доступом до даних. Визначення привілеїв у стандарті мови

Кожна СУБД повинна підтримувати механізм, який гарантує, що доступ до бази даних зможуть отримати лише користувачі, які мають відповідний дозвіл. Мова SQL включає оператори GRANT та REVOKE, призначені для організації захисту таблиць у базі даних. Механізм захисту побудований на використанні ідентифікаторів користувачів, що надаються імправ володіння та привілеїв.

Ідентифікатор користувача називається звичайний ідентифікатор мови SQL, що використовується для позначення деякого користувача бази даних. Кожному користувачеві має бути призначений власний ідентифікатор, який надається адміністратором бази даних. З очевидних міркувань безпеки ідентифікатор користувача зазвичай зв'язується з деяким паролем. Кожен виконуваний СУБД SQL-оператор виконується від імені будь-якого

користувача. Ідентифікатор користувача визначає, на які об'єкти бази даних користувач може посилатися та які операції з цими об'єктами він має право виконувати.

Кожен створений у середовищі SQL об'єкт має свого власника, який спочатку є єдиною персоною, яка знає існування даного об'єкта і має право виконувати з ним будь-які операції.

Привілеями або правами називаються дії, які користувач має право виконувати щодо даної таблиці бази даних або подання. У стандарті SQL визначається наступний набір привілеїв:

– SELECT - право вибирати дані з таблиці;

– INSERT - право вставляти в таблицю нові рядки;

– UPDATE – право змінювати дані у таблиці;

– DELETE – право видаляти рядки з таблиці;

– REFERENCES – право посилатися на стовпці вказаної таблиці в описах вимог підтримки цілісності даних;

– USAGE – право використовувати домени, перевірки та набори символів.

Привілеї INSERT і UPDATE можуть обмежуватися лише окремими стовпцями таблиці, у разі користувачеві дозволяється модифікувати значення лише зазначених стовпців. Аналогічним чином привілей REFERENCES може поширюватися виключно на окремі стовпці таблиці, що дозволить використовувати їх імена у формулюваннях вимог захисту цілісності даних – наприклад, у пропозиціях CHECK та FOREIGN KEY, що входять до визначення інших таблиць, тоді як застосування для подібних цілей інших стовпців буде заборонено.

Коли користувач за допомогою оператора CREATE TABLE створює нову таблицю, він автоматично стає її власником і отримує по відношенню до неї повний набір привілеїв, яких інші користувачі не мають. Щоб забезпечити їм доступ, власник повинен явно надати необхідні права, для чого використовується оператор GRANT.

Створюючи подання за допомогою оператора CREATE VIEW, користувач автоматично стає власником цієї подання та отримує повний набір прав. Для створення уявлення користувачеві достатньо мати привілей SELECT для всіх таблиць, що входять до нього, і привілей REFERENCES для всіх стовпців, згадуваних у визначенні цього уявлення. Привілеї INSERT, UPDATE та DELETE щодо створеного уявлення користувач отримає лише в тому випадку, якщо має відповідні привілеї щодо всіх таблиць, що використовуються в поданні.

4 Надання привілеїв користувачам

Оператор GRANT застосовується для надання привілеїв щодо названих об'єктів бази даних зазначеним користувачам. Зазвичай його використовує власник таблиці для надання доступу до неї іншим користувачам. Оператор GRANT має такий формат:

<надання_привілеїв>::= GRANT {<привілей>[,...n] | ALL PRIVILEGES} ON ім'я_об'єкта TO {<ідентифікатор_користувача> [,...n]| PUBLIC} [WITH GRANT OPTION]

Параметр <привілей> є:

<привілей>::= {SELECT | DELETE | INSERT [(ім'я_стовпця[,...n])] | UPDATE [(ім'я_стовпця[,...n])]} | REFERENCES [(ім'я_стовпця[,...n])] | USAGE }

З міркувань спрощення оператора GRANT можна вказати ключове слово ALL PRIVILEGES, що дозволить надати зазначеному користувачеві всі існуючі привілеї без необхідності їх перерахування. Крім того, в цьому операторі може вказуватися ключове слово PUBLIC, що означає надання доступу вказаного типу не тільки всім існуючим користувачам, але також усім тим, хто буде визначений в базі даних згодом.

Параметр ім'я_об'єкта може використовуватися як ім'я таблиці бази даних, уявлення, домену, набору символів, перевірки.

Завдяки параметру WITH GRANT OPTION, зазначені в операторі GRANT користувачі мають право передавати всенадані їм щодо зазначеного об'єкта привілею іншим користувачам, які, у свою чергу, будуть наділені таким самим

правом передачі своїх повноважень. Якщо цей параметр не буде вказано, одержувач привілею не зможе передати свої права іншим користувачам. Таким чином, власник об'єкта може чітко контролювати, хто отримав праводоступу до об'єкта та які повноваження йому надано.

5 Скасування наданих користувачам привілеїв

У мові SQL для скасування привілеїв, наданих користувачам за допомогою оператора GRANT, використовується оператор REVOKE. За допомогою цього оператора можуть бути скасовані всі або деякі з привілеїв, отриманих вказаним користувачем раніше. Оператор REVOKE має такий формат:

<скасування_привілеїв>::= REVOKE[GRANT OPTION FOR] {<привілей>[,...n] | ALL PRIVILEGES} ON ім'я_об'єкта FROM {<ідентифікатор_користувача> [,...n]| PUBLIC} [RESTRICT | CASCADE]

Ключове слово ALL PRIVILEGES означає, що для зазначеного користувача скасовуються всі привілеї, надані йому раніше тим користувачем, який запровадив цей оператор. Необов'язкова фраза GRANT OPTION FOR дозволяє всім привілеїв, переданих у вихідному операторі GRANT фразою WITH GRANT OPTION, скасовувати можливість передачі незалежно від самих привілеїв.

