МІНІСТЕРСТВО ОСВІТИ І НАУКИ ДЕРЖАВНИЙ ЗАКЛАД „ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА”

Укладачі: Г.А. Могильний, В.Ю. Донченко

БАЗИ ДАНИХ ТА ІНФОРМАЦІЙНІ СИСТЕМИ

Матеріали до самостійної роботи з освітнього компонента (модуль 1) для здобувачів освіти спеціальності 122 – „Комп'ютерні науки”

Матеріали до самостійної роботи створено на засадах різноманітних публікацій з відкритих джерел:
https://studfile.net/preview/7406711/
https://virt.ldubgd.edu.ua/mod/resource/view.php?id=13329
https://virt.ldubgd.edu.ua/mod/resource/view.php?id=13327

Укладачі висловлюють свою вдячність всім авторам відкритих публікацій

Полтава
ДЗ „ЛНУ імені Тараса Шевченка”
2024

ЗМІСТ

ЛЕКЦІЯ. ВСТУП ДО БАЗ ДАНИХ

План:

1. Технології баз даних.
2. Компоненти банків даних.
3. Компоненти систем баз даних.

1. Технології баз даних

Виникнення технологій баз даних припадає на початок 60-х років. Їх швидкому розвитку сприяли потреби в обробці інформації, досягнення в суміжних областях інформаційних технологій таких, як операційні системи, мови програмування, технічне забезпечення. Спочатку зароджувалися певні ідеї щодо управління ресурсами даних, формувалися основи методології побудови систем баз даних. Із самого початку було зрозуміло, що цей напрям має самостійне значення і буде відігравати одну з ключових ролей у побудові інформаційних систем різного призначення.

Інформаційні задачі на відміну від обчислювальних мають такі особливості:

−збереження даних складної структури;

−відносно прості алгоритми обробки;

−великі обсяги оброблюваної інформації.

Інформаційна система виконує функції збирання, зберігання, розповсюдження і обробки інформації. Під *інформацією* розуміють будь-які відомості про будь-яку подію, сутність, процес і т.ін., які є об'єктом певних операцій: передачі, перетворення, зберігання або використання. *Дані* можна визначити, як інформацію зафіксовану у певній формі, яка придатна для

подальшої обробки, зберігання і передачі (рис. 1.1). *Предметна область* – область застосування конкретної бази даних.

Інформації відповідає *інфологічне представлення*, яке розглядає питання пов'язані зі змістом інформації, незалежно від представлення її в пам'яті комп'ютера. *Даталогічне представлення* розглядає питання представлення даних в пам'яті комп'ютера.

Функції управління інформацією в інформаційних системах виконують системи управління базами даних.

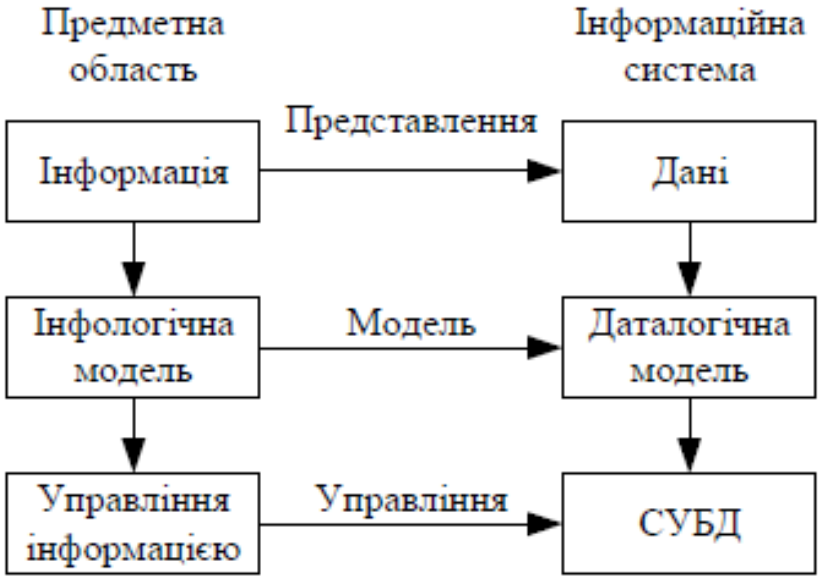


Рис. 1.1. Зв'язок між інформацією і даними

Спочатку для збереження даних застосовувалися *файлові системи* (рис. 1.2). Для цих систем були характерні такі особливості:

– структура запису файла даних була відома тільки прикладній програмі, яка з ним працювала, а система управління файлами її не знала; кожна програма, яка працювала з файлом даних, повинна була мати у себе структуру даних, яка відповідала цьому файлу (така ситуація характеризувалась, як *залежність програм від даних*);

– система управління файлами повинна була забезпечити авторизацію доступу до файлів для різних користувачів, але були відсутні централізовані методи управління доступом до інформації;

– для організації паралельної роботи користувачів з даними необхідне узгоджене управління, а система управління файлами при цьому працювала надто повільно.

Для подолання цих недоліків практично паралельно почалися роботи над *ієрархічними* і *мережними* базами даних (БД) і над відповідними системами управління цими БД СУБД. В основі цих БД лежали відповідні моделі даних: ієрархічні і мережні (рис. 1.2).

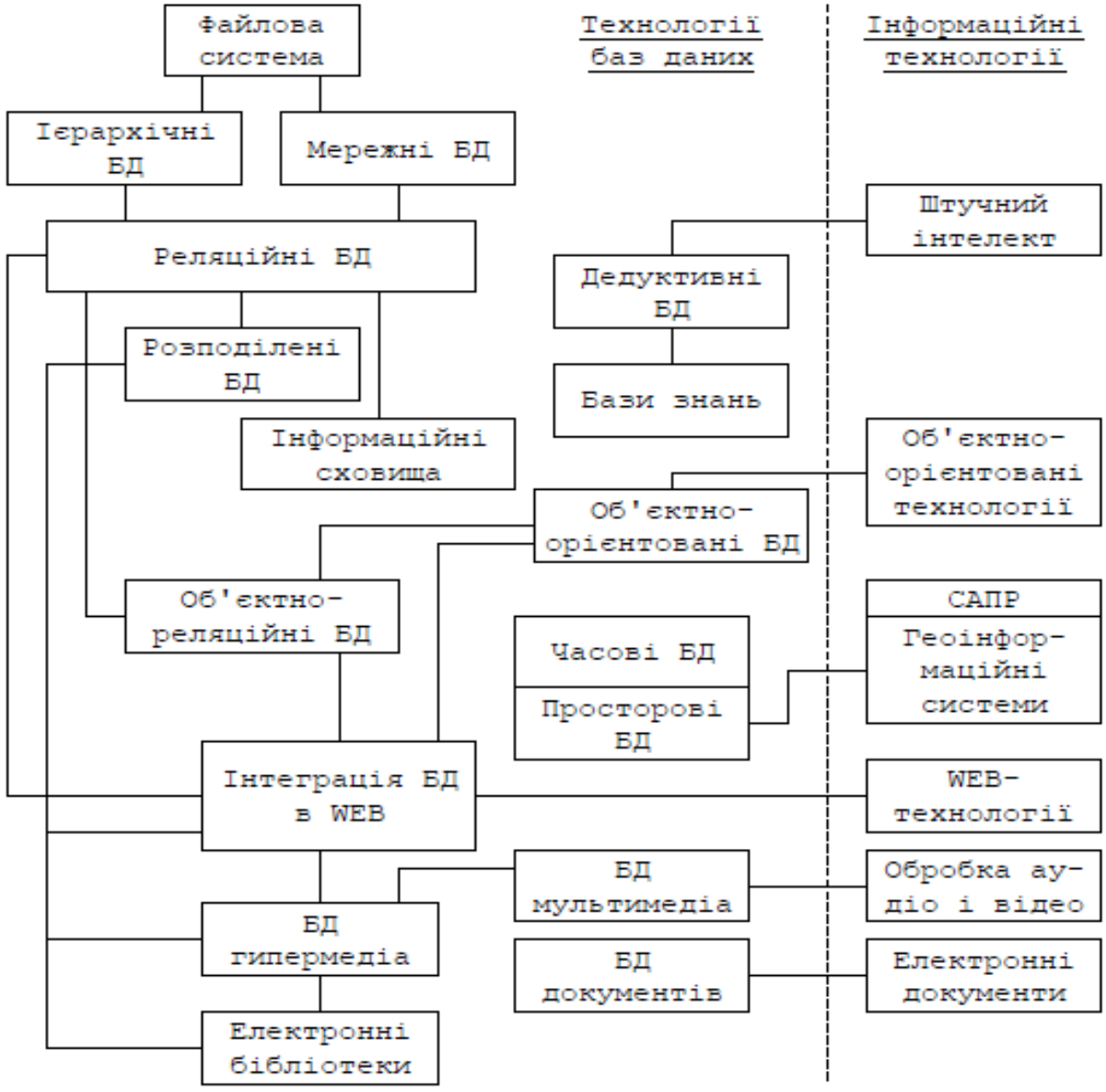


Рис. 1.2. Етапи розвитку баз даних

Ці моделі належать до теоретико-графових моделей. До переваг цих БД можна віднести ефективне використання пам'яті комп'ютера, швидке виконання основних операцій над даними.

Недоліками цих БД є таке:

– труднощі при обробці інформації з достатньо складними зв'язками;

– складність розуміння змісту звичайним користувачем, залежність від фізичної реалізації.

На початку 70-х років були сформовані теоретичні основи сучасних технологій БД і остаточно сформувався самостійний напрям інформаційних технологій – наука о базах даних. Головною подією цього періоду стало виникнення *реляційних* баз даних. В основі реляційної моделі лежить поняття відношення. Реляційна модель є простою і зрозумілою, а її фізичне представлення добре реалізується на комп'ютері. До недоліків реляційних моделей можна віднести складність реалізації ієрархічних і мережних зв'язків.

У 70-х роках почали формуватися підходи в БД, які пов'язані з використанням апарата логіки в якості моделі даних. Ці роботи привели до створення *дедуктивних* БД. Розвиток цього напряму дозволяє створювати *бази знань*.

У 80-х роках реляційні БД набули домінуючого положення. Однак уже тоді існувало і постійно розширювалося коло застосувань, для яких ця технологія була неадекватною. Це стосується мультимедійних застосувань, систем, які оперують просторовими даними і т.ін. В середині 80-х років успіхів досягло об'єктно-орієнтоване програмування. Під його впливом і в зв'язку з необхідністю реалізації нетрадиційних застосувань, вимоги яких погано узгоджуються з можливостями реляційних систем, почались роботи з практичної реалізації *об'єктно-орієнтованих* БД. Логічна структура об'єктно-орієнтованої БД зовні схожа на структуру ієрархічної БД, але вона доповнена об'єктно-орієнтованими механізмами. Об'єктно-орієнтовані БД у порівнянні з реляційними БД мають можливість відображати інформацію складних взаємозв'язках об'єктів, визначати функції обробки окремих записів. До недоліків об'єктно-орієнтованих моделей належить висока понятійна складність, низька швидкість виконання запитів, незручність обробки даних.

Бажання розробників реляційних БД зберегти лідируючі позиції і підвищити ефективність реляційної моделі, привели до розробки об'єктно-реляційної моделі, на основі якої почали розроблятися *об'єктно-реляційні* БД. В

цих моделях припускається застосування багатозначних полів – полів, які складаються з підзначень. Ці БД дозволяють забезпечити високу наглядність представлення інформації і підвищити ефективність її обробки. До недоліків об'єктно-реляційних моделей можна віднести складність рішення забезпечення цілісності і несуперечливості даних.

На початку 90-х років почало збільшуватися значення інформаційного забезпечення систем підтримки прийняття рішень. Це привело до необхідності створення багатомірних СУБД і на їх основі розробки *інформаційних сховищ*. Ці системи використовують історичну інформацію, яка представляється в агрегованому вигляді. На основі цієї інформації виконується аналітична обробка, прогнозування даних, а також інтелектуальний аналіз даних.

В цей час також велика увага приділялася розвитку розподілених систем. Успіхи в розробці комп'ютерних мереж стимулювали дослідження в технологіях *розподілених* БД, були розроблені архітектурні концепції *клієнт – сервер*.

Одним з найбільших досягнень 90-х років в області інформаційних технологій стало створення відкритої глобальної розподіленої неоднорідної гіпермедійної інформаційної системи, яка використовує комунікаційну мережу Internet. Ця система отримала назву WWW (Web). З самого початку виконувались спроби інтегрувати системи БД у Web. Одним з напрямів роботи є *інтеграція структурованих даних БД і слабкоструктурованих даних Web*, проводяться роботи зі створення БД на мові XML.

У даний час в багатьох комп'ютерних компаніях здійснюються роботи зі створення *цифрових бібліотек* – інформаційних систем, які призначені для зберігання, пошуку, обробки, аналізу і розповсюдження інформаційних ресурсів різної природи – структурованих і слабко-структурованих даних, мультимедійної інформації, повнотекстових документів. Для створення таких систем використовуються Web-технології, розробляються методи інтеграції неоднорідних ресурсів, розпізнавання і пошуку інформаційних ресурсів.

2. Компоненти банків даних

Банк даних – це система спеціальним чином організованих даних (баз даних), програмних, мовних, технічних, організаційно-методичних засобів призначених для підтримки інформаційної моделі предметної області з метою забезпечення інформаційних потреб користувачів (рис. 1.3).

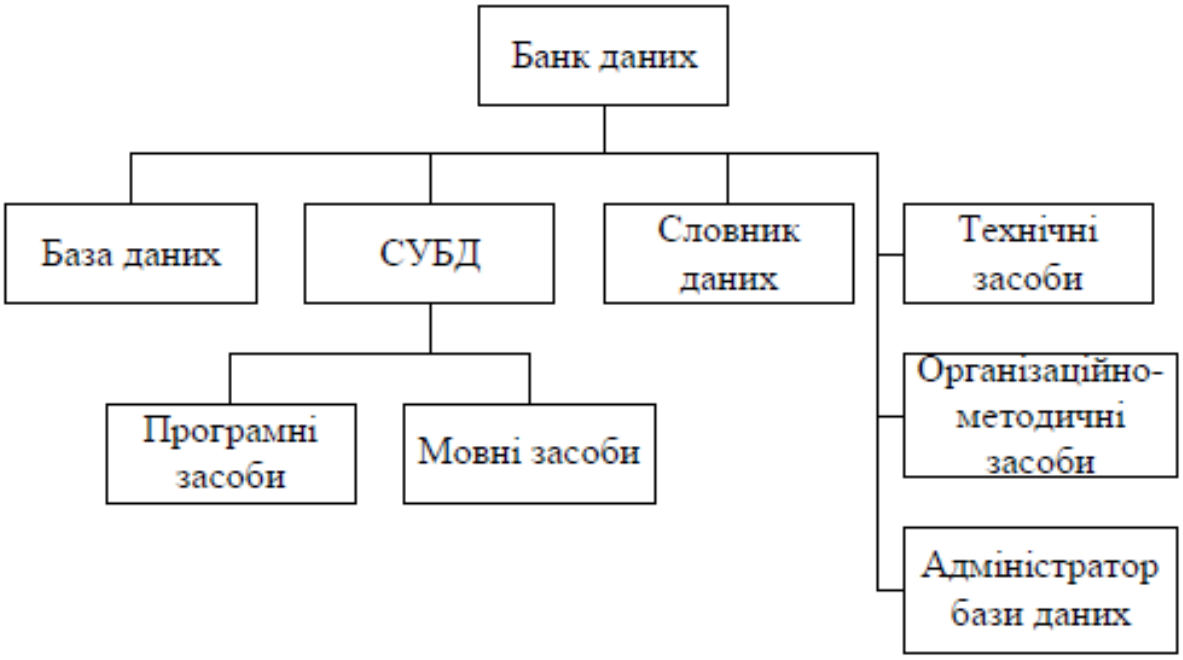


Рис. 1.3. Банк даних

База даних – пойменована сукупність взаємозв'язаних даних, які знаходяться під управлінням СУБД. В БД зберігаються дані, логічно пов'язані між собою. До головних властивостей БД належать такі:

– *цілісність* означає, що в будь-який момент часу відомості в БД повинні бути несуперечливі;

– *безпека* означає, що виконується захист даних від санкціонованого і несанкціонованого доступу;

– *відновленість* означає можливість відновлення БД після збоїв роботи системи.

Система управління базами даних (СУБД) – сукупність мовних і програмних засобів, які призначені для створення, ведення і сумісного використання БД багатьма користувачами.

До головних функцій СУБД належать такі:

− управління даними у зовнішній пам'яті і буферами оперативної пам'яті;

− управління транзакціми і паралельним доступом;

− відновлення БД;

− підтримка мов БД;

− контроль доступу до даних;

− підтримка цілісності даних;

− підтримка незалежності даних;

− підтримка обміну даними.

Склад БД містить не тільки дані, що зберігаються, але і опис БД. Опис БД належить до *метаінформації*, тобто інформації про інформацію. Опис БД часто називають *схемою*.

Централізоване сховище метаінформації називається словником даних.

Словник даних (*каталог даних*) – використовується для централізованого накопичення і опису ресурсів даних. Словник даних відповідає за визначення всіх елементів даних:

− імена, типи і розміри елементів даних;

− імена зв'язків;

− обмеження даних по підтримці цілісності;

− схеми БД (зовнішня, концептуальна і внутрішня), а також відображення між ними;

− імена користувачів і їх права доступу до даних;

− статистична інформація.

Програмні засоби БД включають в свій склад ядро СУБД, транслятори, утиліти, прикладні програми.

Мовні засоби поділяються на мови опису даних (МОД) і мови маніпулювання даними (ММД). МОД призначені для опису схеми БД або її частини. З її допомогою виконується опис типів даних, їх структур і зв'язків між собою. Відповідно до отриманого опису СУБД знаходить в програмі необхідні дані, перетворює їх і передає в прикладну програму. ММД виконує

функції вибірки з БД даних за певними умовами, зміну даних, додавання даних, вилучення даних і т.ін.

Адміністратор даних – людина, яка відповідає за управління даними (планування БД, розробку і супроводження стандартів, прикладних алгоритмів і ділових процедур), а також за концептуальне і логічне проектування БД.

Адміністратор БД – людина, яка відповідає за фізичну реалізацію БД (фізичне проектування і втілення проекту), за забезпечення безпеки і цілісності даних, за супроводження операційної системи, а також за забезпечення максимальної продуктивності застосувань і користувачів.

Адміністратор даних і адміністратор БД виконують функції: управління структурою БД, управління паралельною обробкою, розподіл прав і обов'язків при обробці, забезпечення безпеки БД, відновлення БД, управління СУБД, підтримка репозиторія даних.

Адміністрування даними і БД передбачає управління інформаційними ресурсами, проектування БД, управління реалізацією застосувань, підтримку цілісності даних, захист даних, спостереження за поточною продуктивністю системи, а також реорганізацію БД при необхідності.

Переваги і недоліки застосування СУБД наведені в табл. 1.1.

Таблиця 1.1

Переваги і недоліки застосування СУБД

Переваги СУБД	Недоліки СУБД
Мінімізація збитковості даних	Використання значної частини ресурсів на потреби СУБД, а не на прикладну задачу
Несуперечливість даних і контроль їх цілісності	Вартість СУБД
Незалежність прикладних програм від даних	Підвищені вимоги до технічного і програмного забезпечення
Підвищена безпека	Продуктивність
Розвинені засоби резервного копіювання і відновлення	Підвищені вимоги до кваліфікації робітників
Багатокористувацький режим роботи	Наслідки збоїв

3. Компоненти системи баз даних

Компонентами системи баз даних є БД, СУБД і прикладні програми, з якими працюють як розробники, так і користувачі.

В СУБД входять такі компоненти (рис. 1.4): ядро СУБД, підсистема засобів проектування і підсистема засобів обробки.

Ядро СУБД – містить сукупність базових механізмів СУБД, які використовуються при будь-яких варіантах конфігурації системи. Ядро СУБД виконує функцію посередника між підсистемами засобів проектування і обробки і даними. Сучасні БД у більшості представляють користувачу дані у вигляді таблиць. Ядро СУБД отримує запити від інших компонентів в термінах таблиць, стовпців, рядків і перетворює ці запити в команди операційної системи, які виконують запис і читання з фізичних носіїв інформації.