Якщо в операторі вказано ключове слово RESTRICT, успішне виконання команди REVOKE можливе лише в тому випадку, коли перераховані в операторі привілеї не можуть спричинити появу у будь-яких інших користувачів так званих "залишених" привілеїв. За допомогою параметра CASCADE видаляються всі привілеї, які могли б залишитися в інших користувачів.

"Залишеними" є привілеї, що збереглися у користувача, якому вони свого часу були надані за допомогою GRANT OPTION.

Оскільки наявність привілею необхідна створення певних об'єктів, разом із її видаленням можна втратити права, з допомогою використання якого було утворено той чи інший об'єкт (подібні об'єкти називаються " покинутими ").

Якщо в результаті виконання оператора REVOKE можуть з'явитися кинуті об'єкти (наприклад, уявлення), право буде скасовано за умови, що в ньому не вказується ключове слово CASCADE. Якщо ключове слово CASCADE в операторі є, то для будь-яких кинутих об'єктів, що виникають при виконанні вихідного оператора REVOKE, будуть автоматично видані оператори DROP.

Привілеї, які були надані зазначеному користувачеві іншими користувачами, не можуть бути порушені оператором REVOKE. Отже, якщо інший користувач також надав даному користувачеві привілей, що видаляється, то праводоступу до відповідної таблиці у зазначеного користувача збережеться. Наприклад, нехай користувач A та користувач E мали право INSERT на таблицю Товар. Користувач A надав користувачеві B привілей INSERT для таблиці Товар, причому із зазначенням WITH GRANT OPTION (етап 1). Користувач B передав цей привілей користувачеві C (етап 2). Потім користувач C отримав її від користувача E (етап 3). Далі користувач C надав згаданий привілей користувачу D (етап 4). Коли користувач A скасовує привілей INSERT для користувача B, вона не може бути скасована і для користувача C, оскільки раніше він вже отримав її від користувача E. Якби користувач E не надав даного привілею користувачеві C, то видалення привілею користувача B мало б наслідком каскадне видалення привілеїв для користувачів C та D (див. таблицю 37.1).

Таблиця 37.1 – Надання прав користувачам

Користувач A	Користувач B	Користувач C	Користувач D	Користувач E
GRANT INSERT ON Товар TO B WITH GRANT OPTION	Отримання прав			
	GRANT INSERT ON Товар TO C WITH GRANT OPTION	Отримання прав від B. Оримання прав від E		GRANT INSERT ON Товар TO C WITH GRANT OPTION
		GRANT IN-SERT ON Товар TO D	Отримання прав	

REVOKE INSERT ON Товар TO B CASCADE	Відміна прав	Збереження прав	Збереження прав	Збереження прав

При підключенні до SQL Server всі можливі дії користувачів визначаються правами (привілеями, дозволами), виданими їх облікового запису, групою або роллю, в яких вони складаються.

Права можна розділити на три категорії:

– права на доступ до об'єктів;

– права виконання команд;

– неявні права.

Робота з даними та виконання збережених процедур вимагають наявності класу доступу, званого правами на доступ до об'єктів баз даних. Під об'єктами маються на увазі таблиці, стовпці таблиць, уявлення, процедури, що зберігаються.

Для різних об'єктів застосовуються різні набори прав доступу до них:

– SELECT, INSERT, UPDATE, DELETE, REFERENCES – для таблиці чи подання;

– SELECT, UPDATE – для конкретного стовпця таблиці чи подання;

– EXECUTE – для збережених процедур та функцій.

Право INSERT дозволяє вставляти нові рядки в таблицю або подання та видається лише на рівні таблиці або подання; воно може бути видано лише на рівні стовпця.

Право UPDATE видається або лише на рівні таблиці, що дозволяє змінювати у ній усі дані, або лише на рівні окремого стовпця, що дозволяє змінювати дані лише у межах.

Право DELETE дозволяє видаляти рядки з таблиці або подання, видається лише на рівні таблиці або подання, але не може бути видано на рівні стовпця.

Право SELECT дозволяє вибірку даних і може видаватися як на рівні таблиці, так і на рівні окремого стовпця.

Право REFERENCES надає можливість посилатися на вказаний об'єкт. Стосовно таблиць дозволяє створювати зовнішні ключі, що посилаються на первинний ключ або унікальний стовпець цієї таблиці.

Права на доступ до об’єктів.

Для керування дозволом користувача на доступ до об'єктів бази даних використовується команда:

```
<надання_привілеїв>::=
GRANT { ALL [ PRIVILEGES] | <привілей>
    [,...n]}
{ [( ім'я_стовпця [,...n])]
    ON { ім'я_таблиці |
        ім'я_перегляду} |
            ON {ім'я_таблиці |
        ім'я_перегляду }
            ([ім'я_стовпця
                [,...n])]
    | ON {ім'я_збереженої_процедури |
        ім'я_зовнішньої_процедури}}
TO { ім'я_користувача | ім'я_групи |
        имя_роли} [,...n]
[WITH GRANT OPTION ]
[AS {ім'я_групи | имя_роли }]
```

Параметр <привілей> є такою конструкцією:

```
<привілей>::=
{SELECT | DELETE | INSERT |
  UPDATE | EXECUTE | REFERENCES }
```

Параметр WITH GRANT OPTION допоможе користувачеві, якому ви надаєте права, призначити права доступу до об'єкта іншим користувачам. Його використання вимагає особливої обережності, оскільки власник втрачає контроль над наданням прав на доступ іншим користувачам. Найкраще

обмежити коло користувачів, які мають можливість керувати призначенням прав.

Необов'язковий параметр AS {ім'я_групи | имя_роли } дозволяє вказати участь користувача ролі, що забезпечує надання прав іншим користувачам.

Єдине право доступу, яке може бути надано для процедури, що зберігається, - право на її виконання (EXECUTE). Природно, крім цього власник процедури, що зберігається, може переглядати і змінювати її код.

Для функції можна видати право на її виконання, а також видати право REFERENCES, що забезпечить можливість зв'язування функції з об'єктами, на які вона посилається. Таке зв'язування дозволить заборонити внесення змін до структури об'єктів, які можуть призвести до порушення роботи функції.

Права виконання команд SQL.

Цей клас прав контролює можливість створення об'єктів у базі даних, самої бази даних та виконання процедури резервного копіювання. Можна використовувати таку команду для надання права на виконання команд SQL:

```
<надання_права_виконання>::=
GRANT {ALL | <команда>[,...n]}
TO {ім'я_користувача | ім'я_групи |
   имя_роли} [,...n]
```

Параметр <команда> є такою конструкцією:

```
<команда>::=
{CREATE DATABASE | CREATE TABLE |
   CREATE VIEW | CREATE DEFAULT |
   CREATE RULE | CREATE PROCEDURE
      | BACKUP DATABASE |
BACKUP LOG | ALL }
```

Таким чином, можна надати право на створення бази даних, таблиці, перегляду, замовчування, правила, процедури, що зберігається, резервної копії бази даних і журналу транзакцій або надати відразу всі вищезазначені права.

Неявні права

Виконання деяких дій не потребує явного дозволу і доступне за замовчуванням. Ці дії можуть виконуватися лише членами ролей сервера або власниками об'єктів у базі даних.

Неявні права не надаються користувачеві безпосередньо, він одержує їх лише за певних обставин. Наприклад, користувач може стати власником об'єкта бази даних, тільки якщо сам створить об'єкт або якщо хтось інший передасть право володіння своїм об'єктом. Таким чином власник об'єкта автоматично отримає права на виконання будь-яких дій з об'єктом, у тому числі на надання доступу до об'єкта іншим користувачам. Ці права ніде не вказуються, виконувати будь-які дії дозволяє лише факт володіння об'єктом.

Неявне відхилення доступу.

Неявне відхилення подібне до заборони доступу з тією відзнакою, що воно діє тільки на тому рівні, на якому визначено. Якщо користувачеві на певному рівні неявно відхилений доступ, він все ж таки може отримати його на іншому рівні ієрархії через членство в ролі, що має право перегляду. За промовчанням доступ користувача до даних неявно відхилено. Для неявного відхилення доступу до об'єктів бази даних використовується команда:

<неявне_відхилення_доступу>::=
REVOKE [GRANT OPTION FOR]
{ALL [PRIVILEGES] | | <привілей>
[,...n]}
{ [(ім'я_стовпця [,...n])] ON
{ ім'я_таблиці | ім'я_перегляду}
| ON {ім'я_таблиці |
ім'я_перегляду }
[ім'я_стовпця [,...n])]
| ON {ім'я_збереженої_процедури |
ім'я_зовнішньої_процедури}}
TO | FROM {ім'я_користувача |
ім'я_групи |
имя_роли}[,...n]

[CASCADE]

[AS {ім'я_групи | имя_роли }]

Для неявного відхилення дозволу виконання команд SQL використовується така команда:

<неявне відхилення дозволу>::=

REVOKE {ALL | <команда>[,...n]}

FROM {ім'я_користувача | ім'я_групи |

имя_роли}[,...n]

Сенс параметрів аналогічний параметрам команд GRANT та DENY. Параметр GRANT OPTION FOR використовується, коли необхідно відкликати право, надане параметром WITH GRANT OPTION команди GRANT. Користувач зберігає дозвіл на доступ до об'єкта, але втрачає можливість надавати цей дозвіл іншим користувачам.

Конфлікти доступу.

Дозволи, надані ролі чи групі, успадковуються їх членами. Хоча користувачеві може бути надано доступ через членство у ролі, роль іншого рівня може мати заборона на дію з об'єктом. У разі виникає конфлікт доступу.

При вирішенні конфліктів доступу SQL Server керується наступним принципом: дозвіл на надання доступу має найнижчий пріоритет, а на заборону доступу - найвищий. Це означає, що доступ до даних може бути отриманий лише явним наданням за відсутності заборони доступу на будь-якому іншому рівні ієрархії системи безпеки. Якщо доступ явно не надано, користувач зможе працювати з даними.

Приклад 37.1. Створити нову базу даних нового користувача для цієї бази даних, надавши йому всі права.

-- створення адміністратором нової

-- бази даних

CREATE DATABASE basa_user

-- створення нового користувача з

-- ім'ям UserA та паролем '123'

-- базою даних за умовчанням для

```
-- користувача UserA буде база
- з ім'ям basa_user.
sp_addlogin 'UserA', '123', 'basa_user'
-- перехід до бази даних basa_user
USE basa_user
-- додавання до поточної бази даних
-- (basa_user) користувача з ім'ям
-- userA
sp_adduser 'UserA'
-- надання користувачеві userA
-- у базі даних basa_user всіх прав
GRANT ALL TO UserA
```

Приклад 37.2. Використання ролей.

Створимо роль stud і включимо до цієї ролі двох користувачів user1 і user2:

```
sp_addrole 'stud'
sp_addrolemember 'stud','user1'
sp_addrolemember 'stud','user2'
```

Надамо права ролі stud та безпосередньо користувачеві user2:

```
GRANT SELECT, INSERT ON Товар TO stud
GRANT SELECT, INSERT ON Товар TO user2
```

Після виконання цих команд користувачі user1 та user2 можуть виконувати команди вибірки та додавання запису до таблиці Товар.

Припинимо право на виконання вставки в таблицю Товар для ролі stud:

```
REVOKE INSERT ON Товар TO stud
```

Після попередньої команди користувач user1 втрачає право вставки запису, а user2 зберігає це право, оскільки право вставки надано йому явно.

Виконаємо команду:

```
DENY INSERT ON Товар TO stud.
```

Після виконання цієї команди обидва користувачі позбавляються права вставки таблицю Товар.

Питання та завдання до контролю знань студентів

1) Як створити користувача та визначити його права?
2) Як видалити користувачів з бази даних?
3) Як змінии пароль користувачу в базі даних?
4) Як змінии роль користувача?
5) Як видалити роль користувача?
6) Як надаи привілеї користувачу або ролі?
7) Як скасуваи прівелеї користувачу?
8) Які привілеї можна надавати користувачам?