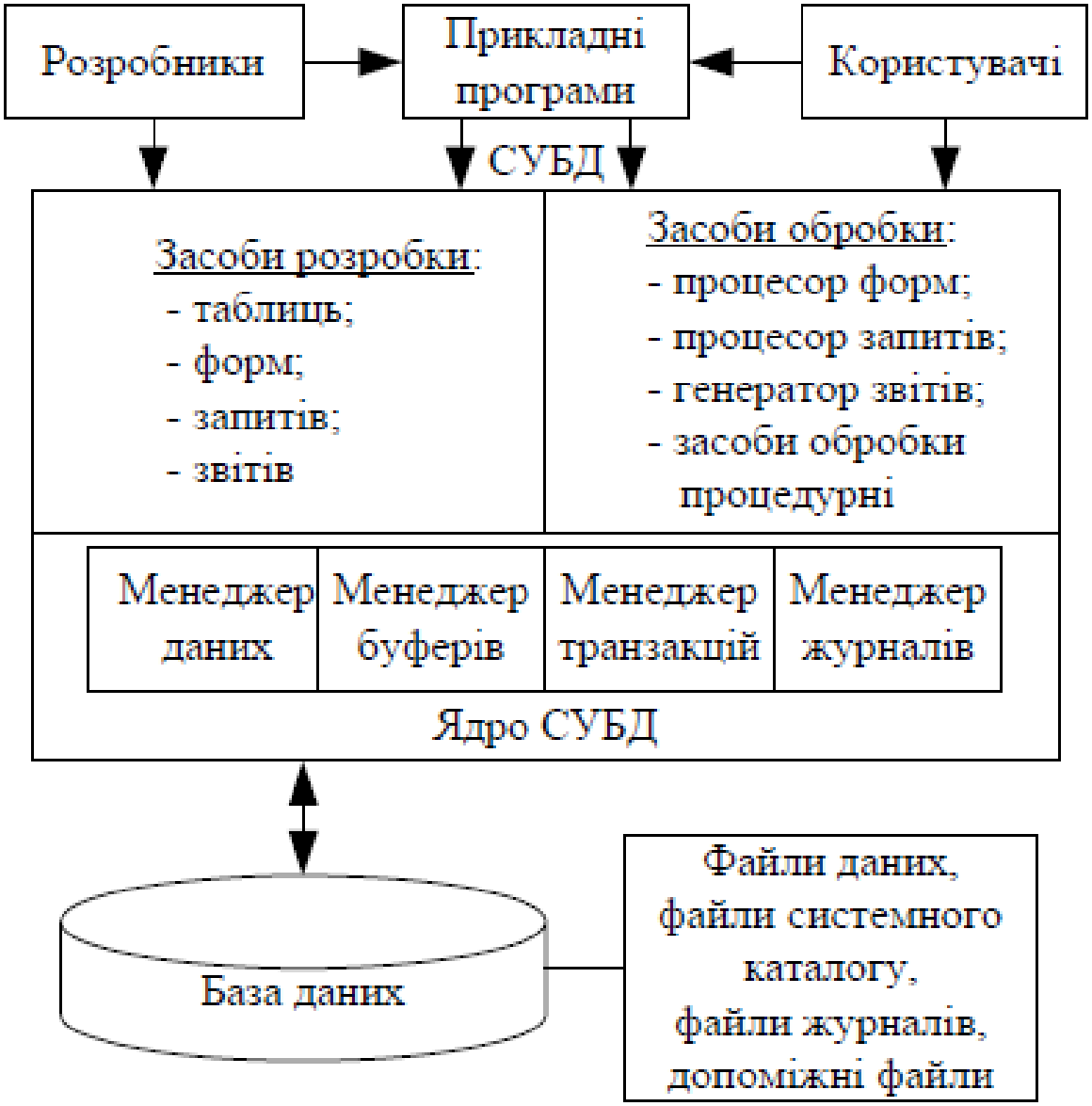


Рис. 1.4. Компоненти системи бази даних

Крім того, ядро СУБД задіяне в управлінні транзакціями, блокуваннях, резервному копіюванні і відновленні. В ядро СУБД входять менеджери буферів, даних, транзакцій, журналів.

Менеджер буферів – призначений для рішення задач ефективної буферізації оперативної пам'яті.

Менеджер даних – призначений для управління зовнішньою пам'ятью, забезпечення створення структур для даних, що зберігаються і допоміжних структур (індекси і т.ін.).

Менеджер транзакцій – підтримує механізми фіксації і відкату транзакцій, пов’язаний з менеджером буферів оперативної пам'яті і забезпечує зберігання всієї інформації, яка потрібна після збоїв системи.

Менеджер журналів – забезпечує реєстрацію відомостей про виконання транзакцій, про працюючих користувачів, про виконання застосування, про доступи до різних структур даних і т.ін.

Підсистема засобів проектування являє собою набір інструментів, які спрощують проектування і реалізацію баз даних і їх застосувань. Як правило, цей набір містить засоби для створення таблиць, форм, запитів й звітів. В СУБД є також мови програмування і інтерфейси до них.

Підсистема обробки здійснює обробку компонентів застосування, які створені за допомогою засобів проектування.

Застосування БД складається з форм, запитів, звітів, меню і прикладних програм. Форми, запити і звіти можна створювати за допомогою засобів, що постачаються у комплекті з СУБД. Прикладні програми повинні бути написані або на вхідній мові СУБД, або на одній зі стандартних мов, а потім за допомогою СУБД з'єднані з БД.

Контрольні запитання

1. Дати визначення таких термінів: *інформація*, *інформаційна система*, *дані*, *предметна область*.

2. Дати визначення *бази даних* і *СУБД*.

3. Пояснити, чому база даних є моделлю. Яка існує різниця між реальністю і моделлю реальності?

4. Перелічити основні функції СУБД.

5. Перелічити переваги і недоліки СУБД.

6. Дати визначення словника даних.

7. Назвати основні компоненти системи бази даних і пояснити функцію кожної з них.

8. Перелічити основні етапи розвитку технологій баз даних.

9. Що таке банк даних?

10. Які складові частини містить банк даних?

11. Кого називають адміністратором бази даних?

12. Що таке ядро СУБД?

13. Які підсистеми входять в СУБД?

ЛЕКЦІЯ. СЕРЕДОВИЩЕ БАЗ ДАНИХ

План:

1. Архітектура баз даних.
2. Моделі даних.
3. Програмні і мовні засоби баз даних.
4. Архітектура інформаційної системи.

Література

1. Гайна Г.А. Основи проектування баз даних: Навчальний посібник. К.: КНУБА, 2005. – 204 с.
2. Гайна Г.А. Організація баз даних і знань. Мови баз даних: Конспект лекцій.−К.:КНУБА, 2002. – 64 с.
3. *Гайна Г.А., Попович Н.Л.* Організація баз даних і знань. Організація реляційних баз даних: Конспект лекцій.−К.:КНУБА, 2000. – 76 с.

1. Архітектура баз даних

Для організації роботи з БД необхідно забезпечити незалежність прикладних програм від даних. Це обумовлено тим, що при зміні системи, а також з метою забезпечення ефективного обслуговування користувачів необхідно виконувати роботи щодо зміни методів зберігання даних в БД, шляхів доступу до даних, змінювати структури і формати даних та зв'язки між ними. Якщо не застосовувати спеціальні підходи і при написанні застосувань вводити програмний опис методів доступу, засобів зберігання даних, формати даних, то при будь-якій зміні в БД для перелічених випадків буде необхідно корегувати текст програми користувача, що потребує значних витрат.

Незалежність застосувань від даних забезпечується засобами СУБД. Цей підхід базується на тому, що користувачі застосовуючи БД, не знають внутрішнє представлення даних.

На рис. 1 показана трирівнева модель архітектури СУБД, що була запропонована Комітетом планування стандартів і норм SPARC (Standarts Planning and Requirements Committee) Американського національного інституту стандартів ANSI (American National Standarts Institute).

Опис структури даних на будь-якому рівні називається схемою. Існує три різних типи схем БД, які визначаються згідно з рівнями абстракції архітектури

СУБД. На самому верхньому рівні є декілька зовнішніх схем, які відповідають різним представленням даних. Цей рівень визначає точку зору на БД окремих застосувань. Кожне застосування бачить і обробляє тільки ті дані, які необхідні цьому застосуванню.

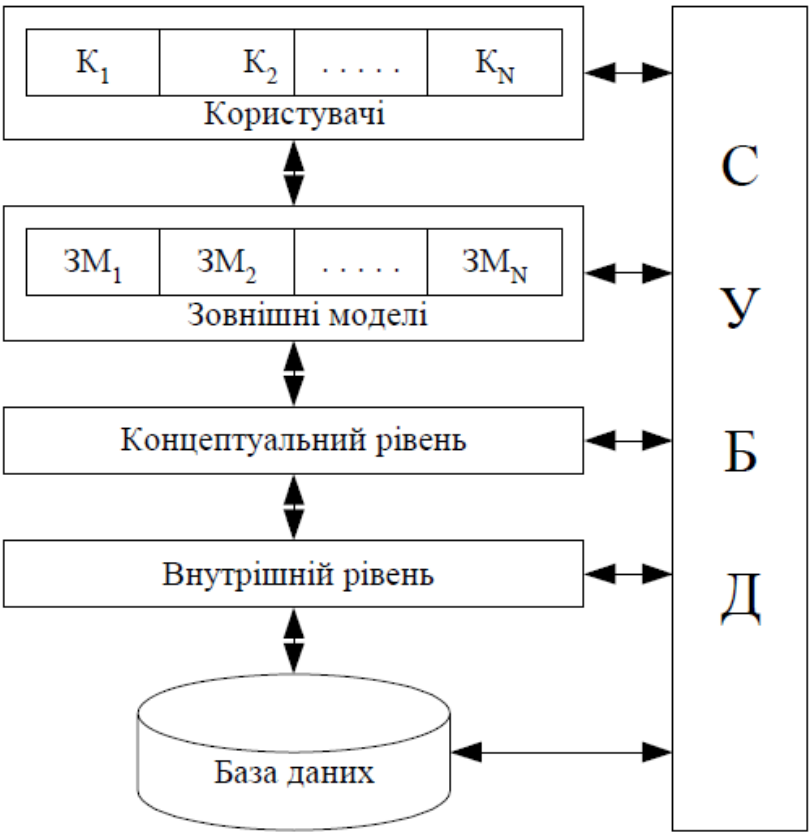


Рис. 1. Трирівнева архітектура СУБД

На концептуальному рівні опис БД називається концептуальною схемою. Тут БД представлена в найбільш загальному вигляді, який об'єднує дані, що використовуються всіма застосуваннями, які працюють з БД. Фактично концептуальний рівень відображає модель предметної області, для якої створювалася БД.

На внутрішньому рівні опис БД називається внутрішньою схемою. Тут БД представлена у вигляді безпосередньо даних, що розташовані в файлах, які відповідають фізичній організації БД.

Трирівнева архітектура СУБД дозволяє забезпечити незалежність від даних. Це означає, що зміни на нижніх рівнях не впливають на верхні рівні. Розрізняють логічну і фізичну незалежність при роботі з даними. Логічна незалежність від даних означає захищеність зовнішніх схем від змін, що вносяться в концептуальну схему. Зміни концептуальної схеми БД не

викликають необхідності в корегуванні існуючих зовнішніх схем для користувачів, і відповідно не викликають змін в застосуваннях, що працюють з цими схемами. Фізична незалежність від даних означає захищеність концептуальної і зовнішніх схем від змін, що вносяться у внутрішню схему. До змін внутрішньої схеми належать використання різних файлових систем або структур даних, різних пристроїв зберігання, модифікація пошукових структур тощо.

Крім трьох названих рівнів абстрагування в БД існує ще один рівень, що передує їм. Цей рівень відображає інформацію про предметну область, а модель цього рівня називається інфологічною моделлю предметної області. Таким чином головними рівнями абстрагування в БД є рівні:

− інфологічний;

− зовнішній;

− концептуальний;

− внутрішній.

Перехід від одного рівня абстрагування до наступного і складає в загальному вигляді процес проектування БД.

2. Моделі даних

Модель даних – це деяка абстракція, в якій знаходять своє відображення найбільш важливі аспекти функціонування визначеної предметної області, а другорядні – ігноруються. Модель даних являє собою деяку цільову модель предметної області. У моделі даних розрізняють три головні складові:

− структурна частина, яка визначає правила породження допустимих для даної СУБД видів структур даних;

− управляюча частина, яка визначає можливі операції над такими структурами ;

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

Моделювання даних – це процес створення логічного представлення структури бази даних.

На рис. 2 показана класифікація моделей даних.

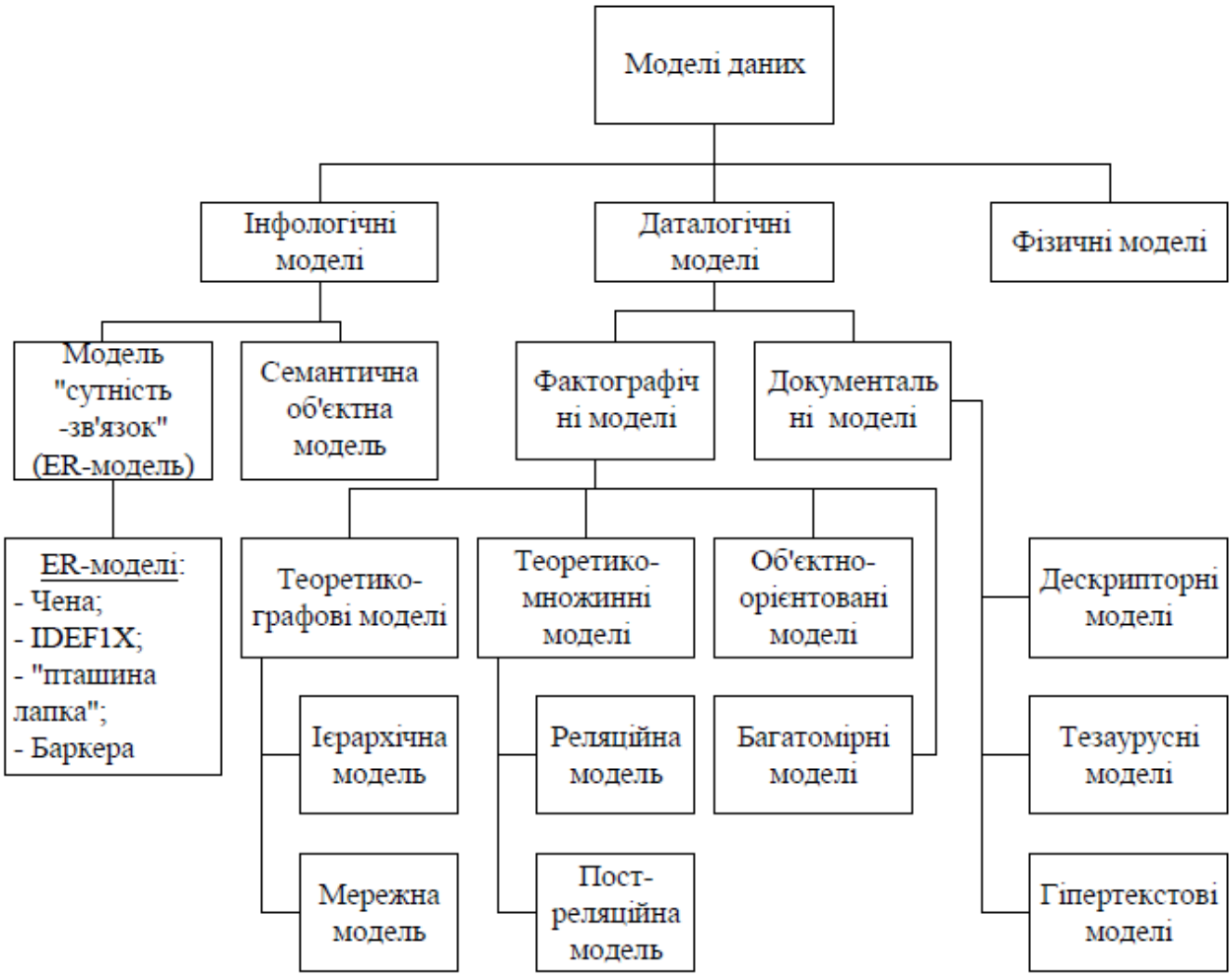


Рис. 2. Класифікація моделей даних

Кожному рівню представлення інформації відповідає певна модель.

Інфологічна модель – відображає інформацію про предметну область у вигляді незалежному від СУБД, що використовується. Ця модель відображає інформаційно-логічний рівень абстрагування, який пов'язаний з описом об'єктів предметної області, їх властивостей і взаємозв'язків.

Часто ці моделі ототожнюють з концептуальними моделями предметної області і називають концептуальними інфологічними моделями (внутрішня і зовнішня концептуальні інфологічні моделі).

Даталогічна модель – модель логічного рівня, яка відображає логічні зв'язки між елементами даних безвідносно до їх змісту і середовища збереження. Часто ці моделі ототожнюють з логічними моделями.

Фізична модель – описує те, як дані зберігаються в комп'ютері, представляючи інформацію про структуру записів, їх впорядкованість і про існуючі шляхи доступу до даних.

Модель "*сутність-зв'язок*" (ER-модель) – описує модель предметної області і складається з множини сутностей, множини звязків між сутностями, а також з атрибутів сутностей і зв'язків. В модель входить обмеження цілісності даних, що пов'язано з двома множинами сутностей і називається залежністю по існуванню. ER-моделі дозволяють графічно представляти моделі предметних областей. Вони є складовою частиною багатьох CASE-продуктів.

Семантична об'єктна модель – описує модель предметної області і являє собою модель даних. Ця модель складається з семантичних об'єктів, що містять сукупність атрибутів. Атрибути групуються у класи. Модель даних володіє більш розвиненими засобами відображення семантики у порівнянні з теоретико-множинними і теоретико-графовими моделями.

Теоретико-графова модель – модель даних, в якій дозволені структури даних можуть бути представлені у вигляді графа загального або спеціального виду, наприклад дерева. Необхідну групу операцій на мові маніпулювання даними, що засновані на цій моделі, представляють навігаційні операції. Операції над даними мають позаописовий характер.

Теоретико-множинна модель – модель даних, в якій використовується математичний апарат реляційної алгебри, реляційного обчислення, а операції над даними маніпулюють таблицями.

Фактографічні моделі – містять відомості, які представлені у вигляді спеціальним чином організованих сукупностей формалізованих записів даних.

Документальні моделі – передбачають, що в якості одиничного елемента інформації виступає неподільний на менші складові частини документ, а інформація про документ, як правило, не структурується, або структурується в обмеженому вигляді. В цих моделях в основному розглядаються тексти на природній мові, формати документів є вільними.

Ієрархічна модель – модель даних в основі якої використовується ієрархічна, деревоподібна структура даних. Вершинами цієї структури є записи,

які складаються з простих елементів даних різних типів. Батьківському запису відповідає довільне число екземплярів підлеглих записів кожного типу.

Мережна модель – модель даних, в якій дозволені структури даних можуть бути представлені у вигляді графа загального вигляду. Вершинами такого графа можуть бути дані різних типів – від атомарних елементів даних до записів складної структури. На відміну від ієрархічної моделі наступник в цій моделі може мати довільне число батьків.

Реляційна модель – модель даних, яка заснована на математичному понятті відношення і представленні відношень у формі таблиць.

Постреляційна модель – розширена реляційна модель, яка знімає обмеження неподільності даних, що зберігаються в записах таблиць. Ця модель допускає багатозначні поля – поля, значення яких складається з підзначень. Набір значень багатозначних полів вважається самостійною таблицею, яка вбудована в основну таблицю. Часто ці моделі ототожнюють з об'єктно-реляційними моделями.

Об'єктно-орієнтована модель – модель даних, яка базується на понятті об'єкта, тобто сутності, що володіє станом і поведінкою. Стан об'єкта визначається його атрибутами, а поведінка визначається сукупністю операцій, що визначені для цього об'єкта. Також передбачається можливість підтримки зв'язків між типами об'єктів.

Багатомірна модель – модель даних, яка оперує багатомірним представленням даних (у вигляді гіперкубу) і орієнтована на підтримку аналіза даних. Передбачається конструювання різноманітних агрегацій даних у межах гіперкубу, побудова різних його проекцій – підмножин гіперкубу, деталізація і обертання даних, а також цілий ряд інших операцій.

Дескрипторна модель – описує кожен документ за допомогою дескриптора. Дескриптор має жорстку структуру і являє собою набори деяких лексичних одиниць (слов, словосполучень, термінів), які потрібні для роботи з документами. Дескриптори між собою не зв'язані.

Тезаурусна модель – описує кожен документ за допомогою дескрипторів, а також змістовних відношень між лексичними одиницями (ціле-частина, род-

вид, клас-підклас і т.ін.). Ці моделі дозволяють підвищити ефективність дескрипторних моделей за рахунок більш ефективного відображення предметної області.

Гіпертекстова модель – модель, що заснована на розмітці документа за допомогою спеціальних навігаційних конструкцій, які відповідають змістовим зв'язкам між різними документами, або окремими фрагментами одного документа. Такі конструкції утворюють деяку семантичну мережу в базі документів.

2. Програмні даних

Основу програмних засобів банка даних складає СУБД. В СУБД можна виділити ядро СУБД, яке підтримує сукупність базових механізмів роботи з БД, а також інші компоненти, які забезпечують засоби тестування, налагодження системи, утиліти, які забезпечують виконання таких додаткових функцій, як відновлення БД, збір статистики і т.ін. Важливою компонентою СУБД є транслятори і компілятори для мов, що використовуються. Для роботи з БД розробляються застосування.

Застосування – програма, яка призначена для рішення деякої сукупності задач в даній предметній області, або яка являє собою типовий інструментарій, що застосовується в різних предметних областях. Застосування може використовувати різні джерела даних (фактографічні, документальні, WEB і т.ін.), мати різну архітектуру (дволанкову, триланкову, розподілену).

Застосування бази даних – застосування, яке використовує ресурси деякої системи баз даних. Для доступа до БД використовується інтерфейс прикладного програмування СУБД, в середовищі якої він підтримується. Застосування можуть бути написані на стандартній алгоритмічній мові програмування (Pascal, C, Basic тощо) з вбудованими операторами на мові SQL.

Мова даних – мова, яка призначена для визначення даних, маніпулювання даними, а також інших функцій в термінах понять і рамках можливостей, які передбачені в моделі даних, що підтримується розглядуваною СУБД.

Мова запитів – мова доступу до БД, яка орієнтована на користувача. Мова запитів належить до декларативних мов, описує властивості і взаємозв'язки сутностей, але не описує алгоритм рішення задачі. Як правило мова запитів використовується в інтерактивному режимі, а також може вбудовуватися в програмний код застосувань.

Мова маніпулювання даними (Data Manipulation Language – DML) – мова, яка реалізує операційні можливості моделіданих, що використовується. Ця мова визначає операції, які допустимі над даними, що знаходяться в БД.

Мова визначення даних (Data Definition Language – DDL) – мова, яка служить для опису структури БД, обмежень цілісності, а також, можливо, для специфікації процедур, що зберігаються, тригерів, обмежень управління доступом і т.ін. Функціональні можливості мов визначення і маніпулювання можуть інтегруватися в єдину мову даних.

Мова програмування баз даних – мова, яка забезпечує концептуально єдине інтегроване середовище, яке засновано на єдиній моделі даних, для програмування застосувань і управління даними в БД. Такі мови об'єднують функції традиційних мов програмування із засобами опису і маніпулювання даними в БД.

Мова програмування базова – традиційна мова програмування, для якої дана СУБД забезпечує інтерфейс прикладного програмування (API). Прикладна програма, яка написана на цій мові, має доступ до деяких функціональних можливостей СУБД і може виконувати з її допомогою доступ до БД.

Мови, які належать до мов четвертого покоління (Fourth-Generation Language – 4GL), мають такі функціональні можливості:

− генератори екранних форм для створення шаблонів вводу і відображення даних;

− генератори звітів на основі інформації, що зберігається в БД;

− генератори застосувань для створення програм обробки даних;

− генератори запитів;

− генератори для представлення даних у вигляді різного роду діаграм.

Для формування запиту за допомогою різних СУБД найчастіше використовуються дві основні мови опису запитів:

− *SQL (Structured Query Language)* – структурована мова запитів;

− *QBE (Query By Example)* – мова запитів за зразком.

Головна різниця між цима мовами полягає в тому, що мова QBE передбачає ручне або візуальне формування запиту, а мова SQL – програмування запиту.

Мова SQL є найбільш поширеною мовою для роботи з БД.

На даний час існують такі міжнародні стандарти на мову SQL: SQL1, SQL2, SQL3.

Мова SQL не володіє функціями повноцінної мови розробки і орієнтована на доступ до БД. Використання мови SQL може бути самостійним і вона може включатися в склад засобів розробки програм. В цьому випадку її називають вбудованим SQL. Розрізняють два головних методи використання вбудованого SQL: статичний і динамічний.

Статичне використання передбачає застосування в програмі функцій викликів мови SQL, які включаються в програмний модуль і виконуються після компіляції програми.

Динамічне використання передбачає динамічну побудову викликів функцій мови SQL та інтерпретацію цих викликів у ході виконання програми. Динамічний метод застосовується тоді, коли вид SQL запиту заздалегідь невідомий і будується у діалозі з користувачем.

Будь-яке SQL-застосування реляційної БД складається з трьох частин: інтерфейса користувача, набору таблиць в БД і SQL-машини.

4. Архітектура інформаційної системи

Ефективність функціонування інформаційної системи багато в чому залежить від її архітектури. Функціональні частини інформаційної системи можуть розміщуватися на одному або на декількох комп'ютерах. У разі, якщо інформаційна система розміщується на одному комп'ютері, можливі такі варіанти використання програмних засобів:

застосування і СУБД, застосування і ядро СУБД, незалежне застосування.

У першому випадку взаємодія користувача і СУБД виконується або напряму через користувацький інтерфейс СУБД, або за допомогою застосування (рис. 3).

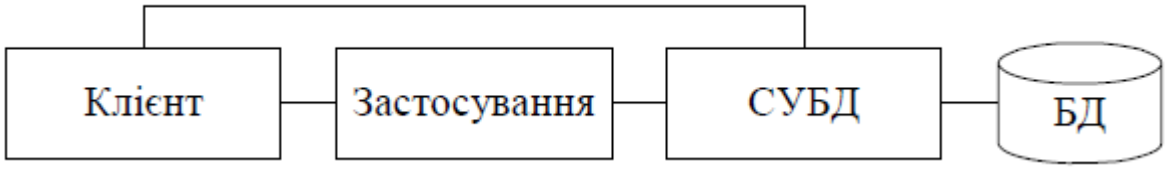


Рис. 3. Використання застосування і СУБД

У другому випадку взаємодія користувача і СУБД виконується за допомогою застосування (рис. 4). Такий підхід дозволяє підвищити швидкість роботи застосування, зменшити об'єм необхідної пам'яті.

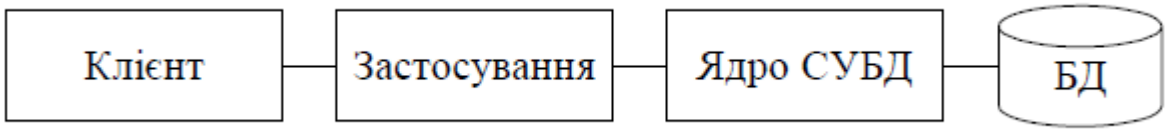


Рис. 4. Використання застосування і ядра СУБД

Створення незалежних застосувань дозволяє звертатися до БД без СУБД (рис. 5). Такий підхід дозволяє ще більше підвищити швидкість роботи застосування, зменшити об'єм необхідної пам'яті. Недоліки такого підходу пов'язані з трудомісткістю доробки застосувань, відсутністю стандартних засобів СУБД по обслуговуванню БД.

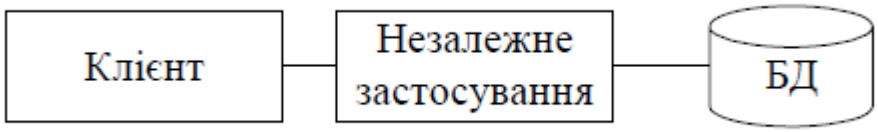


Рис. 5. Використання незалежного застосування

При інтеграції комп'ютерів в мережі виникає можливість розподілу застосувань, що працюють з єдиною БД, а також самої БД по мережі. Найбільш поширеною є схема, при якій кожен користувач маю свою персональну БД (КБД) і звертається до серверної БД (СБД) за інформацією, що спільно використовується багатьма користувачами (рис. 6).

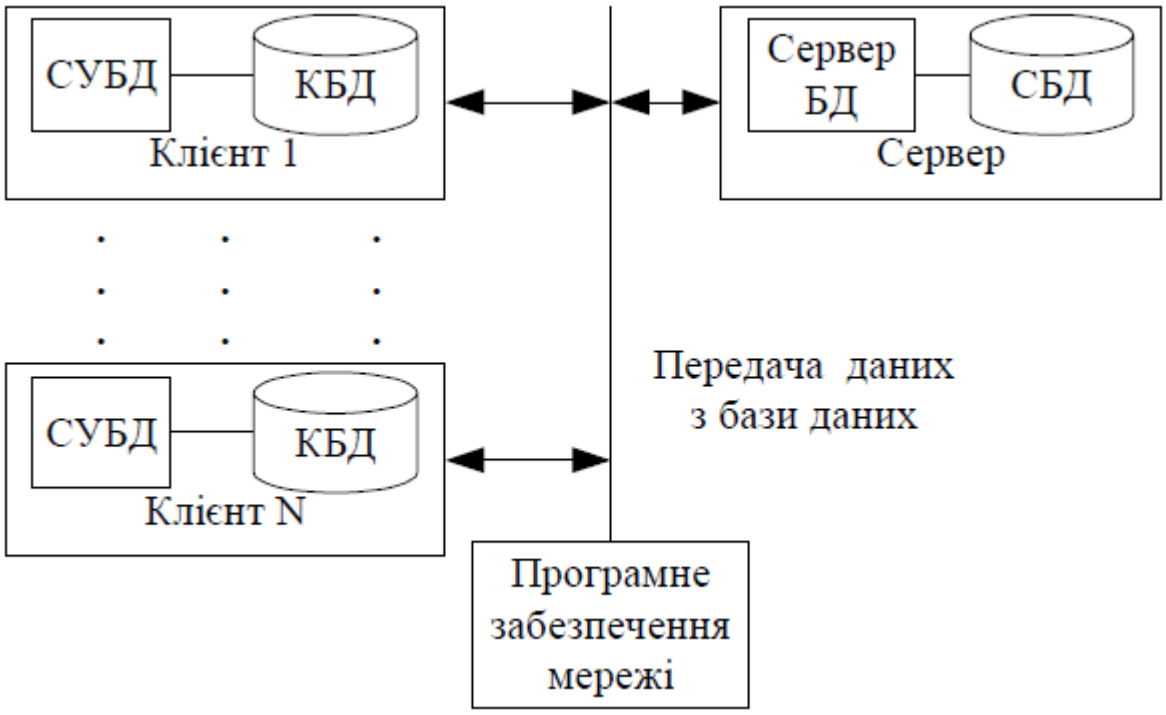


Рис. 6. Використання сервера БД

Під сервером розуміється комп'ютер або програма, які керують певними ресурсами. Клієнт – це теж комп'ютер або програма, які використовують цей ресурс.

Такий підхід дозволяє поєднувати переваги централізованого зберігання з індивідуальною роботою користувачів.

Контрольні запитання

1. Представити та пояснити архітектуру бази даних.
2. Які головні рівні абстрагування в базах даних?
3. Назвати основні моделі даних і дати їх характеристику.
4. В чому різниця між реляційною і об'єктно-реляційною моделями?
5. Які переваги і недоліки об'єктно-орієнтованої моделі?
6. Дати класифікацію програмних і мовних засобів по роботі з БД.
7. Вказати основні архітектурні рішення інформаційної системи з базами даних.
8. Що таке схема БД? Перелічити її компоненти.
9. Що розуміється під незалежністю даних?

Архітектура БД. Моделі даних.

Архітектура БД

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

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

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

Розглянемо такі важливі для нас поняття як «дані» і «інформація». Незважаючи на величезну кількість визначень для цих понять зупинимося на наступних визначеннях.

Інформація являє собою відомості про оточуючих людини предмети, явища і процеси і є об'єктом таких операцій як сприйняття, передача, перетворення, зберігання і використання.

Коли використовується термін «дані», то мова йде про інформацію, представленої в формалізованому вигляді, придатної для автоматичної обробки при можливу участь людини.

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

При створенні БД в основному мають на меті впорядкувати дані за різними ознаками, щоб мати можливість брати з даних потрібну інформацію.

Створення БД, її підтримка, управління, а також доступ користувачів до самих даних здійснюється за допомогою спеціальних програмних продуктів, які називаються системами управління базами даних (СКБД).

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

Файли, забезпечені описом збережених у них даних і знаходяться під управлінням СУБД, стали називати БД [4].

Функції СУБД:

- управління буферами оперативної пам'яті;
- управління транзакціями;
- захист від відмов і відновлення (журнализация);
- забезпечення різних рівнів доступу до даних.

Термін «архітектура» може мати різні тлумачення. Наприклад, коли вказують-ся функціональні модулі системи й способи їхньої взаємодії, йдеться про функці-ональну архітектуру. Спосіб реалізації функцій системи, її компоненти та вза-ємозв'язки між ними фіксує архітектура реалізації системи. У цьому контексті можна також згадати архітектуру технічних засобів систем. Говорячи ж про термін «архітектура БД», ми матимемо на увазі архітектуру інформаційного за-безпечення.

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

За принципами обробки даних БД класифікуються на централізовані і розподілені.

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

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

В основі концепції клієнт-сервер лежить ідея про те, що крім зберігання файлів БД, центральний сервер повинен виконувати основну частину обробки даних. Користувачі звертаються до сервера за допомогою спеціальної мови структурованих запитів (SQL, Structed Query Language), на котрому описується список завдань, які виконуються сервером. Запити приймаються сервером і породжують процеси обробки даних. У відповідь користувач отримує вже відпрацьований набір даних. Технологія клієнт-сервер дозволяє уникнути передачі по мережі величезних обсягів інформації, переклавши всю обробку на центральний сервер. Такий підхід також дозволяє уникнути конфліктів при редагуванні одних і тих же даних безліччю користувачів [2].

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

Клієнтська частина ("тонкий клієнт"), що взаємодіє з користувачем, являє собою HTML-сторінку в Web-браузері або Windows-додаток, що взаємодіє з Web-сервісами. Вся програмна логіка винесена на сервер додатків, який забезпечує

формування запитів до бази даних, що передаються на виконання сервера баз даних. Сервер додатків може бути Web-сервером або спеціалізованою програмою (див. Рис. 2.1).

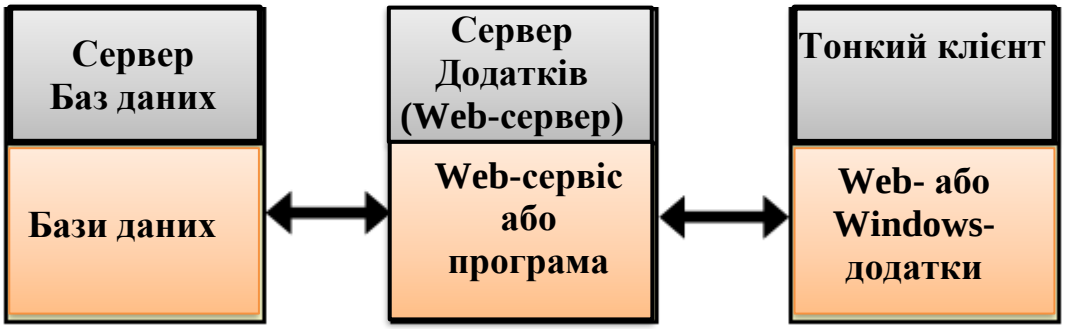


Рисунок 2.1 - Схема роботи з БД в трирівневої архітектурі

Розподілена БД розташовується на кількох комп'ютерах. Інформація на цих комп'ютерах може перетинатися і навіть дублюватися. Для управління такими БД призначена система управління розподіленими БД. Система приховує від користувачів доступу до даних, розташованим на інших комп'ютерах. Для користувача все виглядає так, як ніби вся інформація знаходиться на одному сервері.

Моделі даних

Виділяють наступні моделі даних:

- інфологічні;
- даталогічні;
- фізичні.

2.1. Інфологічна модель даних

Процес проектування БД починається зі створення інфологічної моделі.

Інфологічна модель даних - узагальнене неформальний опис створюваної бази даних, виконане з використанням природної мови, математичних формул, таблиць, графіків і інших засобів, зрозумілих всім людям, які працюють над проектуванням бази даних .

Інфологічна модель даних - узагальнене, Неприв'язані до будь-яких СУБД опис предметної області.

Інфологічна модель відображає реальний світ у деякі зрозумілі людині концепції, повністю незалежні від параметрів середовища зберігання даних. Тому інфологіческая модель не повинна змінюватися до тих пір, поки якісь зміни в реальному світі не зажадають зміни в ній деякого визначення, щоб ця модель продовжувала відображати предметну область.

Існує безліч підходів до побудови таких моделей: графові моделі, семантичні мережі, модель «сутність-зв'язок» та ін. [5, 6].

2.1.1. Семантичні мережі [2]

Семантична мережа (СМ) - це граф, дуги якого є відносини між вершинами (значеннями). Семантичні мережі з'явилися при вирішенні завдань розбору і розуміння сенсу природної мови. Приклад семантичної мережі для пропозиції типу "Постачальник здійснив поставку виробів на замовлення клієнта до 1 червня 2004 року в кількості 1000 штук" наведено на рис. 2.2.

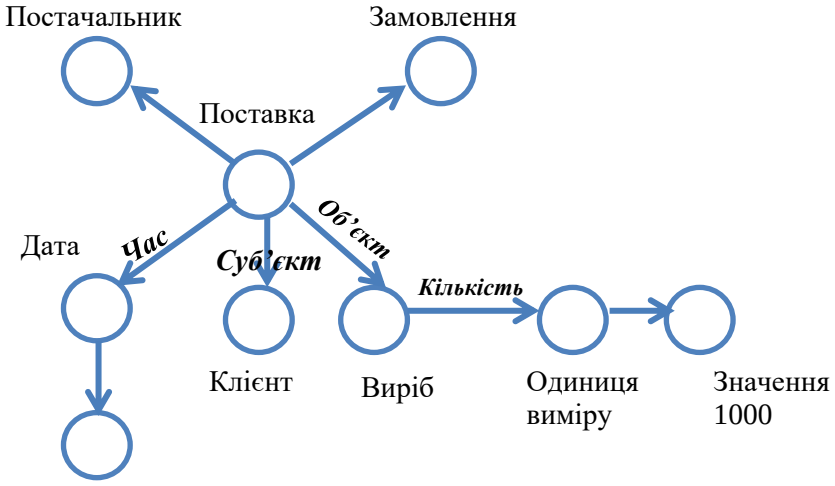


Рисунок 2.2 – Приклад семантичної мережі

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

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

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

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

Інструмент - то засіб, який використовується агентом для виконання дії, наприклад, "Роббі відкрив двері за допомогою ключа".

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

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

Траєкторія - переміщення від пункту відправлення до пункту призначення: "Вони пройшли через двері по сходах на сходи".

Засіб доставки - то в чому або на чому відбувається переміщення: "Він завжди їде додому на метро".

Місцезнаходження - то місце, де відбулося (відбувається, буде відбуватися) дію, наприклад, "Він працював за столом".

Споживач - то особа, для якого виконується дія: "Роббі зібрав кубики для Суззі".

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

Час - вказує на момент вчинення дії: "Він закінчив свою роботу пізно ввечері".

Найбільш типовий спосіб виведення в семантичних мережах (СС) - це спосіб зіставлення частин мережевої структури. Це видно на наступному простому прикладі, представленому на рис. 2.3.

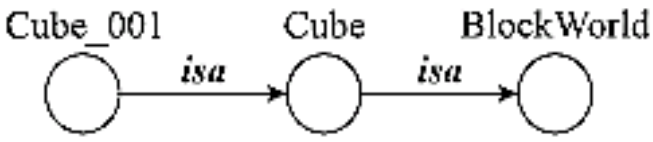


Рисунок 2.3 – Процедура співставлення СМ

Куб Cube належить світу BlockWorld.

Куб Cube_001 є різновид куба Cube.

Легко зробити висновок: Куб Cube_001 є частина світу BlockWorld.

2.2 Даталогічна модель

Інфологічна модель повинна бути відображена в даталогіческую модель, «зрозумілу» СУБД.

Даталогічна модель - опис мовою конкретної СУБД.

2.2.1. Ієрархічна модель

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

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

Прикладом ієрархічної моделі даних може служити адреса. На першому рівні (корені дерева) лежить наша планета - Земля. На другому - країна. На третьому - регіон (республіка, край, район), потім - населений пункт, вулиця, будинок, квартира (рис.2.4).

Ще один приклад - це система доменних імен в Інтернеті.

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

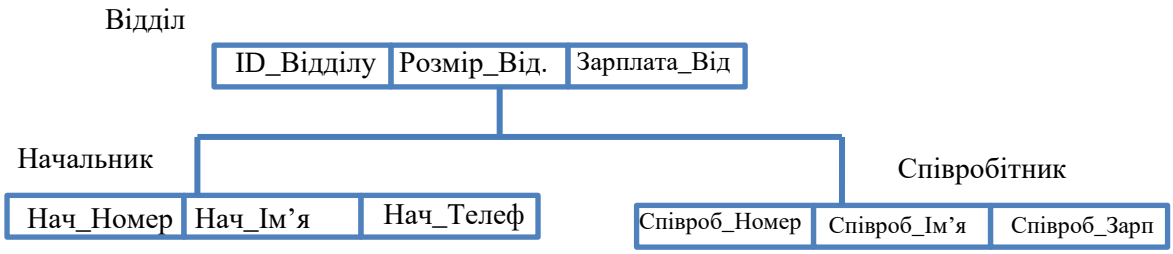


Рисунок 2.4 – Приклад ієрархічної структури

В даному прикладі відділ є предком для Начальник і Співробітники, а Начальник і Співробітники - нащадки Відділ. Між типами записи підтримуються зв'язки.

2.2.2. Мережева модель

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

Мережевий підхід до організації даних є розширенням ієрархічного.

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

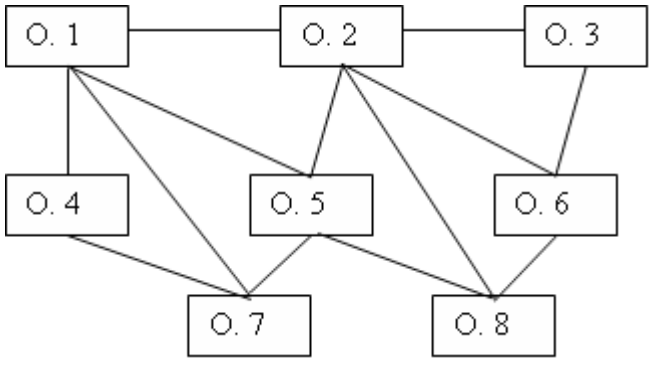


Рисунок 2.5 – Схема мережевої архітектури

Мережеві моделі також створювалися для мало ресурсних ЕОМ. Це досить складні структури, що складаються з "наборів" - пойменованих дворівневих дерев. "Набори" з'єднуються за допомогою "записів-зв'язок", утворюючи ланцюжки і т.д. При розробці мережевих моделей було вигадано безліч "маленьких хитрощів", що дозволяють збільшити продуктивність СУБД, але істотно ускладнити останні. Прикладний програміст повинен знати масу термінів, вивчити декілька внутрішніх

мов СУБД, детально представляти логічну структуру бази даних для здійснення навігації серед різних примірників, наборів, записів і т.п.

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

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

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

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

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

1. Рядки таблиць упорядковані системою в деякої фізичної послідовності.

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

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

Підтримуються два класи операторів:

1. Оператори, встановлюють адресу записи, серед яких:

1.1. прямі пошукові оператори (наприклад, знайти перший запис таблиці по деякому шляху доступу);

1.2. оператори, що знаходять запис в термінах відносної позиції від попередньої записи по деякому шляху доступу.