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 38

з навчальної дисципліни «Бази даних»

Тема Адміністрування бази даних

Мета заняття Огляд команд MySQL, що дозволяють отримати інформацію про сервер та бази даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

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 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BV, 2006. – 384 с.: іл..

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 копіямysqld 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 (Засобу адміністрування), Services (Сервисы). 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 набор 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;

і

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 привілею, 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(ll) NOT NULL auto_increment,

'name' varchar(30) default NULL,

PRIMARY KEY ('departmenti')

) TYPE-InnoDB CHARSET-latinl

(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.)

З 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 використає синтаксис

set змінна=значення;

Наприклад, можна ввести

set sql_safe_updates=l;

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_nроцесу;

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.

6 Вміст файлів журналу

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. Єдинийз журналів, запис у який за замовчуванням включена, розміщується у вашому каталозі даних. Файл називається hostname.err в Linuxй mysql.err в Windows. Можна призначити йому будь-яке інше розміщення, використовуючи опцію log-errог=ім'я_файлу у файлі конфігураціїmy. ini (або my.cnf).

Журнал реєстрації запитів. Реєструє всі запити, виконувані системою. Можна включити запис цього журналу й указати його розташуванняза допомогою опції 1og=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.O. Можна включити записцього журналу й указати його розташування за допомогою опції log-bin=ім'я_файлу.

Журнал реєстрації повільних запитів. Реєструє всі запити, часвиконання яких виявився більше, ніж значення, збережене змінною long_query_time. Можна включити запис цього журналу й указати йогорозташування за допомогою опції 1og-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 системаMySQL 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 idl,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.

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.

Лекція № 39

з навчальної дисципліни «Бази даних»

Тема Поняття розподілених баз даних. Переваги та недоліки властиві розподіленим СУБД.

Мета заняття Огляд команд MySQL, що дозволяють отримати інформацію про сервер та бази даних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН: мультимедійний комплект

Час – 2 години

План проведення лекції

1 Основні означення.

2 Властивості розподілених баз даних.

3 Логічна архітектура розподілених баз даних.

4 Архітектура програмно-технічних засобів розподілених СУБД.

5 Різновиди архітектури.

6 Розподілене зберігання даних.

7 Обробка розподілених транзакцій

Література

1 Веллинг, Люк. MySQL Учебное пособие [Текст]: Пер. с англ. / Люк Веллинг, Лора Томпсон – М.: Издательский дом «Вильямс», 2005. – 304 с.: ил.

2 Фиайли, К. SQL Руководство по изучению языка [Текст]: Пер. с англ. / Крис Фиайли – М.: ДМК Пресс; СПб.: Питер, 2004, – 464с.: ил.

3 Пасічник, В,В. Організація баз даних та знань [Текст] / В.В. Пасічник, В.А. Резніченко – К.: Видавнича група BV, 2006. – 384 с.: іл..

Навчальні матеріали лекції

1 Основні означення

Розподілена база даних (РБД) – це множина логічно взаємозалежних баз даних, розподілених у комп'ютерній мережі.

Розподілена система управління базами даних (РСУБД) – це програмне забезпечення, яке керує РБД і надає такі механізми доступу до них, що їх застосування дає користувачу можливість працювати з РБД як з однією цілісною базою даних.

Розподілена система баз даних (РСБД) – це РБД разом із РСУБД.

Не слід плутати РСБД з централізованою базою даних, що використовується в мережі, як наведено на рисунку 39.1. У цьому випадку база даних розташована на одному з комп'ютерів, а всі інші мають доступ до неї через комунікаційну мережу. Не є розподіленою також база даних, що працює в середовищі багатопроцесорних комп'ютерів. У цьому випадку ми маємо справу з паралельною БД.

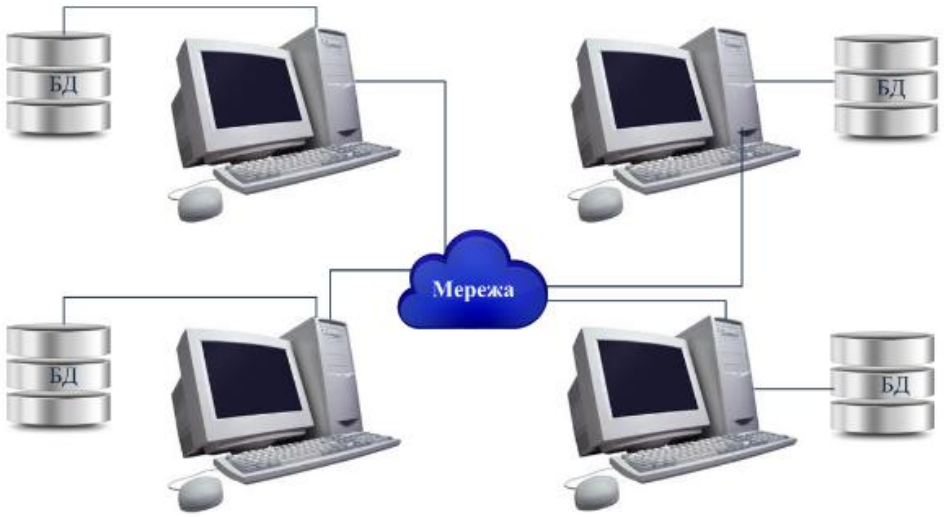


Рисунок 39.1 – Централізована база даних у комп'ютерній мережі

Архітектура розподіленої СУБД наведена на рисунку 39.2. Кожний з вузлів мережі містить свою базу даних, однак вони розглядаються як логічно єдина база, а не як сукупність розкиданих у мережі файлів. Усі дані є логічно взаємозалежними. Розподілена СУБД — це повноцінна СУБД, що виконує всі необхідні функції з керування даними.

Залежно від типу програмного забезпечення розрізняють два типи РСУБД:

– однорідні;

– неоднорідні.

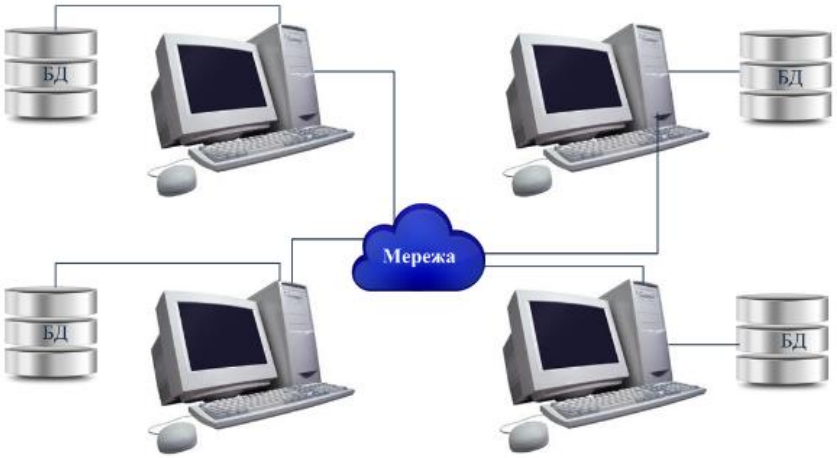


Рисунок 9.2 - Розподілена база даних

В однорідних РСУБД передбачається, як мінімум, що на всіх вузлах використовуються однотипні СУБД (наприклад, реляційні), які мають схожі функціональні можливості. Як максимум, припускається, що на всіх вузлах використовуються однакові технічні засоби, тобто однакові типи комп'ютерів і програмного забезпечення. Це стосується операційних систем, програмного забезпечення СУБД та моделей даних, що підтримуються.

У неоднорідних РСУБД вузли базуються на різних програмно-технічних платформах, які можуть містити різні типи СУБД. Окрім того, такі СУБД можуть підтримувати різні моделі даних. У цьому випадку ускладнюється вирішення проблеми їхньої взаємодії.

Однією з важливих проблем РСУБД є досягнення логічної незалежності даних від місця зберігання, тобто прозорість доступу до даних. Це означає, що користувач повинен мати можливість сприймати всі необхідні йому дані як єдине ціле, не зважаючи на те, в який спосіб вони розподілені у мережі.

2 Властивості розподілених баз даних

Вперше завдання про дослідження основ і принципів створення і функціонування розподілених інформаційних систем було поставлене відомим фахівцем в області баз даних Д. Дейтом.

Локальна автономія (local autonomy). Ця якість означає, що управління даними на кожному з вузлів розподіленої системи виконується локально. База даних, розташована на одному з вузлів, є невід'ємним компонентом розподіленої системи. Будучи фрагментом загального простору даних, вона в той же час функціонує як повноцінна локальна база даних, а управління нею здійснюється локально, незалежно від інших вузлів системи.

Незалежність вузлів (no reliance on central site). Всі вузли рівноправні і незалежні, а розташовані на них БД є рівноправними постачальниками даних в загальний простір даних. База даних на кожному з вузлів самодостатньою — вона включає повний власний словник даних і повністю захищена від несанкціонованого доступу.

Безперервні операції (continuous operation). Це можливість безперервного доступу до даних в рамках розподіленої БД незалежно від їх розташування і незалежно від операцій, що виконуються на локальних вузлах.

Прозорість розташування (location independence). Користувач, що звертається до БД, нічого не повинен знати про реальне, фізичне розміщення даних у вузлах інформаційної системи.

Прозора фрагментація (fragmentation independence). Можливість розподіленого (тобто на різних вузлах) розміщення даних, логічно поєднаних в єдине ціле. Існує фрагментація двох типів: горизонтальна і вертикальна. Перша означає, що рядки таблиці зберігаються на різних вузлах. Друга означає розподіл стовпців логічної таблиці по декількох вузлах.

Прозоре тиражування (replication independence). Тиражування даних - це асинхронний процес перенесення змін об'єктів вихідної бази даних в бази, розташовані на інших вузлах розподіленої системи

Обробка розподілених запитів (distributed query processing). Можливість виконання операцій вибірки даних з розподіленої БД, за допомогою запитів, сформульованих на мові SQL

Обробка розподілених транзакцій (distributed transaction processing). Можливість виконання операцій оновлення розподіленої бази даних, які не порушують цілісність і узгодженість даних. Ця мета досягається вживанням двофазного протоколу фіксації транзакцій.

Незалежність від устаткування (hardware independence). Ця властивість означає, що як вузли розподіленої системи можуть виступати ПК будь-яких моделей і виробників

Незалежність від операційних систем (operationg system independence). Ця якість витікає з попереднього і означає різноманіття операційних систем, керуючих вузлами розподіленої системи

Прозорість мережі (network independence). Доступ до будь-яких баз даних відбувається по мережі. Спектр підтримуваних конкретною СУБД мережевих протоколів не має бути обмеженням системи, заснованої на розподіленій БД

Незалежність від баз даних (database independence). Ця якість означає, що в розподіленій системі можуть працювати СУБД різних виробників, і можливі операції пошуку і оновлення в базах даних різних моделей і форматів.

3 Логічна архітектура розподілених баз даних

Логічна архітектура розподілених баз даних – це архітектура логічно взаємозалежних даних. Графічне зображення логічної архітектури розподілених баз даних наведене на рисунку 39.3. Особливість архітектури РБД полягає в тому, що виникає ще один рівень, глобальний концептуальний, завданням якого (як і в моделі ANSI/SPARC) є зображення концептуальної моделі предметної області в цілому. На цьому рівні описується характер розподілу даних.

На локальному концептуальному рівні здійснюється локальний опис ПО. Тобто схема цього рівня містить опис тільки тієї частини предметної області, що є специфічною для конкретного вузла розподіленої ПО; дані інших вузлів розподіленої ПО в схемі не вказуються. Відображення «глобальний-концептуальний/локальний-концептуальний» дають змогу зобразити глобальну

концептуальну схему у вигляді сукупності локальних концептуальних схем, і навпаки.