2. Оператори над адресованими записами

Типовий набір операторів:

– LOCATE FIRST - знайти перший запис таблиці T у фізичному порядку; повертає адресу записи;

– LOCATE FIRST WITH SEARCH KEY EQUAL - знайти перший запис таблиці T з заданим значенням ключа пошуку K; повертає адресу записи;

– LOCATE NEXT - знайти перший запис, наступну за записом з заданим адресою в заданому шляху доступу; повертає адресу записи;

– LOCATE NEXT WITH SEARCH KEY EQUAL - знайти следует запис таблиці T в порядку шляхи пошуку з заданим значенням K; має бути відповідність між використовуваним способом сканування і ключем K; повертає адресу записи;

– LOCATE FIRST WITH SEARCH KEY GREATER - знайти перший запис таблиці T в порядку ключа пошуку K со значенням ключового поля, великим заданого значення K; повертає адресу записи;

– RETRIVE - вибрати запис з вказаною адресою;

– UPDATE - оновити запис з вказаною адресою;

– DELETE - видалити запис з вказаною адресою;

– STORE - включити запис в зазначену таблицю; операція генерує адреса записи.

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

Однак такі СУБД мають ряд обмежень на кількість файлів для зберігання даних, кількість зв'язків між ними, довжину запису і кількість її полів [4, 7].

ЛЕКЦІЯ. РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ

План:

1. Базові поняття.
2. Цілісність бази даних.
3. Реляційна алгебра.
4. Обчислення кортежів.
5. Обчислення доменів.

Література

1. Гайна Г.А. Основи проектування баз даних: Навчальний посібник. К.: КНУБА, 2005. – 204 с.
2. Гайна Г.А. Організація баз даних і знань. Мови баз даних: Конспект лекцій.−К.:КНУБА, 2002. – 64 с.
3. *Гайна Г.А., Попович Н.Л.* Організація баз даних і знань. Організація реляційних баз даних: Конспект лекцій.−К.:КНУБА, 2000. – 76 с.

1. Базові поняття.

Реляційна модель даних заснована на математичному понятті відношення і представленні відношень у вигляді таблиць. Запропонована на початку 70-х років американським вченим Е.Коддом. В будь-якій реляційній СУБД припускається, що користувач сприймає БД як набір таблиць. Це стосується тільки логічної структури БД, тобто відноситься до концептуального і зовнішнього представлень.

На фізичному рівні БД реалізується за допомогою різних структур зберігання. В табл. 1 наведені елементи реляційної моделі. Для однозначної ідентифікації рядків, для зв'язування таблиць між собою, для прискорення операцій над даними застосовують ключі. В табл. 2 наведені можливі види реляційних ключів. Зовнішній і відповідний йому потенційний ключі повинні бути визначені на одному домені.

Таблиця 1

Елементи реляційної моделі

Елементи реляційної моделі	Форми представлення
Відношення	Таблиця
Кортеж	Рядок таблиці
Атрибут	Заголовок стовпця таблиці
Ключ	Сукупність атрибутів, які унікально визначають

	кожен рядок таблиці, або виконують функції зв'язування таблиць, або дозволяють прискорити операції над таблицями
Домен	Множина значень атрибута
Схема відношення	Рядок заголовків стовпців таблиці

Таблиця 2

Реляційні ключі

Назва	Пояснення
Потенційний ключ (Candidate Key)	Мінімальна підмножина атрибутів відношення, які єдиним чином ідентифікують кортеж даного відношення
Первинний ключ (Primary Key)	Потенційний ключ, який обрано для унікальної ідентифікації кортежів відношення
Вторинний ключ (Secondary Key)	Ключ, кожному значенню якого може відповідати більш ніж один екземпляр індексованих даних
Зовнішній ключ (Foreign Key)	Сукупність атрибутів відношення, значення яких є одночасно і значеннями первинного або потенційного ключа іншого відношення

Приклад. Розглянемо відношення Студент (рис. 1).

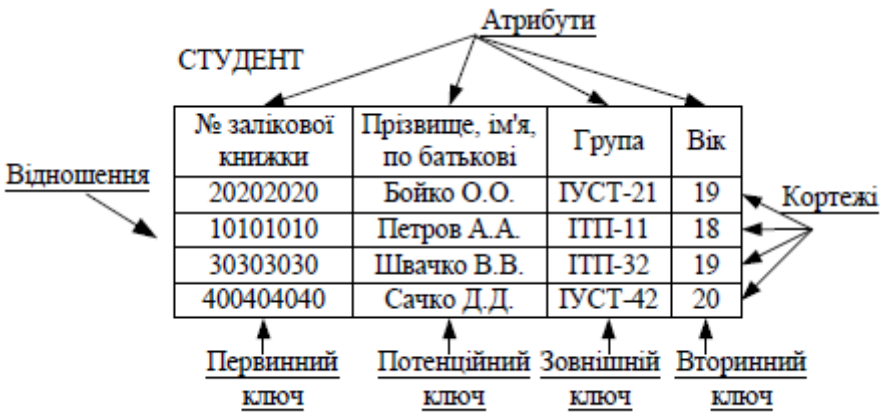


Рис. 1. Відношення *Студент*

Порядок кортежів у відношенні не визначений. В реляційних СУБД для зручності кортежі впорядковують за допомогою ключів (первинних або вторинних). В якості первинного ключа виступає атрибут № залікової книжки, який дозволяє унікально ідентифікувати кожен кортеж. Атрибут Вік обирається в якості вторинного ключа (не є обов'язковим) для виконання операцій сортування і групування студентів за віком. Атрибут Група обирається в якості

зовнішнього ключа для зв'язування з таблицею Група (на рис. 1 не представлена). Домени показують множину всіх можливих значень певного атрибута відношення. Наприклад, для атрибута Вік значення домену відноситься до типу цілих чисел.

Реляційна модель складається з таких частин:

− структурна (тут фіксується відношення як єдине ціле);

− маніпуляційна (тут використовуються два базових механізми маніпулювання реляційною БД – реляційна алгебра і реляційні обчислення);

− цілісність (тут використовується механізм, який запобігає руйнуванню даних).

Реляційній моделі даних властиві простота і природність використовуваних структур даних і операцій маніпуляції даними, повна незалежність від середовища зберігання даних, підтримка віртуальних, а не фізичних зв'язків між даними (на основі значень даних, а не покажчиків).

Реляційна БД включає в себе такі складові:

− інформаційні масиви (таблиці, індекси);

− системна інформація (структура БД, обмеження цілісності);

− прикладні програми (процедури, тригери).

Операційні можливості відношення мають дві еквівалентні форми – реляційна алгебра і реляційне обчислення. У свою чергу реляційне обчислення поділяється на реляційне обчислення зі змінними кортежами, яке називається обчислення кортежів, і на реляційне обчислення зі змінними доменами, яке називається обчислення доменів. Для виконання запитів до БД Е.Кодд запропонував відповідні принципи побудови трьох мов.

Мови запитів реляційної алгебри – це алгебраїчні мови, які дозволяють висловлювати запити засобами спеціалізованих операторів, що застосовуються до відношень.

Мови реляційного обчислення дозволяють висловлювати запити шляхом специфікації предиката, якому повинні відповідати потрібні кортежі (домени).

Реальні мови запитів (SQL, QBE і т.ін.) забезпечують не тільки функції відповідної теоретичної мови, але і реалізують деякі додаткові операції (арифметичні, друку і т.ін.).

2. Цілісність баз даних.

Цілісність баз даних – властивість даних, що визначає повноту і правильність інформації, яка вміщується в БД. Підтримка цілісності даних включає такі складові:

− структурна цілісність;

− обмеження реальних значень даних;

− посилкова цілісність.

Структурна цілісність передбачає виконання таких умов:

− наявність тільки однорідних структур даних типу "реляційне відношення";

− відсутність дублікатів кортежів;

− обов'язкова наявність у кожному відношенні первинного ключа;

− обмеження доменів, яке передбачає визначення кожного атрибуту на своєму домені;

− можливість застосування невизначених значень NULL (позначає відсутність будь-якого значення атрибуту).

Невизначене NULL значення розглядається, як значення невідоме на даний момент часу. Це значення при появі додаткової інформації може бути замінено на деяке конкретне значення. Введення NULL викликало необхідність застосування замість двозначної логіки тризначної логіки. У цьому випадку передбачаються реляційні операції з невизначеними значеннями.

Обмеження реальних значень даних вимагають, щоби значення поля належали деякому діапазону значень, або задовольняли певному арифметичному співвідношенню між значеннями різних полів. Обмеження значень можуть включати також визначення певних форматів для полів, задоволення значень полів певним статистичним умовам, бізнес правилам предметної області і т.ін.

Посилкова цілісність означає, що зміни в таблицях повинні виконуватися синхронно, а зміст двох пов'язаних таблиць має відповідати таким правилам:

− кожному запису основної таблиці відповідає нуль або більше записів підлеглої таблиці;

− в підлеглій таблиці немає записів, які не мають батьківських записів в основній таблиці;

− кожний запис підлеглої таблиці має тільки один батьківський запис основної таблиці.

Умови цілісності даних визначають, які дані можуть бути записані в БД у результаті додавання або оновлення даних. При маніпулюванні даними в таблицях виконується контроль дій відповідно до табл. 3.

Таблиця 3

Правила вилучення і оновлення

Операція	Правило	Пояснення
Вилучення (DELETE)	RESTRICT	Заборона вилученні рядка з батьківської таблиці, якщо в підлеглій таблиці цей рядок має нащадків
	CASCADE	При вилученні рядка з батьківської таблиці в підлеглій таблиці всі рядки-нащадки автоматично вилучаються
	SET NULL	При вилученні рядка з батьківської таблиці в підлеглій таблиці всім зовнішнім ключам рядків-нащадків автоматично присвоюється значення NULL
	SET DEFAULT	При вилученні рядка з батьківської таблиці в підлеглій таблиці всім зовнішнім ключам рядків-нащадків автоматично присвоюється певне значення встановлене за замовчуванням
Оновлення (UPDATE)	RESTRICT	Заборона зміни первинного ключа в рядку батьківської таблиці, якщо в підлеглій таблиці цей рядок має нащадків
	CASCADE	При зміні первинного ключа в рядку батьківської таблиці в підлеглій таблиці відповідні значення зовнішнього ключа

		також автоматично змінюються у всіх рядках-нащадках для того, щоби відповідати новому значенню первинного ключа
	SET NULL	При зміні первинного ключа в рядку батьківської таблиці в підлеглій таблиці відповідні значення зовнішнього ключа також автоматично змінюються у всіх рядках-нащадках і їм присвоюється значення NULL
	SET DEFAULT	При зміні первинного ключа в рядку батьківської таблиці в підлеглій таблиці відповідні значення зовнішнього ключа також автоматично змінюються у всіх рядках-нащадках і їм присвоюється певне значення встановлене за замовчуванням

Також можливо виконання правила NONE – не виконуються ніякі дії і правила NULL ALLOWED – дозволяються невизначені значення.

При введенні нових рядків (INSERT) необхідно дотримуватися такої послідовності введення: спочатку дані вводяться в батьківську таблицю, а потім − в підлеглу.

3. Реляційна алгебра.

Алгеброю називається множина об'єктів із заданою на ній сукупністю операцій, які замкнені відносно цієї множини.

Основною множиною в реляційній алгебрі є множина відношень. Варіант реляційної алгебри, запропонований Коддом, містить такі основні операції: об'єднання, різниця, перетин, декартовий добуток, проекція, селекція, з'єднання, ділення. На рис. 2 показані основні операції реляційної алгебри.

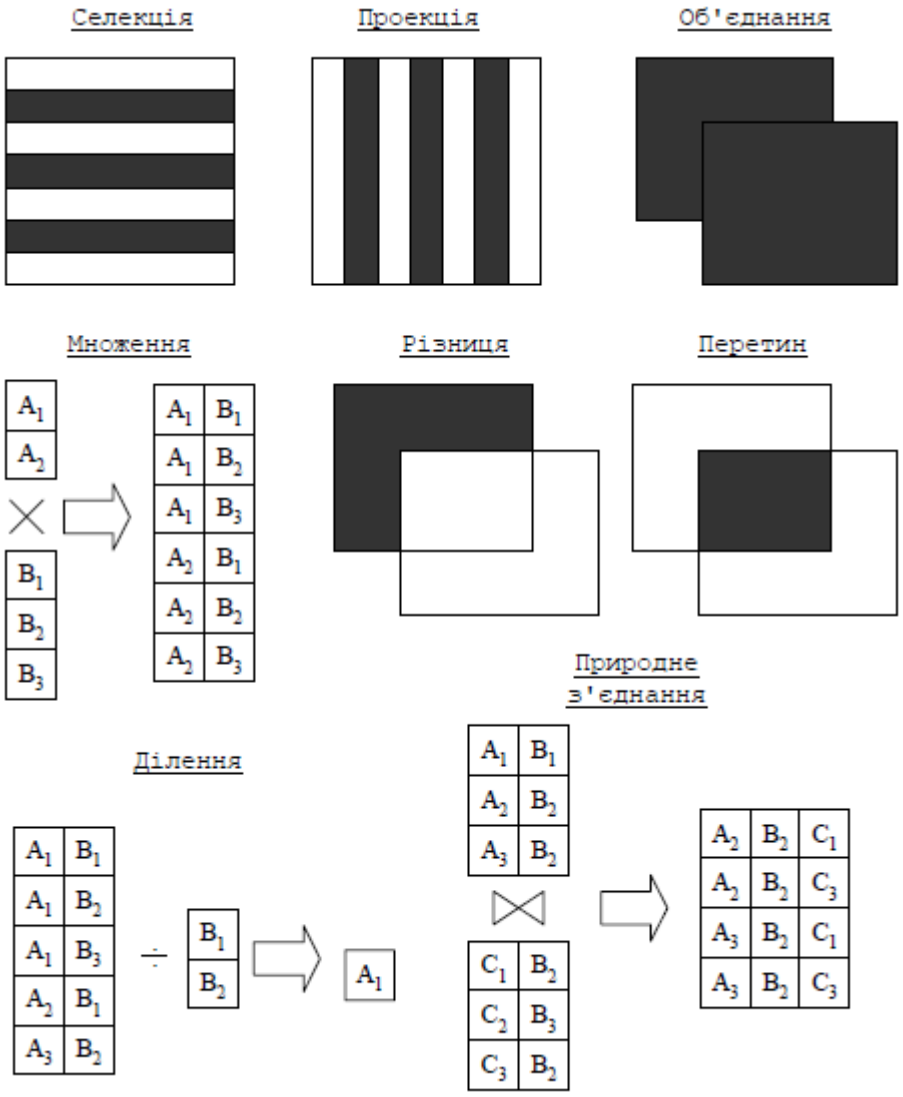


Рис. 2. Основні операції реляційної алгебри

В табл. 4 подані основні і додаткові операції реляційної алгебри.

Таблиця 4

Реляційні операції

Операція	Позначення	Зміст
Вибірка	$\sigma_{\text{предикат}}(R)$	Визначення відношення, яке вміщує тільки ті кортежі з відношення R, які задовольняють заданому предикату
Проекція	$\Pi_{a1 \ldots an}(R)$	Визначення відношення, яке вміщує вертикальну підмножину відношення R, що утворюється шляхом отримання значень вказаних

		атрибутів і виключення з результату рядків-дублікатів
Об'єднання	$R \cup S$	Визначення відношення, яке вміщує всі кортежі, що належать R або S, при цьому виключаються з результату всі дублікати кортежів. Відношення R і S повинні бути сумісні за об'єднанням
Перетин	$R \cap S$	Визначення відношення, яке вміщує всі кортежі, що належать R і S. Відношення R і S повинні бути сумісні за об'єднанням
Різниця	$R - S$	Визначення відношення, яке вміщує всі кортежі, що належать R і відсутні в S. Відношення R і S повинні бути сумісні за об'єднанням
Декартовий добуток	R*S	Визначення відношення, яке є результатом конкатенації кожного кортежа з відношення R з кожним кортежем з відношення S
Тета-з'єднання	$R \bowtie_F S$	Визначення відношення, яке вміщує кортежі з декартового добутку відношень R і S, що задовольняє предикату F
З'єднання по еквівалентності	$R \bowtie_F S$	Визначення відношення, яке вміщує кортежі з декартового добутку відношень R і S, що задовольняє предикату F (предикат виконує порівняння тільки на рівність)
Природне з'єднання	$R \bowtie S$	Визначення відношення, яке отримано з'єднанням по еквівалентності двох

		відношень R і S, що виконано по всім спільним атрибутам *x*, з результатів якого вилучається по одному екземпляру кожного спільного атрибута
Ліве зовнішнє з'єднання	$R \supset \triangleright S$	Визначення відношення, для якого кортежі відношення R, які не мають співпадаючих значень в спільних стовпцях відношення S, також включаються в результуюче відношення
Напівз'єднання	$R \lhd_F S$	Визначення відношення, яке вміщує ті кортежі відношення R, які входять в з'єднання відношень R і S
Ділення	$R \div S$	Визначення відношення, яке вміщує ті кортежі відношення R, які визначені на атрибуті C, що відповідає комбінації всіх кортежів відношення S, де C − множина атрибутів, які є у відношенні R, але відсутні у відношенні S

Приклад. Задані два відношення *Студент* і *Дисципліна*.

Студент

Прізвище	Курс	Група	Спеціальність
Бойко	3	1	ІТП
Левченко	4	2	ІУСТ

Дисципліна

Назва	Курс	Спеціальність	Викладач	Семестр
Бази даних	3	ІТП	Петренко	5
Системний аналіз	4	ІУСТ	Гавриш	7

1. Визначити всіх студентів спеціальності ІУСТ.

$\Pi_{\text{прізвище}}(\sigma_{\text{спеціальність="ІУСТ"}}(\textit{Студент}))$

2. Визначити всіх студентів, для яких у 7 семестрі викладач Гавриш проводить заняття.

$\Pi_{\text{прізвище}}(\sigma_{\text{семестр=7}\wedge\text{викладач="Гавриш"}}(\textit{Студент} \bowtie \text{Дисципліна}))$

$\Pi_{\text{прізвище}}(\textit{Студент} \bowtie (\sigma_{\text{семестр=7}\wedge\text{викладач="Гавриш"}} \text{Дисципліна}))$

4. Обчислення кортежів.

Будь-який вираз обчислення кортежів може бути представлений у такому вигляді:

$$\{ t / f(t) \}$$

де t – єдина вільна змінна – кортеж, яка позначає кортеж фіксованої довжини; $f(t)$ – деякий предикат над змінною t.

Формули в реляційному обчисленні кортежів будують з атомів і сукупності операторів (арифметичних і логічних). Вираз в реляційному обчисленні кортежів будують над множиною відношень. Типи можливих атомів подані в табл. 5.

Для побудови формули (запиту) використовуються логічні зв'язки ($\wedge,\vee,\neg$), а також квантори загальності $\forall x$ і квантори існування $\exists x$. Квантори зв'язують певні змінні. Зв'язана змінна відповідає локальній змінній у мові програмування, а вільна змінна (змінна, яка не зв'язана кванторами) відповідає глобальній змінній.

Таблиця 5

Правила побудови атомів формул

Номер типу атому	Правила побудови атомів
1	$R(t)$, де R − ім'я відношення; цей атом означає, що t є кортеж у

	відношенні R
2	($s[i]\theta u[j]$), де s і u − змінні кортежі; θ − арифметичний оператор відношення; i і j − номери або імена компонентів (стовпчиків) у відповідних кортежах; $s[i]$ − позначення i-го компонента в кортежі-змінній s; $u[j]$ − позначення j-го компонента в кортежі-змінній u
3	($s[i]\theta a$) або ($a\theta s[i]$), де а − константа

Формули будуються за певними правилами. Правила побудови формул наведені в табл. 6.

Таблиця 3.6

Правила побудови формул

Номер правила	Правила побудови формул
1	Кожен атом є формула
2	Якщо f_1 і f_2 − формули, то вирази: – $f_1 \wedge f_2$; – $f_1 \vee f_2$; – $\neg f_1$; також є формулами
3	Якщо f − формула, то вирази: – $\forall t\, f(t)$; – $\exists t\, f(t)$; також є формулами
4	Якщо f − формула, то (f) – також є формулою
5	Ніщо інше не є формулою

Приклад.

1. Вираз $\{t/R_1 \wedge R_2\}$ означає, що в якості формули виступає запис $R_1 \wedge R_2$ і що тут визначається множина кортежів, яка одночасно належить відношенням R_1 і R_2. Цей вираз є еквівалентним до виразу реляційної алгебри $R_1 \cap R_2$.

2. Визначити всіх студентів спеціальності ІУСТ.

{t(*прізвище*)/$\exists t$*Студент*(t)∧t(*спеціальність*)="ІУСТ"}

3. Визначити всіх студентів, для яких у 7 семестрі викладач Гавриш проводить заняття.

{*t*(*прізвище*) / ∃*t* *Студент*(*t*) ∧ ∃ *s* *Дисципліна*(*s*) ∧

s(*семестр*)=7∧ *s*(*викладач*)="Гавриш"∧

t(*курс*, *спеціальність*) =

s(*курс*, *спеціальність*) }

5. Обчислення доменів.

Будь-який вираз обчислення доменів може бути представлений у такому вигляді:

{ $x_1,x_2,\ldots,x_n$ / f($x_1,x_2,\ldots,x_n$) }

де *f* − формула; $x_1,x_2,\ldots,x_n$ − вільні змінні

В обчисленні доменів не існує змінних кортежів. Замість них вводяться змінні на доменах. У всіх інших випадках реляційне обчислення зі змінними на доменах будується так само, як і реляційне обчислення зі змінними на кортежах.

Формули в реляційному обчисленні доменів будують з атомів і сукупності операторів (арифметичних і логічних). Вираз в реляційному обчисленні доменів будують над множиною відношень. Типи можливих атомів подані в табл. 7.

Для побудови формули (запиту) використовуються логічні зв'язки (∧, ∨, ¬), а також квантори загальності ∀*x* і квантори існування ∃*x*.