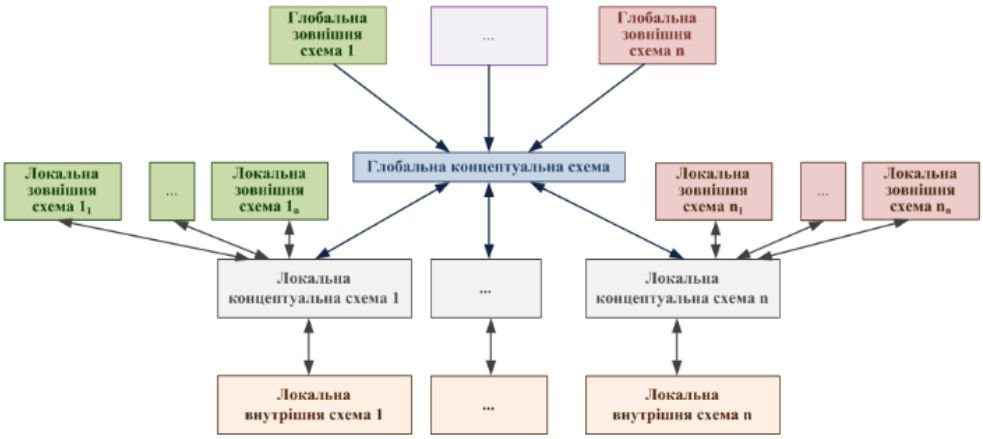


Рисунок 9.3 – Логічна архітектура розподіленої СУБД

Глобальна зовнішня схема зображує дані, необхідні користувачам і додаткам, у бажаному для них вигляді, незалежно від способу розподілу даних за вузлами. Це завдання вирішується завдяки тому, що глобальні зовнішні схеми базуються на глобальній концептуальній схемі.

Локальні зовнішні схеми базуються на локальних концептуальних схемах, відтак вони можуть посилатися лише на ті дані, що розташовані на відповідному вузлі розподіленої ПО.

Локальні внутрішні схеми – це схеми зберігання даних на конкретних вузлах розподіленої ПО. Вони пов’язані з відповідними локальними концептуальними схемами.

4 Архітектура програмно-технічних засобів розподілених СУБД

Властивості архітектури.

Архітектура програмно-технічного комплексу розподілених СУБД має три головні характеристики:

розподіленість;

неоднорідність;

автономність.

Розподіленість. Спосіб розподілу компонентів системи баз даних за комп'ютерами мережі визначається тим, чи є в мережі єдиний комп'ютер з повноваженнями розподіленої СУБД, або ж їх кілька. Якщо таких комп'ютерів кілька, то чи є між ними взаємодія тощо.

Неоднорідність. Важливою характеристикою є міра однорідності програмно-технічних засобів СУБД. Ідеться про технічне забезпечення (типи комп'ютерів), комунікаційні засоби зв'язку, операційні системи і типи баз даних (моделі даних, мови запитів, алгоритми керування транзакціями тощо).

Автономність. Характеризує, наскільки самостійно компоненти СУБД можуть виконувати свої функції. До питань автономності належать:

– автономність проектних рішень (наскільки самостійно компоненти СУБД можуть розв'язувати задачі, які були спроектовані для них); автономність вирішенням комунікаційних проблем (наскільки самостійно компоненти СУБД можуть встановлювати зв'язки з іншими компонентами);

– автономність обчислень (наскільки самостійно компоненти СУБД можуть приймати рішення щодо виконання локальних операцій).

5 Різновиди архітектури

Основними різновидами архітектури програмно-технічних засобів розподіленої СКБД є:

– клієнт-серверна архітектура;
– архітектура з багатьма незалежними серверами;
– архітектура із взаємодіючими серверами;
– архітектура однорангової мережі.

Клієнт-серверна архітектура. Така архітектура передбачає наявність єдиного комп'ютера-сервера і багатьох комп'ютерів-клієнтів, що взаємодіють між собою через канали зв'язку. На сервері розташована СКБД та інтегрована (централізована) база даних. Ніякого розподілу баз даних за вузлами мережі немає. На клієнтських комп'ютерах виконуються додатки, які працюють із серверною базою даних, а також розміщене програмне забезпечення для зв'язку

з віддаленою СКБД. Як клієнти, так і сервер оснащені комунікаційним програмним забезпеченням.

Архітектура з багатьма незалежними серверами. Ця архітектура передбачає існування багатьох серверів, що мають доступ до своїх локальних баз даних. У такому випадку на комп'ютерах клієнтів має зберігатись інформація стосовно того, які дані на яких серверах розташовані, а також має бути розміщене програмне забезпечення, що дає змогу декомпозувати запити для їхнього виконання на різних серверах і потім об'єднати результати.

Архітектура із взаємодіючими серверами. У цьому випадку передбачається, що кожний із серверів містить повну інформацію стосовно того, які дані на яких серверах зберігаються, а також здатний обробляти розподілені запити. У разі потреби один сервер може звернутися до іншого для одержання необхідних даних. Кожен клієнт має свій сервер, до якого звертається для виконання операцій над розподіленою базою даних.

Архітектура однорангової мережі. Усі комп'ютери мережі є серверами. На кожному комп'ютері розміщено розподілену СКБД і базу даних, з кожного комп'ютера можна надіслати до іншого запит на отримання необхідних даних.

6 Розподілене зберігання даних

У розподіленій СКБД дані розподіляються за вузлами мережі. Існують два основні механізми розподіленого зберігання даних:

– фрагментація;

– реплікація.

Фрагментація.

Суть фрагментації полягає в тому, щоб поділити логічну базу даних на фрагменти з метою зберігання кожного фрагмента на певному вузлі мережі. Одиницями фрагментації можуть бути відношення та складені відношення. У випадку, коли одиницею фрагментації є відношення, вирішується проблема, яке відношення в якій базі даних має зберігатися. За іншого підходу допускається, що будь-яке відношення може бути зображене у вигляді сукупності фрагментів, що розподіляються за різними базами даних.