Таблиця 7

Правила побудови атомів

Номер типу атому	Правила побудови атомів
1	$R(x_1,x_2,\ldots,x_n)$, де *R* − *n*-арне відношення; x_i − константа або змінна на деякому домені. Атом $R(x1,x2,\ldots,xn)$ вказує на те, що значення тих *xi*, які є змінними, повинні бути вибрані так, щоби $(x1,x2,\ldots,xn)$ було кортежем відношення *R*
2	$(x\theta y)$, де *x* і *y* − константи або змінні на деякому домені; θ −

	арифметичний оператор відношення; Атом ($x\theta y$) вказує на те, що x і y являють собою значення, при яких істинно ($x\theta y$)

Приклад.

1. Визначити всіх студентів спеціальності ІУСТ.

{*прізвище*/∃*x* ∃*y* *Студент*(*прізвище*,*x*,*y*, "ІУСТ")}

2. Визначити всіх студентів, для яких у 7 семестрі

викладач Гавриш проводить заняття.

{*прізвище*/∃*x* ∃*y* ∃*z* *Студент*(*прізвище*,*x*,*y*,*z*)∧

∃*s* *Дисципліна*(*s*,*x*,*z*,"Гавриш",7)}

Контрольні запитання

1. Які головні переваги реляційної моделі?
2. Дати визначення термінів: відношення, схема відношення, кортеж, ключ.
3. Які види ключів існують і навіщо вони потрібні?
4. Що таке домени і навіщо вони потрібні?
5. Що таке цілісність БД і як вона підтримується?
6. Що таке логічна і фізична цілісність БД?
7. Що таке посилкова цілісність і як вона підтримується?
8. Перелічити правила вилучення і оновлення даних у зв'язаних відношеннях.
9. Навести приклади обмежень значень і структурних обмежень.
10. Назвати основні операції реляційної алгебри.
11. Назвати додаткові операції реляційної алгебри, навести приклади.
12. Охарактеризувати варіанти реляційного обчислення.
13. Що таке обчислення кортежів?
14. Що таке обчислення доменів?
15. Перелічити правила побудови атомів в обчисленні кортежів і в обчисленні доменів.

16. Перелічити правила побудови формул в обчисленні кортежів і в обчисленні доменів.

17. Порівняти можливості реляційної алгебри і реляційного обчислення.

1. РЕЛЯЦІЙНА АЛГЕБРА

До складу *реляційної моделі даних*, крім структури даних, мають входити операції маніпулювання даними. З усіх таких операцій складається *мова запитів*. Найбільш відомими мовами запитів у реляційній моделі є реляційна алгебра та реляційне числення.

У класичному розумінні алгебра визначається як пара, що складається з основної множини і множини операцій (сигнатури), при цьому аргументи й результат кожної операції належать основній множині.

Реляційна алгебра є алгеброю в строгому класичному розумінні її визначення. Елементами основної множини є реляційні відношення. У зв'язку з цим операції алгебри можуть вкладатися одна в одну, тобто аргументом певної операції може бути результат виконання іншої операції. Це дає можливість записувати запити довільного рівня складності у вигляді виразів, що містять вкладені одна в одну операції.

Сигнатура реляційної алгебри Кодда складається з восьми операцій. Перш ніж детально розглянути ці операції, введемо поняття сумісності реляційних відношень. Це поняття є необхідним, оскільки деякі операції (а саме: теоретико-множинні операції об'єднання, перетину та різниці) визначені лише для сумісних реляційних відношень.

Реляційні відношення $R_1(A_1,\ldots A_n)$ і $R_2(B_1,\ldots B_k)$ називаються *сумісними*, якщо:

1) у них однакова кількість атрибутів, тобто $k=n$;

2) можна встановити взаємно однозначну відповідність між доменами атрибутів першої та другої реляцій, тобто існує таке бієктивне відображення

$S:\{1,\ldots,k\}\rightarrow\{1,\ldots,k\}$

Що $N(A_i)=N(B_{S(i)}), i=1,\ldots k$, тобто домени зіставлених атрибутів однакові.

Для зручності вважатимемо, що зіставлені атрибути сумісних відношень повинні мати однакові імена.

Дамо також означення кількох властивостей бінарних операцій:

- операція ϕ є комутативною, якщо $A\ \phi B=B\ \phi A$;
- операція ϕ є асоціативною, якщо $(A\ \phi B)\ \phi C=A\ \phi(B\ \phi C)$;
- операція ϕ є дистрибутивною з операцією θ, якщо $A\ \phi(B\ \theta C)=(A\ \phi B)\ \theta(A\ \phi C)$.

Даючи означення бінарним операціям реляційної алгебри, ми будемо вказувати, які з цих властивостей вони мають.

Оскільки різні відношення можуть містити атрибути з однаковими іменами, то під час виконання бінарних операцій у кінцевому відношенні можуть повторюватися імена атрибутів. Для забезпечення унікальності імен атрибутів вони уточняються іменами відповідних відношень згідно з таким синтаксисом: < ім'я відношення >.< ім'я атрибута >.

Під час розгляду операцій реляційної алгебри атрибути позначатимемо великими літерами з початку латинського алфавіту: *A, B, ...,* а множини атрибутів – великими літерами з середини латинського алфавіту: *L, M, ….*

1.1. Операції над множинами.

Об'єднання. Нехай L – певна множина атрибутів. *Об'єднанням* сумісних реляційних відношень R_1 і R_2 зі схемами $R_1(L)$ і $R_2(L)$ (позначається як $R_1 \cup R_2$) називається таке реляційне відношення R зі схемою $R(L)$, що містить кортежі обох поєднуваних відношень, але без повторень (рис. 1):

$$R(L)=R_1(L) \cup R_2(L)=\{r \mid r \in R_1 \vee r \in R_2\}$$

Операція комутативна, асоціативна й дистрибутивна щодо перетину.

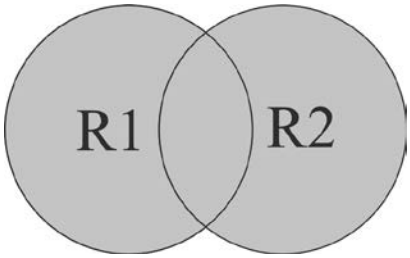


Рис. 1. Операція об'єднання ($R_1 \cup R_2$)

Приклад:

R_1

A	B
a_1	b_1
a_1	b_2
a_2	b_3

R_2

A	B
a_1	b_1
a_2	b_1

$R = R_1 \cup R_2$

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_2	b_3

Перетин. Припустимо, що L – певна множина атрибутів. *Перетином* сумісних реляційних відношень R_1 і R_2 зі схемами $R_1(L)$ і $R_2(L)$ (позначається як $R_1 \cap R_2$) називається таке реляційне відношення R зі схемою $R(L)$, яке містить кортежі, що входять до складу обох операндів (рис. 2):

$$R(L)= R_1(L) \cap R_2(L)=\{r \mid r \in R_1 \ \& \ r \in R_2\}$$

Операція комутативна, асоціативна й дистрибутивна щодо об'єднання.

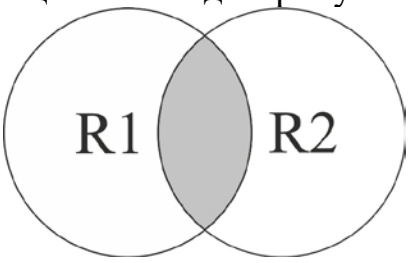


Рис. 2. Операція перетину ($R_1 \cap R_2$)

Приклад:

R_1

A	B
a_1	b_1
a_1	b_2
a_2	b_3

R_2

A	B
a_1	b_1
a_2	b_1

$R_1 \cap R_2$

A	B
a_1	b_1

Різниця. Нехай L – певна множина атрибутів. *Різницею* сумісних реляційних відношень R_1 і R_2 зі схемами $R_1(L)$ і $R_2(L)$ (позначається як R_1-R_2) називається реляційне відношення R зі схемою $R(L)$, що містить ті кортежі з першого операнда R_1, яких немає у другому операнді R_2 (рис. 3):

$R(L)= R_1(L)- R_2(L)=\{r \mid r \in R_1 \ \& \ r \notin R_2\}$

Операція не комутативна, не асоціативна й не дистрибутивна з іншими операціями.

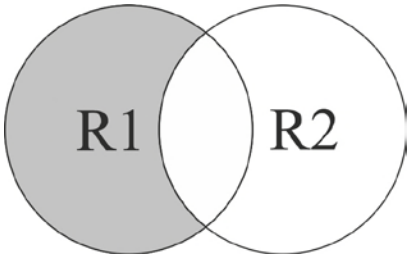


Рис. 3. Операція різниці (R_1-R_2)

Приклад:

R_1

A	B
a_1	b_1
a_1	b_2
a_2	b_3

R_2

A	B
a_1	b_1
a_2	b_1

R_1-R_2

A	B
a_1	b_2
a_2	b_3

Зауважимо, що $R \cap S = R-(R-S)$.

Зазначимо деякі особливості теоретико-множинних операцій:

- у реляційній алгебрі, на відміну від алгебри множин, не використовується операція доповнення, оскільки певні домени можуть бути нескінченними або містити дуже багато значень і в результаті операції доповнення можна отримати або нескінченне відношення, або відношення з дуже великою кількістю кортежів;
- вимога сумісності операндів зумовлена тим, що без цього обмеження результатом теоретико-множинних операцій могли б бути різноструктурні кортежі, а не реляційні відношення.

1.2. Спеціальні реляційні оператори

Розглянемо операції, які визначені лише в реляційній алгебрі.

Проекція. *Проекцією* реляційного відношення R зі схемою $R(A_1,\ldots, A_k)$ за атрибутами $A_{i1},\ldots, A_{in}$, де $\{A_{i1},\ldots, A_{in}\} \subset \{A_1,\ldots, A_k\}$, що позначається $R[A_{i1},\ldots, A_{in}]$, називається таке відношення S зі схемою $S(A_{i1},\ldots, A_{in})$, кортежі якого отримані з кортежів відношення R шляхом видалення значень, що не належать атрибутам, за якими виконується проекція. При цьому в кінцевому відношенні повторні екземпляри кортежів видаляються. Операція проекції є фільтром по атрибутах, тобто деякий вертикальний фільтр.

Якщо r є кортежем відношення R, то записом $r[L]$, де L– підмножина атрибутів відношення R, позначимо множину тих елементів кортежу r, що відповідають значенням атрибутів з L. Тоді наведене вище визначення проекції може бути записане у такий спосіб:

$$S= R[A_{i1},\ldots, A_{in}]=\{ r[A_{i1},\ldots, A_{in}] \mid r\in R\}.$$

Операція проекції також записується як $\pi_{Ai1,\ldots, Ain}(R)$.

З теоретичної точки зору операція проекції не є „чистою”, оскільки список атрибутів не належить основній множині (тобто не є реляційним відношенням), а тому не може бути операндом. За „чистого” теоретичного підходу операцію проекції слід вважати бінарною, а на практиці вона розглядається як унарна з параметрами.

Такі ж самі проблеми мають місце в усіх операціях, де операндами є списки атрибутів.

Приклад:

R

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_2	b_3	c_1
a_2	b_4	c_2

$R[A,C]$

A	C
a_1	c_1
a_2	c_1
a_2	c_2

Декартовий добуток. Декартовим добутком реляційних відношень R і S зі схемами $R(A_1, A_2,\ldots A_n)$ та $S(B_1, B_2,\ldots B_m)$ відповідно, що позначається $R\times S$, називається відношення Q зі схемою $Q(A_1, A_2,\ldots A_n, B_1, B_2,\ldots B_m)$, яке містить усі можливі з’єднання кортежів відношення R з кортежами відношення S:

$$Q = R\times S = \{(r, s) \mid r\in R \ \&\ s\in S\}.$$

Операція комутативна й асоціативна.

Приклад:

R_1

A	B
a_1	b_1
a_1	b_2
a_2	b_3

R_2

C	D
c_1	d_1
c_2	d_1

$R=R_1 \times R_2$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_1
a_1	b_2	c_1	d_1
a_1	b_2	c_2	d_1
a_2	b_3	c_1	d_1
a_2	b_3	c_2	d_1

Обмеження (селекція). Спочатку дамо означення θ-порівнянності атрибутів. Нехай θ є одним з операторів порівняння: $=$, $\neq$, $\geq$, $>$, $\leq$, $<$ (набір операторів можна розширити). Атрибути A і B одного й того самого чи різних

відношень називаються *θ-порівнянними*, якщо для будь-яких значень $a \in A$ і $b \in B$ результат операції $a \theta b$ є визначеним (істинним або хибним). Інакше кажучи, ця операція визначена на відповідних атрибутах. Набори атрибутів $L=(A_1,\ldots, A_k)$ та $M=(B_1,\ldots, B_n)$ називаються *θ*-порівнянними, якщо $k=n$ і A_i *θ*-порівнянне з B_i ($i=1,2,\ldots k$). Тоді вираз $L\ \theta M$ розуміють так:

$$L\ \theta M\ =(A_1\ \theta B_1)\&\ldots\&\ (A_k\ \theta B_k).$$

Тепер дамо означення операції обмеження.

Нехай L і M – набори *θ*-порівнянних атрибутів схеми відношення R. Тоді *обмеженням* реляційного відношення R за умовою $L\ \theta M$, що позначається $R[L\ \theta M]$, називається реляційне відношення, кортежі якого відповідають умові $L\ \theta\ M$:

$$S= R[L\ \theta M]=\{r \mid r \in R\ \&\ r[L]\ \theta r[M]\}$$

Множина M може складатися як з атрибутів, так і з констант.

Операція обмеження також записується як $\sigma_{L\ \theta M}(R)$.

Приклад:

R

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_2	b_3	c_1
a_2	b_4	c_2

$R[A=a_2]$

A	B	C
a_2	b_3	c_1
a_2	b_4	c_2

Крім того, умова порівняння для *θ*-вибірки може містити довільне число простих порівнянь, з'єднаних логічними зв'язками (AND(∧, &), OR(∨)). При необхідності можуть використовуватися круглі дужки.

З'єднання. Припустимо, що відношення R має схему $R(L, M)$, а відношення S– схему $S(N, P)$. Нехай множини атрибутів M і N – *θ*-порівнянні. Тоді *з'єднанням*, або *θ-з'єднанням*, відношень R і S за умовою $M\ \theta\ N$, що позначається як $R[M\ \theta\ N]S$, називається відношення Q зі схемою $Q(L,M,N,P)$, кортежі якого можна отримати з'єднанням тих кортежів відношень R і S, на яких виконується умова $M\ \theta N$:

$$Q= R[M\ \theta N]S=\{(r, s) \mid r \in R\ \&\ s \in S\ \&\ r[M]\ \theta s[N]\}.$$

Під час з'єднання атрибути, за якими виконується така операція, повторюються в кінцевому реляційному відношенні. Операція комутативна й асоціативна.

Іноді операція з'єднання позначається як $R \bowtie_F S$, де F– умова з'єднання.

З'єднання за умовою рівності називається *еквіз'єднанням.* З'єднання за умовою рівності, коли один з порівнюваних атрибутів (чи група порівнюваних атрибутів) видаляється з кінцевого відношення, називається *природним з'єднанням*; на його позначення використовується символ „*". Наприклад, якщо задані відношення $R(A,B,C,D)$ і $S(C,D,E)$, то в результаті виконання операції $Q=R*S$ отримаємо реляційне відношення $Q(A,B,C,D,E)$.

Серед операцій θ-з'єднання виділяють операцію *напівз'єднання,* за якої з результату видаляються всі атрибути одного з відношень, що з'єднуються. Вона записується як $R[M\theta \mathrm{N}\rangle S$ і формально визначається так:

$$Q = R[M\theta \mathrm{N}\rangle S = \{r \mid r \in R \ \& \ s \in S \ \& \ r[M] \ \theta s[N]\}.$$

Операція напівз'єднання не розширює можливостей реляційної алгебри, оскільки вона виражається через з'єднання і проекцію в такий спосіб:

$$R[M\theta \mathrm{N}\rangle S = (\ R[M \ \theta N]S)[L,M]$$

Приклад виконання операції природного з'єднання:

R

A	*B*	*C*	*D*
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_1
a_1	b_2	c_1	d_1
a_2	b_3	c_1	d_1
a_2	b_4	c_2	d_3

S

C	*D*	*E*
c_1	d_1	e_2
c_2	d_1	e_3
c_2	d_1	e_1

R[*C*,*D*=*C*,*D*]*S*

A	*B*	*C*	*D*	*C*	*D*	*E*
a_1	b_1	c_1	d_1	c_1	d_1	e_2
a_1	b_2	c_1	d_1	c_1	d_1	e_2
a_2	b_3	c_1	d_1	c_1	d_1	e_2
a_1	b_1	c_2	d_1	c_2	d_1	e_3
a_1	b_2	c_2	d_1	c_2	d_1	e_3
a_1	b_1	c_2	d_1	c_2	d_1	e_1
a_1	b_2	c_2	d_1	c_2	d_1	e_1

*R*S*

A	*B*	*C*	*D*	*E*
a_1	b_1	c_1	d_1	e_2
a_1	b_2	c_1	d_1	e_2
a_2	b_3	c_1	d_1	e_2
a_1	b_1	c_2	d_1	e_3
a_1	b_2	c_2	d_1	e_3
a_1	b_1	c_2	d_1	e_1
a_1	b_2	c_2	d_1	e_1

Крім операцій природного з'єднання та тета з'єднання є ще операція зовнішнього з'єднання. Часто буває таке, що при з'єднанні двох відношень кортеж з одного відношення не знаходить відповідного кортежу в іншому відношенні. Інакше кажучи, в атрибутах, за якими відбувається операція з'єднання, виявляються незбіжні значення. Може знадобитися, щоб кортеж з одного або двох відношень був поданий у результаті з'єднання, навіть якщо у відповідних атрибутах немає співпадаючих значень. Ця мета може бути досягнута за допомогою операції зовнішнього з'єднання.

Приклад виконання операції лівого зовнішнього з'єднання:

R

A	*B*	*C*
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_2	c_3
a_2	b_3	c_1
a_2	b_4	c_3

S

C	*D*	*E*
c_1	d_1	e_2
c_2	d_1	e_3
c_2	d_1	e_1

$R \supset\!\triangleleft S$

A	*B*	*C*	*D*	*E*
a_1	b_1	c_1	d_1	e_2
a_1	b_1	c_2	d_1	e_3
a_2	b_3	c_1	d_1	e_2
a_1	b_1	c_2	d_1	e_1
a_1	b_2	c_3	*null*	*null*
a_2	b_3	c_1	d_1	e_2
a_2	b_4	c_3	*null*	*null*

Відсутні значення у результуючому відношенні позначаються за допомогою визначника *NULL*. Перевагою операції зовнішнього з'єднання є зберігання вихідної інформації: кортежі, які втрачаються при використанні

інших типів з’єднань. Наприклад лівим зовнішнім з'єднанням ($R \supset\!\triangleleft S$) називається операція, коли кортежі відношення R, які не мають співпадаючих значень у спільних стовпцях з відношенням S, також включаються в результуюче відношення.

Ділення. Нехай задано відношення зі схемою $R(M,N)$. *Образом реляційного відношення R за кортежем* $t_1 \in R[M]$ називається така множина кортежів $t_2 \in R[N]$, для яких зчеплення (t_1, t_2) належить відношенню R. Образ R за кортежем t_1 позначається $I_R(t_1)$ і формально визначається у такий спосіб:

$$I_R(t_1)=\{\ t_2 \mid t_2 \in R[N] \ \&\ (t_1, t_2) \in R\}.$$

Приклад:

R

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_3	c_2
a_2	b_1	c_4

$I_R(a_1)$

B	C
b_1	c_1
b_1	c_2
b_3	c_2

$I_R(a_2)$

B	C
b_1	c_4

$I_R(a_1,b_1)$

C
c_1
c_2

$I_R(c_2)$

A	B
a_1	b_1
a_1	b_3

Нехай задано відношення R і S зі схемами $R(M,N)$ та $S(K,L)$, для яких проекції $R[N]$ та $S[K]$ є сумісними. *Діленням* відношення R на відношення S за наборами атрибутів N і K (позначається $R[N \div K]S$) називається операція, результатом якої є відношення Q зі схемою $Q(M)$, що складається з таких кортежів $t \in R[M]$, образи $I_R(t)$ яких містять усі кортежі проекції $S[K]$, тобто:

$$Q= R[N \div K]S=\{t \mid t \in R[M] \& \ I_R(t) \supseteq S[K]\}$$

Можна сказати, що операція ділення задається через інші операції алгебри в такий спосіб:

$$R[N \div K]S= R[M]-((\ R[M] \times S[K])-R)[M]$$

Операція не комутативна й не асоціативна.

Приклад:

R

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_3	c_2
a_2	b_1	c_4

S

C	D
c_1	d_1
c_1	d_2
c_2	d_1
c_2	d_3

$S[C]$

C
c_1
c_2

$R[\mathrm{C} \div C]S$

A	B
a_1	b_1

1.3. Задачі на застосування реляційної алгебри

Нехай задано реляційну схему бази даних вищого навчального закладу (рис. 4):

а)

STUDENT (Студент)	**LECTURER (Викладач)**	**SUBJECT (Предмет навчання)**	**UNIVERSITY (Університети)**	**EXAM_MARKS (Екзаменаційні оцінки)**	**SUBJ_LECT (Навчальні дисципліни викладачів)**
STUDENT_ID (PK) SURNAME NAME STIPEND YEAR_LEARNING CITY BIRTHDAY UNIV_ID_ID (FK)	LECTURER_ID (PK) SURNAME NAME CITY UNIV_ID_ID (FK)	SUBJ_ID (PK) SUBJ_NAME HOUR SEMESTER	UNIV_ID (PK), UNIV_NAME RATING CITY	EXAM_ID (PK) STUDENT_ID_ID (FK) SUBJ_ID_ID (FK) MARK EXAM_DATE	LECTURER_ID_ID (PK) (FK) SUBJ_ID_ID (PK) (FK)

б)

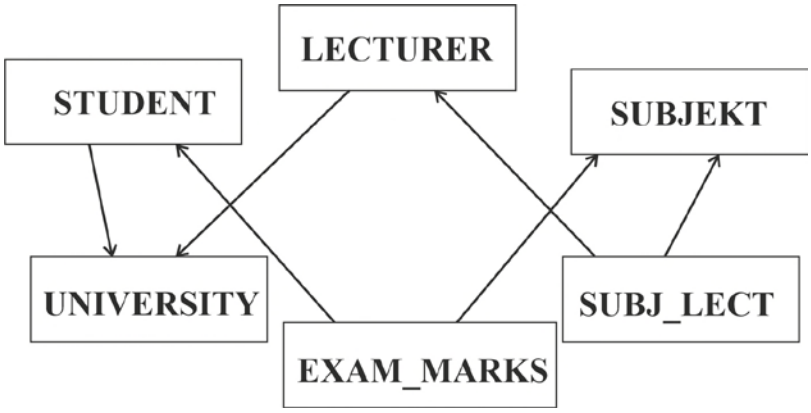


Рис. 4. Реляційна схема бази даних вищого навчального закладу:
а) відношення; б) зв’язки між відношеннями.

Приклади згрупуємо за операціями, застосування яких у них демонструються.

Проекція

1. Отримати прізвища усіх викладачів.

В даному завданні точно вказано, що потрібно отримати відношення, що включає атрибут SURNAME. *Цей атрибут входить в відношення* LECTURER*, отже, треба буде застосувати операцію проекції до відношенню, яке містить необхідні кортежі.*

LECTURER [SURNAME]

$\pi_{SURNAME}$(LECTURER)

2. Отримати прізвища усіх викладачів і місто, де вони проживають.

В даному завданні потрібно отримати відношення, що включає два атрибути SURNAME *і* CITY *відношення* LECTURER.

LECTURER [SURNAME, CITY]

$\pi_{SURNAME, CITY}$(LECTURER)

3. Отримати прізвища усіх студентів їх день народження та рік навчання.

STUDENT [SURNAME, BIRTHDAY, YEAR_LEARNING]

$\pi_{\text{SURNAME, BIRTHDAY, KURS}}(\text{STUDENT})$

4. Отримати назви всіх університетів та їх рейтинг.

UNIVERSITY [UNIV_NAME, RATING]

$\pi_{\text{UNIV\_NAME, RATING}}$ (UNIVERSITY)

Обмеження

5. Отримати всі відомості про студентів першого року навчання.

У даній задачі потрібно отримати всі дані, тобто отримати весь кортеж відношення STUDENT. Але в результуюче відношення повинні потрапити тільки ті кортежі, що задовольняють умові (YEAR_LEARNING =1). В даному випадку не потрібно виконувати операцію проекції.

STUDENT [YEAR_LEARNING =1]

$\sigma_{\text{YEAR\_LEARNING =1}}(\text{STUDENT})$

6. Отримати всі відомості про предмети, які вивчаються в 5 семестрі і на вивчення яких відводиться більше 100 годин.

SUBJECT [(SEMESTER=5) AND (HOUR>100)]

$\sigma_{\text{(SEMESTER=5) AND (HOUR>100)}}(\text{SUBJECT})$

7. Отримати всі відомості про університети, які знаходяться в місті 'Lviv' або 'Lutsk'.

UNIVERSITY [(CITY='Lviv') OR (CITY='Lutsk')]

$\sigma_{\text{(CITY='Lviv') OR (CITY='Lutsk')}}(\text{UNIVERSITY})$

Використання констант у наведених прикладах операції θ-обмеження є відступом від теоретичної „чистоти" реляційної моделі, але розробники мов на основі реляційної алгебри використовували подібні відступи задля зручності користувача та практичної ефективності. Для того, щоб написати теоретично „чистий" варіант даних запитів, потрібно використати допоміжну таблицю з єдиним кортежем, який містить значення константи і виконати з'єднання використаної в запиті таблиці і допоміжної таблиці.

Обмеження з проекцією

8. Отримати прізвища всіх студентів першого року навчання.

В даній задачі потрібно отримати всі відомості про студентів першого року навчання (STUDENT [YEAR_LEARNING =1]), і в отриманому відношенні виконати операцію проекції на атрибут SURNAME.

(STUDENT [YEAR_LEARNING =1])[SURNAME]

$\pi_{\text{SURNAME}}(\sigma_{\text{YEAR\_LEARNING =1}}(\text{STUDENT}))$

9. Отримати прізвища та імена усіх викладачів які проживають в місті 'Lviv'.

(LECTURER[CITY='Lviv'])[SURNAME, NAME]

$\pi_{SURNAME,\ NAME}\ (\sigma_{CITY='Lviv'}\ (LECTURER))$

10. Отримати прізвища усіх студентів першого року навчання та їх день народження.

(STUDENT [YEAR_LEARNING =1])[SURNAME, BIRTHDAY]

$\pi_{SURNAME,\ BIRTHDAY}(\sigma_{YEAR\_LEARNING\ =1}\ (STUDENT))$

11. Отримати назви всіх університетів міста 'Lviv', якщо його рейтинг знаходиться в межах (10–40).

UNIVERSITY[(CITY='Lviv') AND (RATING>10) AND (RATING<40)]
[UNIV_NAME]

$\pi_{UNIV\_NAME}\ (\sigma_{(CITY='Lviv')\ AND\ (RATING>10)\ AND\ (RATING<40)}\ (UNIVERSITY)$

З'єднання

12. Отримати всі відомості про студентів, та всі відомості про їх оцінки (всі дані з таблиці STUDENT та всі дані з таблиці EXAM_MARKS)

Особливістю цього запиту порівняно з попередніми є те, що шукані значення розташовані в різних таблицях. Тому потрібно з'єднати два відношення STUDENT *і* EXAM_MARKS *при умові, що значення коду студента таблиці* STUDENT *буде рівним значенню коду студента таблиці* EXAM_MARKS [STUDENT_ID=STUDENT_ID_ID].

STUDENT[STUDENT_ID=STUDENT_ID_ID] EXAM_MARKS

$STUDENT \bowtie_{STUDENT\_ID=STUDENT\_ID\_ID} EXAM\_MARKS$

З'єднання, обмеження, проекція

13. Отримати прізвища студентів та їх оцінки.

В даній задачі дані про прізвища студентів знаходяться в таблиці STUDENT, а дані про оцінки в таблиці EXAM_MARKS, тому потрібно виконати операцію з'єднання, як у задачі 12, і в отриманому відношенні виконати проекцію на атрибути [SURNAME, MARK].

(STUDENT[STUDENT_ID=STUDENT_ID_ID] EXAM_MARKS) [SURNAME, MARK]

$\pi_{SURNAME,\ MARK}(STUDENT \bowtie_{STUDENT\_ID=STUDENT\_ID\_ID} EXAM\_MARKS\ )$

14. Отримати прізвища студентів та назви предметів по яких вони здавали екзамени та дати здачі цих екзаменів.

В даній задачі дані отримуються з трьох таблиць: STUDENT, EXAM_MARKS, SUBJECT. Ці таблиці нам потрібно з'єднати. Спочатку з'єднаємо таблиці STUDENT та EXAM_MARKS (STUDENT[STUDENT_ID=STUDENT_ID_ID] EXAM_MARKS), а потім до отриманої таблиці доєднаємо таблицю SUBJECT ((STUDENT[STUDENT_ID=STUDENT_ID_ID] EXAM_MARKS) [SUBJ_ID_ID = SUBJ_ID] SUBJECT). Остання дія, виконати проекцію з отриманого відношення на задані в умові атрибути.

((STUDENT[STUDENT_ID=STUDENT_ID_ID] EXAM_MARKS) [SUBJ_ID_ID = SUBJ_ID] SUBJECT) [SURNAME, EXAM_DATE, SUBJ_NAME]

$\pi_{SURNAME,\ EXAM\_DATE,\ SUBJ\_NAME}$ ((STUDENT $\bowtie_{STUDENT\_ID=STUDENT\_ID\_ID}$ EXAM_MARKS) $\bowtie_{SUBJ_ID_ID\ =\ SUBJ_ID}$SUBJECT

15. Отримати прізвища викладачів, та предмети, які вони викладають

Таблиці LECTURER та SUBJECT поєднані між собою через таблицю SUBJ_LECT, тому участь в запиті будуть приймати три таблиці. Спочатку з'єднаємо таблиці LECTURER та SUBJ_LECT, (LECTURER[LECTURER _ID=LECTURER _ID_ID] SUBJ_LECT) а потім до отриманої таблиці доєднаємо таблицю SUBJECT – ((LECTURER[LECTURER _ID=LECTURER _ID_ID] SUBJ_LECT) [SUBJ_ID= SUBJ_ID_ID] SUBJECT). Після цього потрібно отримати проекцію на поля SURNAME та SUBJ_NAME.

((LECTURER[LECTURER_ID=LECTURER_ID_ID]SUBJ_LECT) [SUBJ_ID= SUBJ_ID_ID] SUBJECT) [SURNAME, SUBJ_NAME]

$\pi_{SURNAME,\ SUBJ\_NAME}$((LECTURER$\bowtie_{LECTURER_ID=LECTURER_ID_ID}$SUBJ_LECT) $\bowtie_{SUBJ\_ID=\ SUBJ\_ID\_ID}$ SUBJECT)

16. Отримати прізвища студентів та їх оцінки з предмету 'Programming'.

В даному випадку необхідність використання операції з'єднання зумовлена тим, що поля результату і аргумент пошуку і перебувають в різних таблицях. *Дані вибираються з двох таблиць STUDENT та EXAM_MARKS, а умова пошуку ставиться до поля третьої таблиці SUBJECT. Для обчислення цього запиту спочатку здійснюється з'єднання трьох відношень (STUDENT, EXAM_MARKS, SUBJECT) за рівністю первинних і зовнішніх ключів, потім вибираються ті кортежі, які стосуються предмету 'Programming' і нарешті здійснюється проекція за необхідними атрибутами.*

(((STUDENT[STUDENT_ID=STUDENT_ID_ID] EXAM_MARKS) [SUBJ_ID_ID= SUBJ_ID] SUBJECT) [SUBJ_NAME='Programming']) [SURNAME, MARK]

$\pi_{SURNAME,\ MARK}$ ($\sigma_{SUBJ\_NAME='Programming'}$ ((STUDENT $\bowtie_{STUDENT\_ID=STUDENT\_ID\_ID}$ EXAM_MARKS) $\bowtie_{SUBJ\_ID\_ID\ =\ SUBJ\_ID}$ SUBJECT))

17. Отримати прізвища студентів, які вивчають предмет 'Programming' у викладача з прізвищем 'Makarchuk'. Також вивести оцінку з даного предмета.

У даному запиті участь приймають п'ять таблиць, які потрібно з'єднати. Крім того в таблицях, що приймають участь у запиті є однойменні стовпці, то, будуючи запит, слід уточнити імена атрибутів іменами відношень.

(((((STUDENT[STUDENT.STUDENT_ID= EXAM_MARKS.STUDENT_ID_ID] EXAM_MARKS) [EXAM_MARKS.SUBJ_ID_ID= SUBJECT.SUBJ_ID] SUBJECT) [SUBJECT.SUBJ_ID = SUBJ_LECT.SUBJ_ID_ID] SUBJ_LECT) [SUBJ_LECT.LECTURER_ID_ID = LECTURER.LECTURER_ID] LECTURER) [SUBJECT.SUBJ_NAME='Programming' AND LECTURER.SURNAME='Makarchuk']) [STUDENT.SURNAME, EXAM_MARKS.MARK]

$\pi_{\text{STUDENT.SURNAME, EXAM\_MARKS.MARK}}(\sigma_{\text{SUBJECT.SUBJ\_NAME='Programming' AND LECTURER.SURNAME='Makarchuk'}}$ ((((STUDENT $\bowtie_{\text{STUDENT.STUDENT\_ID= EXAM\_MARKS.STUDENT\_ID\_ID}}$ EXAM_MARKS) $\bowtie_{\text{EXAM\_MARKS.SUBJ\_ID\_ID = SUBJECT.SUBJ\_ID}}$ SUBJECT) $\bowtie_{\text{SUBJECT.SUBJ\_ID = SUBJ\_LECT.SUBJ\_ID\_ID}}$ SUBJ_LECT) $\bowtie_{\text{SUBJ\_LECT.LECTURER\_ID\_ID = LECTURER.LECTURER\_ID}}$ LECTURER))

Послідовність виконання операцій запиту можна зобразити графічно, як дерево, що його листки позначають відношення, а інші вершини – операції (рис. 5).

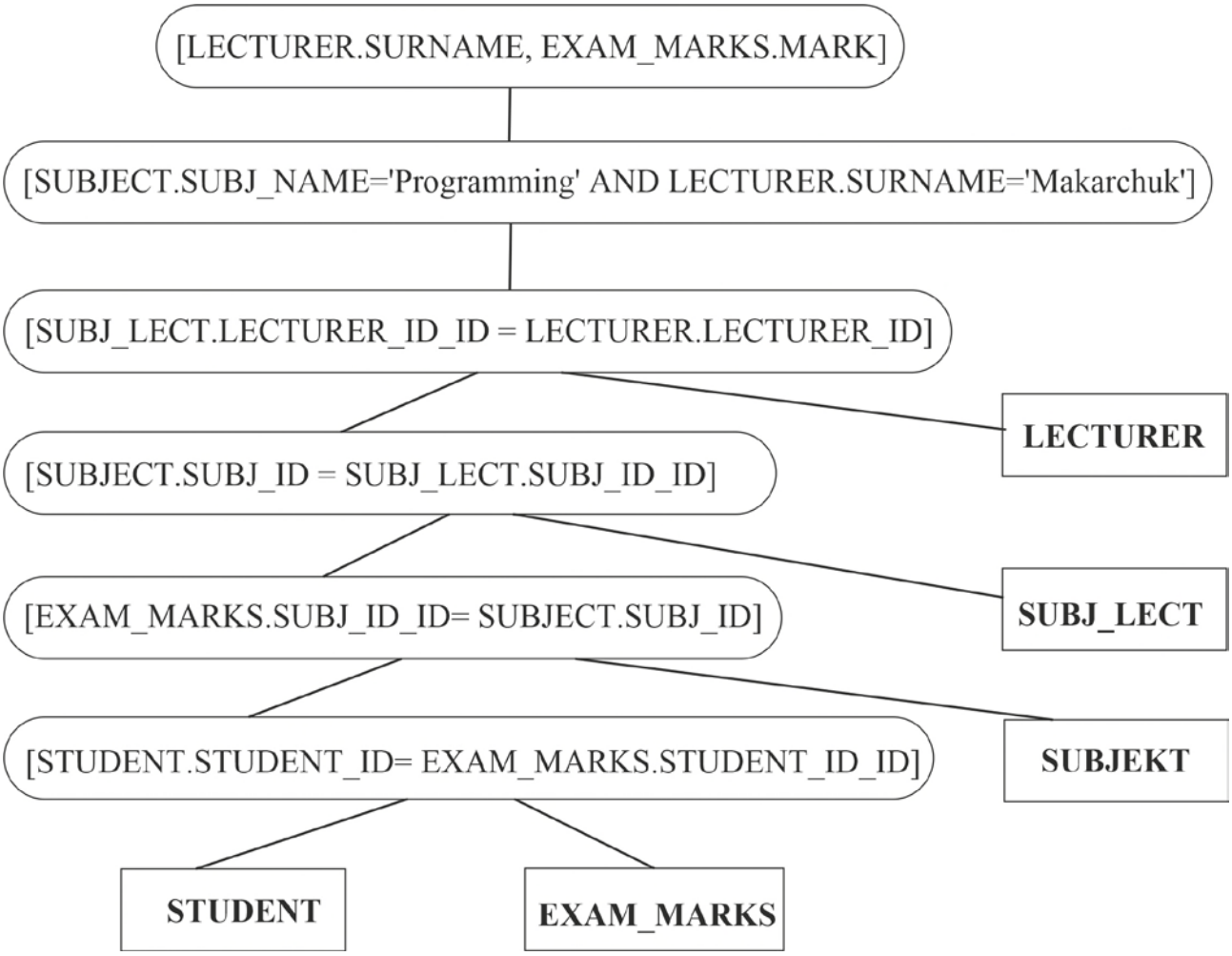


Рис. 5. Схема виконання запиту завдання 17.

В тих випадках, коли виникають проблеми з записом запиту формальною мовою, для зручності можна побудувати схему його виконання.

Теоретико-множинні операції

18. Отримати прізвища та імена усіх студентів і викладачів, які проживають в місті 'Lviv'.

(STUDENT [CITY='Lviv'])[SURNAME, NAME] ∪
(LECTURER[CITY='Lviv'])[SURNAME, NAME]

$\pi_{\text{SURNAME, NAME}}(\sigma_{\text{CITY='Lviv'}}(\text{STUDENT})) \cup \pi_{\text{SURNAME, NAME}}(\sigma_{\text{CITY='Lviv'}}(\text{LECTURER}))$

У першій частині запиту (до символу „∪”) ми знаходимо всіх студенів, які проживають в місті 'Lviv', а в другій частині – отримуємо список викладачів, які проживають в місті 'Lviv'. Операція об’єднання зводить в одну таблицю (без повторень) вміст двох сумісних між собою таблиць-операндів.

19. Отримати міста, де проживають і викладачі і студенти?

В даному завданні ми маємо операцію перетину.

STUDENT [CITY] ∩ LECTURER[CITY]

π_{CITY} (STUDENT) ∩ π_{CITY}(LECTURER)

20. Хто з викладачів не викладає жодної дисципліни?

Цей запит доцільно будувати за методом „від супротивного”, тобто від усієї множини викладачів (точніше їх прізвищ) віднімається множина тих, які викладають хоча б одну дисципліну.

(LECTURER[SURNAME, NAME]-
((LECTURER[LECTURER_ID=LECTURER_ID_ID]SUBJ_LECT)[SUBJ_ID_ID= SUBJ_ID] SUBJECT) [SURNAME, NAME]

$\pi_{SURNAME, NAME}$ (LECTURER)- $\pi_{SURNAME, NAME}$ (((LECTURER
$\bowtie_{LECTURER\_ID=LECTURER\_ID\_ID}$ SUBJ_LECT) $\bowtie_{SUBJ\_ID\ \_ID\ =\ SUBJ\_ID}$ SUBJECT))

Операція ділення

21. Отримати ідентифікаційні номери студентів, які здали всі предмети.

((STUDENT[STUDENT_ID=STUDENT_ID_ID] EXAM_MARKS) [SUBJ_ID_ID= SUBJ_ID] SUBJECT) [STUDENT_ID, SUBJ_ID]
÷ SUBJECT

$\pi_{STUDENT\_ID, SUBJ\_ID}$ ((STUDENT$\bowtie_{STUDENT_ID=STUDENT_ID_ID}$ EXAM_MARKS) ⋈
$_{EXAM\_MARKS.SUBJ\_ID\_ID\ =\ SUBJECT.SUBJ\_ID}$ SUBJECT) ÷ SUBJECT

2. ОПТИМІЗАЦІЯ ОБЧИСЛЕННЯ ВИРАЗІВ РЕЛЯЦІЙНОЇ АЛГЕБРИ

2.1. Властивості операцій реляційної алгебри. Еквівалентні перетворення

Опишемо основні властивості операцій реляційної алгебри, на яких базуються правила еквівалентних перетворень її виразів. Два вирази реляційної алгебри називаються *еквівалентними*, якщо за будь-яких значень реляційних відношень, що входять до їхнього складу, результати обчислення виразів збігаються. Правила еквівалентних перетворень дають змогу вирішувати проблему оптимізації виконання запитів реляційної алгебри.

Наведемо основні властивості операцій реляційної алгебри.

1. Комутативність, асоціативність та дистрибутивність теоретико-множинних операцій об'єднання, перетину і різниці.
2. Ідемпотентність проекцій.

Нехай L і M – множини атрибутів деякого реляційного відношення R. Якщо $L \subseteq M$, то

$\pi_L(\pi_M R) = \pi_L R$.

3. Дистрибутивність проекції з теоретико-множинними операціями, декартовим добутком, з'єднанням, селекцією.

Нехай K– певна підмножина атрибутів реляційного відношення R, P– підмножина атрибутів реляційного відношення S і $N = K \cup P$. Тоді:

а) $\pi_N(R \cup S) = \pi_N R \cup \pi_N S$;

б) $\pi_N(R \cap S) = \pi_N R \cap \pi_N S$;

в) $\pi_N(R - S) = \pi_N R - \pi_N S$;

г) $\pi_N(R \times S) = \pi_K R \times \pi_P S$;

д) $\pi_N(R \bowtie_F S) = \pi_K R \bowtie_F \pi_P S$ (якщо в умові F використовуються лише атрибути з множини N);

е) $\pi_K \sigma_F R = \sigma_F \pi_K R$ (якщо в умові F використовуються лише атрибути з множини K).

4. Ідемпотентність (комутативність) селекцій:

$\sigma_F(\sigma_G(R)) = \sigma_G(\sigma_F(R)) = \sigma_{F\&G}(R)$.

5. Комутативність селекції з декартовим добутком:

а) $\sigma_F(R \times S) = \sigma_F(R) \times S$ (якщо в умові F використовуються лише атрибути з множини R);

б) $\sigma_{F1\&F2}(R \times S) = \sigma_{F2}(\sigma_{F1}(R \times S)) = \sigma_{F1}(R) \times \sigma_{F2}(S)$
(якщо всі атрибути з F_1 містяться у відношенні R і всі атрибути з F_2 міститься у відношенні S);

в) $\sigma_{F1\&F2\&F3}(R \times S) = \sigma_{F3}(\sigma_{F1}(R) \times \sigma_{F2}(S))$ (якщо всі атрибути з F_1 містяться у відношенні R і всі атрибути з F_2 міститься у відношенні S).