Фрагментацію, що здійснюється розподілом відношень за базами даних, теоретично здійснити нескладно, тому розглянемо проблему фрагментації власне відношень.

Фрагментація відношень. Завдання фрагментації відношень формулюється в такий спосіб. Нехай задане відношення R. Його потрібно зобразити у вигляді сукупності відношень R1, ..., Rn так, щоб ця сукупність відповідала критеріям ефективності (за часом доступу, пам'яттю, завантаженістю комп'ютерів тощо).

Фрагментація є коректною, якщо вона повна, не містить перетинів і може бути реконструйована. Пояснимо ці терміни.

Декомпозиція відношення R на фрагменти R_1, R_2, ... R_n є повною тоді й лише тоді, коли кожен елемент даних з R належить якомусь із відношень R_i.

Декомпозиція відношення R на фрагменти $R1$, R_2, ... R_n може бути реконструйована, якщо існує такий реляційний вираз $\varphi(R_1, R2, \dots Rn)$, що $R=\varphi(R_1, R_2, \dots R_n)$.

Декомпозиція відношення R на фрагменти R_1, R_2, ... R_n не містіть перетинів, якщо будь-який елемент даних з R міститься не більш ніж в одному фрагменті.

Є три типи фрагментації відношень:

– горизонтальна;

– вертикальна;

– змішана.

Горизонтальна фрагментація.

Горизонтальна фрагментація полягає в розподілі кортежів відношення за фрагментами. Формально горизонтальну фрагментацію можна визначити в такий спосіб. Нехай задане відношення R і на ньому визначений предикат F_i. Тоді горизонтальний фрагмент Ri відношення визначається так:

$$Ri=\sigma F_i(R)$$

Тобто горизонтальний фрагмент R_i - це множина кортежів R, що задовольняють умову F_i.

Переваги горизонтальної фрагментації:

– дає змогу паралельно обробляти фрагменти відношення;

– дає можливість поділяти відношення так, щоб кортежі розташовувалися там, де вони будуть використовуватися найчастіше.

Вертикальна фрагментація.

Суть вертикальної фрагментації полягає в тому, що відношення поділяється на дві чи більше проекцій, тобто схема відношення поділяється на певну множину підсхем.

Для відновлення вихідного відношення з фрагментів необхідно, щоб усі підсхеми містили первинний ключ. Існує інший підхід, коли під час поділу схеми до кожного фрагмента автоматично додається поле з ідентифікатором кортежу. Значення цього ідентифікатора зазвичай встановлюються системою автоматично.

Переваги вертикальної фрагментації:

– дає змогу поділяти кортежі відношення так, щоб їхні частини розташовувалися там, де вони використовуватимуться найчастіше;

– дає змогу проводити паралельну обробку відношень;

– наявність ідентифікатора кортежу дає можливість здійснювати ефективне з’єднання вертикальних фрагментів.

Змішана фрагментація. Змішана фрагментація передбачає послідовне застосування вертикальної і горизонтальної фрагментацій.

Розподіл даних за вузлами мережі.

Після отримання усіх необхідних фрагментів відношень постає проблема розподілу цих фрагментів за вузлами мережі. Єдиних рекомендацій стосовно того, як це робити, немає. Потрібно знайти оптимальний розподіл фрагментів F за вузлами мережі S за умови, що відомий розподіл додатків Q за вузлами мережі.

Визначаючи оптимальність, слід враховувати різні параметри, зокрема:

– вартість передавання, зберігання й обробки даних;

– часові характеристики;

– продуктивність;

– обмеження (наприклад, ємнісні характеристики вузлів мережі).

Для того щоб розподілити фрагменти за вузлами мережі, необхідна додаткова інформація, яка стосується:

– бази даних та розмірів фрагментів;

– додатків (місць їхнього розташування, частоти використання тих чи інших фрагментів для вибирання даних, частоти використання фрагментів для відновлення даних);

– вузлів мережі (вартості зберігання й обробки даних у вузлах);

– мережі (вартості та часових характеристик передавання даних між двома вузлами).

Реплікація.

Реплікація — це механізм розподілу даних за вузлами, що дозволяє зберігати копії тих самих даних на різних вузлах мережі з метою прискорення пошуку і підвищення стійкості до відмов. Відношення чи фрагмент є реплікованим, якщо його копії зберігаються на двох або більше вузлах (копії ще називають репліками). За повної реплікації відношення його копії зберігаються на всіх вузлах мережі. Допускається ситуація, коли вся база даних зберігається на всіх вузлах мережі — це називається поєною реплікацією бази даних.

Переваги реплікації:

– доступність (у разі перебою в роботі вузла, що містить відношення H, його доступність на інших вузлах зберігається);

– паралелізм (виконання запитів до відношення R може бути розпаралелено за всіма репліками відношення);

– зниження вартості передавання даних (відношення R доступне локально в усіх вузлах, де є його репліки).

Недоліки реплікації:

– підвищується вартість зберігання, створення і відновлення даних; підвищуються вимоги до ресурсів;

– ускладнюється підтримання цілісності даних, наприклад одночасне відновлення різних реплік одного й того ж відношення

Механізми реплікації

Для реалізації реплікації використовуються три сервери: видавець, дистриб’ютор і передплатник.

Видавцем називають сервер, що надає розміщені на ньому дані для копіювання на інші сервери. Окрім створення копії даних, видавець відстежує внесені до його бази даних зміни і готує нову копію.

Дистриб’ютором називається сервер, що підтримує розподілену базу даних. Він виконує роль посередника, копіює всі публікації, підготовлені видавцем, і пересилає їх передплатникам. Дистриб’ютором може бути виділений сервер або сервер, сконфігурований як видавець чи передплатник. Конкретні функції, що їх виконує дистриб’ютор, залежать від методів реплікації.

Передплатником називається сервер, що отримує копії даних, надані видавцем. Механізми зміни даних передплатником відрізняються від механізмів зміни даних видавцем.