6. Комбінування селекції з декартовим добутком та θ-з'єднанням:

а) $\sigma_{K\theta P}(R \times S) = R \bowtie_{K\theta P} S$;

б) $\sigma_G(R \bowtie_F S) = R \bowtie_{G\theta F} S$; (атрибути з G належать відношенню R, атрибути з F – відношенню S).

7. Дистрибутивність селекції з теоретико-множинними операціями.

Нехай атрибути з умови F входять до складу як реляційного відношення R, так і реляційного відношення S. Тоді:

а) $\sigma_F(R \cup S) = \sigma_F(R) \cup \sigma_F(S)$;

б) $\sigma_F(R \cap S) = \sigma_F(R) \cap \sigma_F(S)$;

в) $\sigma_F(R - S) = \sigma_F(R) - \sigma_F(S)$;

8. Комутативність селекції і з'єднання:

а) якщо всі атрибути з логічного виразу G містяться в реляційному відношенні R, то

$\sigma_G(R \bowtie_F S) = \sigma_G(R) \bowtie_F S$;

б) якщо $G = G_1 \& G_2$, всі атрибути з G_1 містяться у реляційному відношенні R і всі атрибути з G_2 – у реляційному відношенні S, то

$\sigma_G(R \bowtie_F S) = \sigma_{G1}(R) \bowtie_F \sigma_{G2}(S)$;

9. Комутативність та асоціативність добутку:

а) $R \times S = S \times R$– комутативність;

б) $R \times (S \times T) = (R \times S) \times T$ – асоціативність.

10. Комутативність та асоціативність з'єднання:

а) $R \bowtie_F S = S \bowtie_F R$ – комутативність;

б) $R \bowtie_F (S \bowtie_F T) = (R \bowtie_F S) \bowtie_F T$ – асоціативність.

11. Дистрибутивність з'єднання з теоретико-множинними операціями:

а) $R \bowtie_F (S \cup T) = (R \bowtie_F S) \cup (R \bowtie_F T)$;

б) $R \bowtie_F (S \cap T) = (R \bowtie_F S) \cap (R \bowtie_F T)$;

в) $R \bowtie_F (S - T) = (R \bowtie_F S) - (R \bowtie_F T)$.

2.2. Використання реляційної алгебри для оптимізації запитів

Описані в п. 2.1. властивості операцій реляційної алгебри дають змогу вирішувати завдання логічної оптимізації алгебраїчних виразів. Під терміном *логічна оптимізація* ми маємо на увазі оптимізацію, що дає можливість прискорити обчислення реляційних виразів незалежно від способів реалізації реляційних відношень. На відміну від алгебри числових виразів, складність виконання формул реляційної алгебри залежить не лише від кількості операцій, але й від розміру операндів.

Розглянемо приклад оптимізаційних перетворень реляційного виразу. Нехай потрібно отримати прізвища студентів першого року навчання, та їх оцінки. (реляційна схема рис. 5), Один з варіантів розв'язку може бути наступним: $\pi_{SURNAME,\ MARK}((\sigma_{STUDENT\_ID=STUDENT\_ID\_ID\&YEAR\_LEARNING\ =1}(STUDENT \times EXAM\_MARKS)$

Можливі шляхи послідовних еквівалентних перетворень, що оптимізують обчислення цього виразу, наведені на рис. 6. Оптимізаційні перетворення здійснюються згідно з властивостями, описаними в п. 2.1.

Наведемо тепер основні правила оптимізації виразів реляційної алгебри.

1. Кожна селекція $\sigma_{F1\&\ldots\&Fn}(E)$ згідно з властивістю 4 подається у вигляді послідовності селекцій $\sigma_{F1}(\ldots(\sigma_{Fn}(E)\ldots))$.
2. Кожна селекція переміщується деревом виразу вниз наскільки це можливо (властивості 3е, 5, 7, 8). У такий спосіб зменшується кардинальність відношень.
3. Розташовані поруч селекції і декартові добутки замінюються з'єднанням, якщо це допускається властивістю 6.
4. Кожна проекція переміщується деревом виразу вниз наскільки це можливо (властивості 2, 3). У такий спосіб зменшується ступінь відношень.
5. Кожен каскад селекцій і проекцій перетворюється на одиничну селекцію, одиничну проекцію чи селекцію з наступною проекцією (властивості 2, 3е, 4). Перетворення може суперечити твердженню «роби проекцію якомога раніше», однак ефективніше виконати всі можливі операції селекції і проекції за один перегляд відношення, ніж здійснювати кілька переглядів.

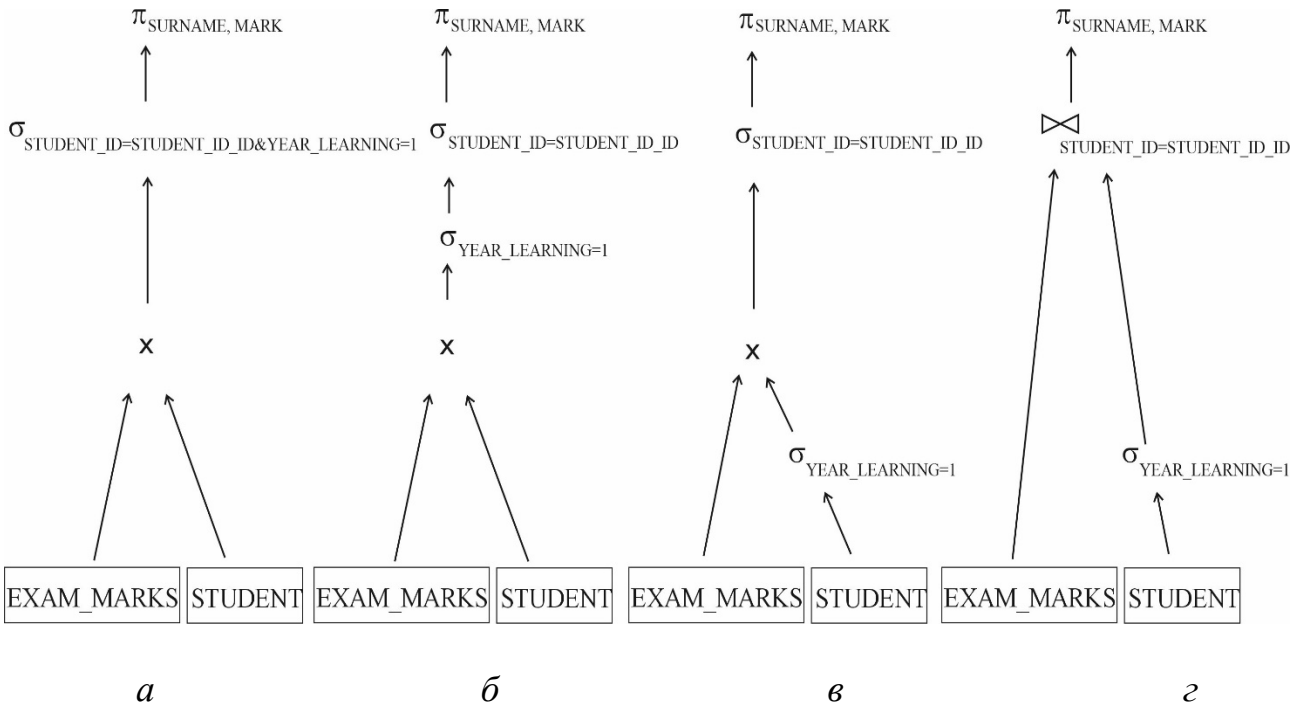


Рис. 6. Приклад еквівалентних перетворень, що оптимізують вирази реляційної алгебри: *а* – вихідний вираз обчислюється так, як він записаний; *б* – селекція поділяється на каскад селекцій відповідно до властивості 4; *в* – одна з отриманих селекцій опускається нижче декартового добутку відповідно до властивості 5а; *г* – декартовий добуток і селекція замінюються на з'єднання згідно з властивістю 6.

3. РЕЛЯЦІЙНЕ ЧИСЛЕННЯ КОДДА

3.1. Завдання реляційного числення

Завдання реляційного числення полягає у тому, щоб можна було сформулювати спеціальне числення предикатів, яке інтерпретується реляційними відношеннями.

З множини формул числення предикатів варто вибрати таку підмножину, і яка давала б змогу, одержувати формули, що інтерпретуються скінченними відношеннями.

Більшість працюючих мов запитів засновані на реляційному численні. Реляційні обчислення – непроцедурні системи. Обчислення висловлюють тільки те, яким повинен бути результат обчислення, але не те, яким чином проводити обчислення. Це завдання покладається на процесор мови запитів даної СУБД.

Розрізняють реляційне числення кортежів і реляційне числення доменів. Оскільки реляційне числення доменів схоже з реляційним обчисленням кортежів за винятком того, що змінні приймають значення в доменах, а не є кортежами, а числення кортежів використовується частіше, розглянемо тільки числення кортежів.

Реляційне числення кортежів є, по суті, формалізацією системи позначень, призначеної для утворення множин. В реляційному численні використовуються булеві операції (І, АБО, НЕ) над умовами, які можуть бути істинними або помилковими. У ньому також використовуються квантори існування і загальності, які означають, відповідно, що елемент певного типу існує або що умова істинна для кожного елемента певного типу.

3.2. Цільовий список і визначальний вираз

Розглянемо реляційне відношення бази даних вищого навчального закладу:

r – **UNIVERSITY** (UNIV_ID, UNIV_NAME, RATING, CITY)

Запит: Які університети знаходяться в місті 'Lutsk'?

У реляційній алгебрі для виконання цього запиту необхідно використовувати вираз, який повинен містити наступні операції:

- операцію Вибір над відношенням r, результатом її застосування до відношення r є інше відношення, яке представляє собою підмножину кортежів відношення r зі значенням, рівним 'Lutsk' в атрибуті CITY;
- проекція результату попередньої операції на атрибут UNIV_NAME.

У реляційному численні формулювання цього запиту повинен мати наступний вигляд:

{t.UNIV_NAME | t in r and t.CITY = 'Lutsk'},

де t – це змінна, що позначає довільний рядок.

Відношення, з якого береться t, визначається виразом "in r" яке означає, що t – рядок відношення. t.UNIV_NAME – значення атрибута UNIV_NAME в рядку r; символ (|) – розділяє цільової список і визначальний вираз. В даному випадку:

t.UNIV_NAME – цільовий список;

t in r and t.CITY = 'Lutsk' – визначальний вираз;

t.CITY = 'Lutsk' означає, що значення атрибута CITY в рядку t дорівнює 'Lutsk'.

Фігурні дужки "{}", в які укладено вираз, визначають результат запиту, як множину значень даних. Що саме входить в цю множину, описує вираз в дужках.

Розв'язком кожного запиту в реляційному численні є відношення, яке задається цільовим списком і визначальним виразом. Цільовий список визначає атрибути відношення розв'язку. Визначальний вираз – це умова, на підставі якої відбираються значення з бази даних, які увійдуть до відношення розв'язку.

У попередньому прикладі UNIV_NAME береться з рядка і поміщається в рядок розв'язку, якщо рядок t задовольняє умові t in r and t.CITY = 'Lutsk' . Система переглядає рядки відношення r один за одним. Першому рядку тимчасово присвоюється ім'я t і перевіряється істинність визначального виразу. Якщо значення виразу істинне, то рядок поміщається у відношення розв'язку. Потім система переходить до наступного рядка, дає їй ім'я t і знову перевіряє істинність визначального виразу. Процес повторюється для кожного рядка відношення r.

Цільовий список може складатися з декількох атрибутів, розділених комами.

{t.UNIV_NAME, t. RATING, t. CITY | t in r and t.CITY = 'Lutsk'},

У визначальному виразі, умова, за якою відбираються записи в результуюче відношення, будується за допомогою стандартних операцій логіки AND, OR, NOT, шести операцій порівняння (=,! =,>,> =, <, <=) і оператора IN.

Таким чином, розглянутий приклад демонструє запис операцій вибірки і створення проекцій. Крім того, з розглянутих вже конструкцій можна отримати і всі аналоги теоретико-множинних операцій. Розглянемо теоретико-множинні операції на прикладах, при цьому ми повинні пам'ятати, що реляційні відношення повинні бути сумісними.

Нехай дані два відношення:

r – **LECTURER1** (LECT_ID, NAME, CITY, PHONE)

s – **LECTURER2** (LECT_ID, NAME, CITY, PHONE)

Нехай ми маємо алгебраїчний вираз $\pi_{CITY}(r \cap s)$. Даний вираз містить операцію перерізу і операцію проекції.

У реляційному ж численні, якщо *t* кортеж *r*, а *y* кортеж *s*, цей запит може включати в себе наступні компоненти:

t.CITY – цільовий список;

t in *r* and *y* in *s* and *t*.LECT_ID = *y*. LECT_ID and *t*. NAME = *y*. NAME and *t*.CITY = *y*.CITY and *t*.PHONE = *y*.PHONE – визначальний вираз.

{*t*.CITY| *t* in *r* and *y* in *s* and *t*.LECT_ID = *y*. LECT_ID and *t*. NAME = *y*. NAME and *t*.CITY = *y*.CITY and *t*.PHONE = *y*.PHONE}

Це означає, що всі поля кортежа *t* рівні полям кортежа *y*. У результуючий набір взято ті кортежі, які є і у *r*-відношенні і у *s*- відношенні.

В даному випадку не обв'зково порівнювати відповідні поля. Тобто даний вираз можна записати:

{*t*.CITY| *t* in *r* and *t* in *s*}

Аналогічно можна отримати конструкції реляційного числення, які відповідають операціям об'єднання, різниці і декартового добутку.

Напрприклад розглянемо вираз, що містить операцію об'єднання: $\pi_{CITY}(r \cup s)$.

t.CITY – цільовий список;

t in *r* or *t* in *s* – визначальний вираз.

{*t*.CITY| *t* in *r* or *t* in *s* }

Розглянемо вираз, що містить операцію різниці: $\pi_{CITY}(r\text{-}s)$.

Результатом буде множина кортежів:

{*t*.CITY| *t* in *r* and *t* not in *s*}

Операцію декартового добутку розглянемо для двох відношень:

r – **LECTURER** (LECT_ID, NAME, CITY, PHONE, UNIV_ID)

s – **UNIVERSITY (**UNIV_ID, UNIV_NAME, CITY, RATING**)**

r – **LECTURER**

LECT_ID	NAME	CITY	UNIV_ID
1	Victor	Uzhhorod	1
2	Semen	Uzhhorod	1
3	Stepan	Kyiv	2

s – **UNIVERSITY**

UNIV_ID	UNIV_NAME	CITY	RATING
1	Uzhhorod National University	Uzhhorod	67
2	Taras Shevchenko National University of Kyiv	Kyiv	1

($r \times s$):

{*t*.LECT_ID, *t*.NAME, *t*.CITY, *t*. UNIV_ID, *y*. UNIV_ID, *y*. UNIV_NAME, *y*.CITY, *y*. RATING, | *t* in *r* and *y* in *s* }

Результатом буде наступна множина:

r.LECT_ID	*r*.NAME	*r*.CITY	*r*.UNIV_ID	*s*.UNIV_ID	*s*.UNIV_NAME	*s*.CITY	*s*.RATING
1	Victor	Uzhhorod	1	1	Uzhhorod National University	Uzhhorod	67
2	Semen	Uzhhorod	1	1	Uzhhorod National University	Uzhhorod	67
3	Stepan	Kyiv	2	1	Uzhhorod National University	Uzhhorod	67
1	Victor	Uzhhorod	1	2	Uzhhorod National University	Kyiv	1
2	Semen	Uzhhorod	1	2	Taras Shevchenko National University of Kyiv	Kyiv	1
3	Stepan	Kyiv	2	2	Taras Shevchenko National University of Kyiv	Kyiv	1

Аналізуючи результат виконання запиту, ми бачимо, що в деяких рядках містяться несумісні дані: в одному рядку дані про викладача і дані про університет іншого викладача. Для того, щоб отримати коректні дані, в результуючий набір нам потрібно взяти тільки відповідні дані, тобто ті дані де поле *r*.UNIV_ID = *s*.UNIV_ID.

{*t*.LECT_ID, *t*.NAME, *t*.CITY, *t*. UNIV_ID, *y*. UNIV_ID, *y*. UNIV_NAME, *y*.CITY, *y*. RATING, | *t* in *r* and *y* in *s* and *r*.UNIV_ID = *s*.UNIV_ID. }

Трохи інакше виглядають аналоги тільки двох операцій реляційної алгебри – операцій з'єднання і ділення. для побудови аналогів цих операцій потрібні квантори: квантор існування для з'єднання і квантор загальності для поділу.

3.3. Квантор існування

Квантор існування означає, що існує хоча б один екземпляр певного типу об'єктів. У реляційному численні квантор існування використовується для завдання умови того, що певний тип рядків у відношенні існує.

Розглянемо відношення:

LECTURER (LECT_ID, NAME, CITY, PHONE, UNIV_ID)

UNIVERSITY (UNIV_ID (PK),UNIV_NAME, RATING, CITY**)**

Запит: Вивести прізвища викладачів, якщо викладач прикріплений до якогось університету.

Цільовим списком тут буде: *t*.NAME.

де *t* – рядок з відношення LECTURER, а *y* – рядок відношення UNIVERSITY.

Формування визначального виразу здійснюється, виходячи з такого. Для того щоб прізвище викладача увійшло в відношення розв'язку, має задовольнятися умова, що є такий університет до якого він прикріплений. Іншими словами, якщо поле UNIV_ID зустрічається в рядку відношення LECTURER і існує таке значення UNIV_ID у відношенні UNIVERSITY, то поле NAME повинно бути включене в розв'язок. Таким чином, умова така: існує хоча б один рядок у відношенні UNIVERSITY, що містить необхідне значення UNIV_ID. Це формується таким чином:

exists y in UNIVERSITY (*y*.UNIV_ID = *t*.UNIV_ID).

Такий вираз читається: «Існує рядок *y* у відношенні UNIVERSITY такий, що *y*.UNIV_ID = *t*.UNIV_ID». Наведений вираз визначає рядок *t*. Якщо він істинний, тобто для рядка *t* існує такий рядок *y*, то *t*. Поле NAME, поміщається в результуюче відношення. Якщо вираз помилковий – тобто такого рядка у не існує – тоді *t*.NAME, не поміщається в результуюче відношення. Повний розв'язок в реляційному численні виглядає наступним чином:

{*t*.NAME | *t* in LECTURER and exists *y* in UNIVERSITY (*y*.UNIV_ID = *t*.UNIV_ID)}.

У реляційній алгебрі для виконання цього запиту потрібно виконати операцію з'єднання. Таким чином, квантор існування використовується в реляційному обчисленні, для того щоб виконувати функцію з'єднання.

3.4. Квантор загальності

Квантор загальності означає, що деяка умова застосовується до всіх рядків або до кожного рядка деякого типу. Він використовується в тих же цілях, що і операція ділення реляційної алгебри.

Нехай дано відношення:

STUDENT_V (Відомість_Студента) (SURNAME, NAME, SUBJ, EXAM_DATE)

EXAM (Розклад_екзаменів) (SUBJ, EXAM_DATE)

Потрібно визначити прізвища тих студентів, кожен з яких здавав всі екзамени перераховані в таблиці **EXAM**. Необхідно звернути увагу на те, що умова вибору прізвища студента містить визначення кожен. У результуюче відношення включаються тільки ті прізвища студентів, які здавали всі екзамени.

Повний розв'язок в реляційному численні з використанням квантора загальності FORALL наступний:

{t.SURNAME | t in STUDENT_V and forall *s* in EXAM (t.SUBJ = *s*.SUBJ and t.EXAM_DATE = *s*.EXAM_DATE)},

де *t* – кортеж відношення STUDENT;

s – кортеж відношення EXAM.

Прізвище студента з стрічки t таблиці STUDENT_V попадає в результуючий набір, якщо визначальний вираз істинний для стрічки t. А визначальний вираз істинний, якщо для кожної стрічки *s* відношення EXAM повинна існувати стрічка t у відношенні STUDENT_V з одним і тим самим прізвищем.

Наприклад маємо заповнені відношення STUDENT_V та EXAM

STUDENT

SURNAME	NAME	SUBJ	EXAM_DATE
Olijnyk	Andrij	Programming	01.05.2020
Bobryk	Ivan	Programming	01.05.2020
Fedun	Pavlo	Programming	01.05.2020
Panasiuk	Vitalij	Programming	01.05.2020
Kotan	Mykola	Discrete Math	13.05.2020
Olijnyk	Andrij	Discrete Math	13.05.2020
Fedun	Pavlo	Discrete Math	13.05.2020
Fedun	Pavlo	Databases	23.05.2020
Bobryk	Ivan	Databases	23.05.2020
Olijnyk	Andrij	Databases	23.05.2020

EXAM

SUBJ	EXAM_DATE
Programming	01.05.2020
Discrete Math	13.05.2020
Databases	23.05.2020

Повна відповідність всім стрічкам відношення EXAM є тільки у студентів Olijnyk Andrij та Fedun Pavlo. Тільки ці студенти здавали всі іспити, тому дані про них і входять в результуючий набір.

3.5. Приклади розв'язування задач на застосування реляційного числення

Розглянемо реляційну схему бази даних вищого навчального закладу подану на рис. 5. Для прикладу будемо розглядати ті самі задачі, що і в п 1.3. Аналогічно приклади згрупуємо за операціями, застосування яких у них демонструється. Пояснення до прикладів залишаються ті самі, що в п.1.3.

Вибірка.

1. Отримати прізвища усіх викладачів.

{t. SURNAME | t in LECTURER }

2. Отримати прізвища усіх викладачів і місто, де вони проживають.