Є два методи відновлення даних передплатників:

– реплікація за запитом. Передплатник періодично звертається до дистриб’ютора із запитом про зміни, що відбулися з моменту останнього з’єднання.

– примусова реплікація. Дистриб’ютор сам встановлює з’єднання з передплатником і пересилає йому необхідні дані.

Залежно від методу реплікації, передплатники можуть чи не можуть вносити зміни в репліковані дані. У найпростішому випадку змінювати дані може тільки видавець, у складніших моделях реплікації - передплатники і видавці. Змінені дані, отримані від усіх передплатників, синхронізуються і поєднуються з даними видавця, а потім розсилаються передплатникам.

Моделі реплікації.

Є такі моделі реплікації:

– реплікація моментальних знімків;

– реплікація транзакцій.

Реплікація моментальних знімків є найпростішою моделлю реплікації. Моментальний знімок це повна копія даних, обраних для реплікації; вона розсилається передплатникам.

Під час реплікації транзакцій використовується журнал транзакцій бази даних. Обрані транзакції копіюються в базу даних дистриб'ютора зі збереженням інформації про послідовність їхнього виконання, потім розсилаються серверам- передплатникам і виконуються на них у тому ж порядку, в якому виконувалися на сервері-видавці.

Цей механізм зменшує завантаження мережі, його рекомендується використовувати у великих базах даних з невеликою кількістю змін.

Топологія реплікацій.

Топологія реплікацій описує характер взаємозв'язків між учасниками реплікації:

– реплікація «один-до-багатьох» передбачає наявність одного видавця і кількох передплатників;

– реплікація «багато-до-одного» має місце, коли дані від кількох видавців пересилаються одному передплатнику;

– реплікація «багато-до-багатьох» означає, що дані від кількох видавців пересилаються кільком передплатникам.

7 Обробка розподілених транзакцій

Транзакція — набір команд, що виконується як єдине ціле. У транзакції або всі команди будуть виконані, або жодна з них не виконається. Якщо хоча б одна з команд транзакції не може бути виконана, здійснюється відкочування (відновлюється стан системи, в якому вона перебувала до початку виконання транзакції).

Транзакції мають задовольняти вимоги ACID (Atomicity, Consistency, Isolation, Durability — атомарність, несуперечність, ізольованість, довговічність), що гарантують правильність і надійність роботи системи.

Атомарність передбачає таке:

– виконуються всі операції транзакції або жодна з них не виконується;

– якщо виконання транзакції було перерване, то всі зроблені транзакцією зміни мають бути скасовані

Дії, спрямовані на підтримку атомарності транзакції під час її аварійного завершення у зв’язку з помилками введення/виведення, перевантаженнями системи чи блокуваннями, називаються відновленням транзакції.

Дії, спрямовані на забезпечення атомарності під час виходу з ладу системи, називаються відновленням у разі виникнення перебоїв.

Несуперечність означає, що транзакція, яка працює з несуперечною базою даних, після завершення роботи залишає її також у несуперечному стані. Транзакція не повинна порушувати цілісності бази даних

Для вирішення проблем одночасного доступу інститут ANSI розробив спеціальний стандарт, який визначає чотири рівні блокування (кожний вищий рівень передбачає виконання умов усіх нижчих рівнів)

– Рівень 0. Заборона «забруднення» даних. На цьому рівні вимагається, щоб змінювати дані могла лише одна транзакція. Якщо іншій транзакції необхідно змінити ці ж дані, то вона має очікувати завершення першої транзакції.

– Рівень 1. Заборона некоректного зчитування. Якщо певна транзакція розпочала змінювати дані, то жодна інша транзакція не зможе зчитати ці дані доти, доки перша не завершиться.

– Рівень 2. Заборона неповторюваного зчитування. Якщо певна транзакція зчитує дані, то жодна інша транзакція не зможе їх змінити. Отже, під час повторного зчитування дані перебуватимуть у початковому стані.

– Рівень 3. Заборона «фантомів». Якщо транзакція звертається до даних, то жодна інша транзакція не зможе додати чи видалити рядки, що можуть бути зчитані під час виконання транзакції. Реалізація цього рівня блокування виконується блокуванням діапазону ключів. Це блокування накладається не на конкретні рядки таблиці, а на рядки, що відповідають певній логічній умові.

Властивість ізольованості означає, що на роботу транзакції не мають впливати інші транзакції. Транзакція «бачить» дані в тому стані, в якому вони перебували до початку роботи іншої транзакції, або в тому стані, в якому вони

перебувають після її завершення. Одна транзакція не може переглядати проміжні стани даних, що використовуються іншими транзакціями. Якщо транзакція зчитує кілька разів ті самі дані, то вона повинна отримувати їх щоразу в тому стані, в якому вони були під час першого зчитування. Ще одним аспектом ізольованості є неможливість роботи з неповними результатами — незавершена транзакція не може передавати свої результати іншим транзакціям до підтвердження свого успішного завершення.

Після того, як було підтверджено успішне завершення роботи транзакції (Commit), система має гарантувати, що її результати не будуть втрачені, незважаючи на можливі перебої. Це й називається довговічністю. Для забезпечення довговічності застосовуються механізми відновлення бази даних.

Питання та завдання до контролю знань студентів

1) Розподілена база даних – це…
   а) база даних, яка дозволяє працювати в мережі –
   б) база даних, яка зберігається на різних комп’ютерах –
   в) база даних, яка дозволяє з’єднати декілька комп’ютерів –
   г) вірної відповіді нема.
2) Найбільш розповсюдженими на практиці є –
   а) розподілені бази даних
   б) ієрархічні бази даних
   в) мережні бази даних
   г) реляційні бази даних
3) База даних, яка зберігається на декількох комп’ютерах, називається…
   а) локальною
   б) розподіленою
   в) загальнодоступною
   г) спеціалізованою

Розробив: Ланська С.С.
Розглянуто та схвалено
на засіданні циклової комісії
програмної інженерії
Протокол № 8 від 13.04.2023 р.
Голова комісії _____ Ланська С.С.