{t. SURNAME, t. CITY | t in LECTURER }

3. Отримати прізвища усіх студентів їх день народження та рік навчання.

{t. SURNAME, t.BIRTHDAY, t.YEAR_LEARNING | t in STUDENT}

4. Отримати назви всіх університетів та їх рейтинг.

{t.UNIV_NAME, t.RATING | t in UNIVERSITY}

Обмеження

5. Отримати всі відомості про студентів першого року навчання.

{t.STUDENT_ID, t.SURNAME, t.NAME, t.STIPEND, t.YEAR_LEARNING, t.CITY, t.BIRTHDAY, t.UNIV_ID_ID| t in STUDENT and t.YEAR_LEARNING =1}

6. Отримати всі відомості про предмети, які вивчаються в 5 семестрі і на вивчення яких відводиться більше 100 годин.

{t.SUBJ_ID, t.SUBJ_NAME, t.HOUR, t.SEMESTER| t in SUBJECT and t.SEMESTER=5 and t.HOUR>100}

7. Вивести всі відомості про університети, які знаходяться в місті 'Lviv' або 'Lutsk'.

{t.UNIV_ID, t.UNIV_NAME, t.RATING, t.CITY| t in UNIVERSITY and ((t.CITY='Lviv') or (t.CITY='Lutsk'))}

Обмеження з проекцією

8. Отримати прізвища всіх студентів першого року навчання.

{t.SURNAME | t in STUDENT and t.YEAR_LEARNING =1}

9. Отримати прізвища та імена усіх викладачів які проживають в місті 'Lviv'.

{t.SURNAME, t.NAME | t in LECTURER and t.CITY='Lviv' },

10. Отримати прізвища усіх студентів першого року навчання та їх день народження.

{t.SURNAME, t.BIRTHDAY | t in STUDENT and t.YEAR_LEARNING =1},

11. Отримати назви всіх університетів міста 'Lviv', якщо його рейтинг знаходиться в межах (10–40).

{t.UNIV_NAME | t in UNIVERSITY and t.CITY='Lviv' and t.RATING>10 and t.RATING<40},

З'єднання

12. Отримати всі відомості про студентів , та всі відомості про їх оцінки (всі дані з таблиці STUDENT та всі дані з таблиці EXAM_MARKS)

В даному запиті, для того *щоб виконувати функцію з'єднання двох таблиць STUDENT і EXAM_MARKS потрібно використати квантор існування.*

{ t.STUDENT_ID, t.SURNAME, t.NAME, t.STIPEND, t.YEAR_LEARNING, t.CITY, t.BIRTHDAY, t.UNIV_ID_ID, *y*.EXAM_ID, *y*.SUBJ_ID_ID, *y*.MARK, *y*.EXAM_DATE | *t* in STUDENT and exists *y* in EXAM_MARKS (*t*.STUDENT_ID = *y*.STUDENT_ID_ID)}.

З'єднання, обмеження, проекція

13. Отримати прізвища студентів та їх оцінки.

{t.SURNAME, *y*.MARK | *t* in STUDENT and exists *y* in EXAM_MARKS (*y*.STUDENT_ID = *t*.STUDENT_ID_ID)}.

14. Отримати прізвища студентів та назви предметів по яких вони здавали екзамени та дати здачі цих екзаменів.

{t.SURNAME, *y*.EXAM_DATE, z. SUBJ_NAME | *t* in STUDENT and exists *y* in EXAM_MARKS ((*y*.STUDENT_ID = *t*.STUDENT_ID_ID) and exists *z* in SUBJECT (*y*.SUBJ_ID_ID =z.SUBJ_ID))}.

15. Отримати прізвища викладачів, та предмети, які вони викладають.

{t.SURNAME, z.SUBJ_NAME | *t* in LECTURER and exists *y* in SUBJ_LECT ((*t*.LECTURER_ID = *y*.LECTURER_ID_ID) and exists *z* in SUBJECT (*y*.SUBJ_ID_ID = z.SUBJ_ID))}.

16. Отримати прізвища студентів та їх оцінки з предмету 'Programming'.

{t.SURNAME, *y*.MARK | *t* in STUDENT and exists *y* in EXAM_MARKS ((t.STUDENT_ID = *y*.STUDENT_ID_ID) and exists *z* in SUBJECT (z.SUBJ_ID = *y*.SUBJ_ID_ID)) and *z*.SUBJ_NAME='Programming'}.

17. Отримати прізвища студентів, які вивчають предмет 'Programming' у викладача з прізвищем 'Makarchuk'. Також вивести оцінку з даного предмета.

{t.SURNAME, *y*.MARK | *t* in STUDENT and exists *y* in EXAM_MARKS ((t.STUDENT_ID = *y*.STUDENT_ID_ID) and exists *z* in SUBJECT (z.SUBJ_ID = *y*.SUBJ_ID_ID) and *p* in SUBJ_LECT ((*z*.SUBJ_ID = *p*.SUBJ_ID_ID) and exists *k* in LECTURER (*p*.LECTURER_ID_ID = *k*.LECTURER_ID))) and *z*.SUBJ_NAME='Programming'}.

Теоретико-множинні операції

18. Отримати прізвища та імена усіх студентів і викладачів, які проживають в місті 'Lviv'.

В даному завданні ми маємо операцію об'єднання.

{t. SURNAME, t. CITY | (t in STUDENT or t in LECTURER) and t.CITY='Lviv'}

19. Отримати міста, де проживають і викладачі і студенти?

В даному завданні ми маємо операцію перетину.

{t. CITY | (t in STUDENT and exists y in LECTURER (t.CITY = y.CITY)}.

20. Отримати прізвища викладачів, які не викладають жодної дисципліни?

В даному завданні ми маємо операцію різниці.

{t.SURNAME | *t* in LECTURER and not(*s* in LECTURER and exists *y* in SUBJ_LECT ((*s*.LECTURER_ID = *y*.LECTURER_ID_ID) and exists *z* in SUBJECT (*y*.SUBJ_ID_ID = z.SUBJ_ID)))}.

Операція ділення

21. Отримати ідентифікаційні номери студентів, які здали всі предмети.

Побудову запиту виконаємо поетапно.

Оскільки потрібні ідентифікаційні номери студентів, то необхідно шукати записи у відношенні STUDENT. Тому цільовий список повинен містити атрибут STUDENT_ID з цього відношення, а визначальний вираз оператор приналежності IN до STUDENT, наприклад, так:

{t.STUDENT_ID | t in STUDENT... }

Оскільки нас цікавлять студенти, котрі складали хоча б один раз КОЖЕН предмет, то до відношення SUBJECT слід застосувати квантор загальності:

{t.STUDENT_ID | t in STUDENT and forall s in SUBJECT... }

І нарешті, оскільки нас цікавлять СТУДЕНТИ, які здавали кожен предмет, то відповідні кортежі повинні існувати в відношенні EXAM_MARKS

{t.STUDENT_ID | *t* in STUDENT and forall s in SUBJECT (exists r IN EXAM_MARKS (r.SUBJ_ID_ID =s.SUBJ_ID and t.STUDENT_ID = r.STUDENT_ID))}

4. ЕКВІВАЛЕНТНІСТЬ РЕЛЯЦІЙНОЇ АЛГЕБРИ І РЕЛЯЦІЙНОГО ЧИСЛЕННЯ

Реляційна алгебра та реляційне числення є формальними мовами запитів для реляційної моделі. Обидві форми є базою для мови SQL, яка використовується в більшості реляційних СУБД. Реляційна алгебра є процедурною мовою, а реляційне числення є декларативною мовою. Розв'язуючи задачі реляційного числення, ми одночасно показували його аналогічність реляційній алгебрі. В табл 1. проілюстровано еквівалентність підходів.

Таблиця 1.

Зведені операції реляційної алгебри та реляційного числення

Операція	**Опис**	**Синтаксис реляційної алгебри**	**Синтаксис реляційного числення**
Проекція	На вході – одне відношення *R*1. Вибір з відношення окремих атрибутів	R1[R1.*поле 1, R*1.*поле2,* ...]	{*r.поле1, r.поле2,* …| *r* in *R*1}
Вибірка	На вході – одне відношення *R*1. Вибір з відношення окремих кортежів, які задовольняють умову відбору	*R*1[умова вибору]	{*r.поле1, r.поле2,* ... | *r* in *R*1 and [умова вибору]}
Об'єднання	На вході – два сумісних відношення *R*1 та *R*2. Включення в результуюче відношення всіх кортежів, які входять у хоча б одне відношення.	*R*1∪ *R*2	{r.*поле1,* r.*поле2,* ... | r in *R*1 or r in *R*2}
Перетин	На вході – два сумісних відношення *R*1 та *R*2. Включення в результуюче відношення всіх кортежів, які входять в обидва відношення одночасно.	*R*1∩ *R*2	{r.*поле1,* r.*поле2,* ... | r in *R*1 and r in *R*2}
Різниця	На вході – два сумісних відношення *R*1 та *R*2. Включення в результуюче відношення всіх кортежів, які входять в *R*1 і не входять в *R*2	*R*1- *R*2	{r.*поле*1, r.*поле*2, ... | r in *R*1 and r not in *R*2}
Декартовий добуток	На вході – два відношення *R*1 та *R*2. Результуюче відношення складається з всіх кортежів *R*1, до кожного з яких приєднані всі кортежі з *R*2	*R*1×*R*2	{r.*поле*1, r.*поле*2, ... , *s.поле*1, *s.поле*2,… | r in *R*1 and s in *R*2}

З'єднання	На вході два відношення *R*1 та *R*2, які мають атрибути зв'язку *R*1.*n1* та *R*2.*n2*. Результуюче відношення містить відповідні дані з обох відношень.	1. Природне з'єднання *R*1**R*2 2. З'єднання за умовою $R1 \bowtie_F R2$ 3.Зовнішнє з'єднання $R \supset \lhd S$	1. {r.*поле*1, r.*поле*2, ... , *s.поле*1, *s.поле*2, … | r in *R*1and exists s in *R*2 (s.*n1*=s.*n2*) } 2. { r.*поле*1, r.*поле*2, ... , *s.поле*1, *s.поле*2, … | r in *R*1 and exists s in *R*2 (s.*n1* <оператор порівняння> s.*n2*) } 3. { r.*поле*1, r.*поле*2, ... , *s.поле*1, *s.поле*2, | (r in *R*1 and exists s in *R*2 (s.*n1* <оператор порівняння> s.*n2*)) or (r in *R*1 and not exists s in *R*2 (s.*n1* <оператор порівняння> s.*n2*))}.
Ділення	На вході два відношення *R*1 та *R*2, причому атрибути *R*2 мають складати підмножину атрибутів *R*1. Результат ділення *R*1/*R*2 містить ті кортежі, що входять в *R*1 з кожним кортежем з *R*2	*R*1 / *R*2	{r. *поле*1 | r in *R*1 and forall s in *R*2 (exists t in *R*1 (t.*n1*=s.*n2*) and (r.*n1*=t.*n1*)) }

Основні відмінності між реляційною алгеброю та реляційним численням:

- Основна відмінність між реляційною алгеброю та реляційним обчисленням полягає в тому, що реляційна алгебра є процедурною мовою, тоді як реляційне числення є декларативною.
- Реляційна алгебра визначає, як отримати результат, тоді як реляційне числення визначає, яку інформацію повинен містити результат.
- Реляційна алгебра вказує послідовність, в якій операції повинні виконуватися в запиті. З іншого боку, реляційне числення не визначає послідовність операцій, що виконуються в запиті.
- Реляційна алгебра не залежить від домену, тоді як реляційне числення може бути залежним від домену, оскільки ми маємо доменне реляційне числення.
- Мова запитів реляційної алгебри тісно пов'язана з мовою програмування, тоді як реляційний числення тісно пов'язане з природною мовою.

Лекція 4. ПРОЕКТУВАННЯ БАЗ ДАНИХ. НОРМАЛІЗАЦІЯ

4.1. Проблеми проектування баз даних

Проектування інформаційних систем, що включають в себе бази даних, здійснюється на фізичному і логічному рівнях. Рішення проблем проектування на фізичному рівні багато в чому залежить від СУБД, що використовується. У ряді випадків користувачеві мають можливість настройки окремих параметрів системи, яка не складає великої проблеми.

Логічне проектування полягає у визначенні числа і структури таблиць, формуванні запитів до БД, визначенні типів звітних документів, розробці алгоритмів обробки інформації, створення форм для введення і редагування даних в базі і вирішенні ряду інших завдань.

Рішення задач логічного проектування БД в основному визначається специфікою завдань предметної області. Найбільш важної тут є проблема структуризації даних, на ній ми зосередимо основний погляд.

При проектуванні структур даних для автоматизованих систем можна виділити три основні підходи:

1. Збір інформації про об'єкти розв'язуваної задачі в рамках однієї таблиці і подальша декомпозиція її на декілька взаємопов'язаних таблиць на основі процедури нормалізації відносин.
2. Формулювання знань про систему (визначення типів вихідних даних і їх взаємозв'язків) і вимог до обробці даних, отримання за допомогою CASE-системи (системи автоматизації проектування і розроблення баз даних) готової схеми БД або навіть готової прикладної інформаційної системи.
3. Структурування інформації для використання в інформаційної системі в процесі проведення системного аналізу на основі сукупності правил.

Розглянемо перший з названих підходів, що є історично першим. Перш за все, охарактеризуємо основні проблеми, що мають місце при визначенні структур даних у відносинах реляційної моделі.

Проблема 1 - Надмірне дублювання даних і аномалії.

Слід розрізняти просте і надмірне дублювання даних. Надлишкове дублювання даних може приводити к проблемам при обробці даних. Наведемо приклади обох варіантів дублювання.

Приклад не надлишкового дублювання даних приведене на рис.4.1 відношення «Співробітник і Телефон» (С_Т).

С_Т	
співробітник	Телефон
Іванов І.М.	3721
Петров М.І.	4328
Сидоров Н.Г.	4328
Єгоров В.В.	4328

Рисунок 4.1 – Не надмірне дублювання

Для співробітників, які перебувають в одному приміщенні, номери телефонів збігаються. Номер телефону 4328 зустрічається кілька разів, хоча для кожного службовця номер телефону унікальний. Тому жоден з номерів не є надмірною. Дійсно, при видаленні одного з номерів буде загублена інформація про те, за яким номером можна додзвонитися до одного з працівників.

Приклад надлишкового дублювання наведене на рис.4.2а. Відношення С_Т_Н доповнене атрибутом «номер кімнати співробітника». Природно припустити, що кожен, хто працює в одній кімнаті мають один і той же телефон. Отже, в даному відношенні є надлишкове дублювання даних. Так, в зв'язку з тим, що Сидоров і Егоров знаходяться в тій же кімнаті, що і Петров, їх номери можна дізнатися з кортежа з даними про Петрова.

На рис.4.2б приведен приклад невдалого відношення С_Т_Н, в якому замість телефонів Сидорова і Єгорова поставлені прочерки. Не вдаліть подібного способу виключення надлишкових даних полягає в наступному. По-перших, при програмуванні доведеться витратити додаткові зусилля на створення механізму пошуку інформації для таблиці.

С_Т_Н

а)

Співробітник	Телефон	Н_комн
Іванов І.М.	3721	109
Петров М.І.	4328	111
Сидоров Н.Г.	4328	111
Єгоров В.В.	4328	111

б)

Співробітник	Телефон	Н_комн
Іванов І.М.	3721	109
Петров КШ.	4328	111
Сидоров Н.Г.	-	111
Єгоров В.В.	-	111

Рисунок 4.2 - Надмірне дублювання

По-друге, пам'ять все одно буде виділитися під комірки з прочерками, хоча дублювання даних і виключено. По-третє, що особливо важливо при виключенні з колективу Петрова кортеж з відомостями про нього буде видалений з відносини, а значить, видалений інформація про телефон 111-й кімнати, що не припустимо.

Можливий спосіб виходу з даній ситуації наведено на рис.4.3. Тут показані два відносини С_Н і Н_Т, отримані шляхом декомпозиції вихідного відношення С_Т_Н. Перше з них містить інформацію про номери кімнат, в яких розташовуються співробітники, а друге - інформацію про номери телефонів в кожній з кімнат. Тепер, якщо Петрова і звільнять з установи і, як наслідок цього, видалять будь-яку інформацію про нього з баз даних установи, це не призведе до втрати інформації про номер телефону в 111-й кімнаті.

Т_Н

Телефон	Н_комн
3721	109
4328	111

С_Н

співробітник	Н_комн
Іванов І.М.	109
Петров М.І.	111
Сидоров Н.Г.	111
Єгоров В.В.	111

Рисунок 4.3 - Виняток надлишкового дублювання

Процедура декомпозиції відношення С_Т_Н на два відношення С_Н і Н_Т є основною процедурою нормалізації відносин.

Надмірне дублювання даних створює проблеми при обробленні кортежів відносини, названі «*аномаліями проблеми ставлення*». Ці проблеми виникають при спробі видалення, додання або редагування кортежів.

Аномаліями називають таку ситуацію в таблицях БД, яка приводить до суперечностей в БД або істотно ускладнює обробку даних.

Виділяють три основні види аномалій: аномалії модифікації (або редагування), аномалії видалення і аномалії додання.

Аномалії модифікації проявляються в тому, що зміна значення одного даного може спричинити за собою перегляд всієї таблиці і відповідна зміна деяких інших записів таблиці.

Так, наприклад, зміна номера телефону в кімнаті 111 (рис 4.2а) зажадає перегляду всієї таблиці С_Т_Н і зміни поля Н_комн відповідно поточного вмісту таблиці в записах, що відносяться до Петрова, Сидорова і Єгорова.

Аномалії видалення полягають у тому, що при видаленні будь-якого даного з таблиці може пропасти і інша інформація, яка не пов'язана безпосередньо з даними що видаляються.

У тій же таблиці С_Т_Н видалення запису про співробітника Іванова (наприклад, через звільнення) приводить до зникнення інформації про номер телефону, встановленого в 109-й кімнаті.

Аномалії додавання виникають у випадках, коли інформацію в таблицю не можна помістити до тих пір, поки вона неповна, або вставка нової записи вимагає додаткового перегляду таблиці.

Прикладом може служити операція додавання нового співробітника все в ту ж таблицю С_Т_Н. Очевидно про, буде протиприродним зберігання в цій таблиці відомостей тільки про кімнату і номер телефону в ній, поки ніхто зі співробітників не поміщений в неї. Більш того, якщо в таблиці С_Т_Н поле «Службовець» є ключовим, то зберігання в ній неповних записів з відповідним прізвищем службовця просто неприпустимо через не визначеності значення ключового поля.

Другим прикладом виникнення аномалії додавання може бути ситуація включення в таблицю нового співробітника. При додаванні таких записів для виключення протиріч бажано перевірити номер телефону та відповідний номер кімнати хоча б з одним із співробітників, сидячих з новим співробітником в тій же кімнаті. Якщо ж виявиться, що у не скількох співробітників, сидячих в одній кімнаті, є різні телефони, то взагалі незрозуміло, що робити (чи то в кімнаті кілька телефонів, то чи якійсь з номерів помилковий).

Проблема 2 - Формування вихідного відношення.

Проектування БД починається з визначення всіх об'єктів, відомості про яких будуть включені в базу, і визначення їх атрибутів. Потім атрібути зводяться в одну таблицю - вихідне відношення.

Приклад. Формування вихідного відношення.

Припустимо, що для навчальної частини факультету створюється БД про викладача. На першому етапі проектування БД в результаті спілкування з завідувачем навчальною частиною повинні бути визначені відомості про те,

як база даних повинна використовуватися, і яку інформацію замовник хоче получати в процесі її експлуатації. В результаті встановлюються атрибути, які повинні міститися в відношеннях БД, і зв'язки між ними. Перечислимо імена виділених атрибутів і їх кратки характеристики:

ПІБ - прізвище та ініціали викладача. Виключаємо можливість збігу прізвища та ініціалів у викладачів.

Должн - посада, яку займає викладачем.

Оклад - оклад викладача.

Стаж - викладацький стаж.

Д_Стаж - надбавка за стаж.

Каф - номер кафедри, на якій значиться викладач.

Предм - назва предмета (дисципліни), що читається викладачем.

Група - номер групи, в якій викладач проводить заняття.

ВідЗан - вид занять, що проводяться викладачем в навчальній групі.

Одна з вимог до відносин полягає в тому, щоб все атрибути відносини мали атомарні (прості) значення. У вихідному відношенні кожен атрибут кортежу також повинен бути простим. Приклад вихідного відношення ВИКЛАДАЧ наведений на рис.4.4.

ВИКЛАДАЧ

ПІБ	должн	оклад	стаж	Д_Стаж	каф	предм	Група	ВідЗан
Іванов І.М.	преп	5000	5	1000	25	СУБД	256	практ
Іванов І.М.	преп	5000	5	1000	25	ПЛ / 1	123	практ
Петров М.І.	ст.преп	8000	7	1000	25	СУБД	256	лекція
Петров М.І.	ст.преп	8000	7	1000	25	Паскаль	256	практ
Сидоров Н.Г.	преп	5000	10	1500	25	ПЛ / 1	123	лекція
Сидоров Н.Г.	преп	5000	10	1500	25	Паскаль	256	лекція
Єгоров В.В.	преп	5000	5	1000	24	ПЕОМ	244	лекція

Рисунок 4.4 - Початкове відношення ВИКЛАДАЧ

Початкове відношення ВИКЛАДАЧ містить надмірне дублювання даних, яке і є причиною аномалій редагування. Розрізняють надмірність явну і неявну.

Явна надмірність полягає в тому, що по відношенню до ВИКЛАДАЧ рядка з даними про викладачів, які проводять заняття в декількох групах, повторюються відповідне число раз. Наприклад, всі дані по Іванову повторюються двічі. Тому, якщо Іванов І.М. стані старшим викладачем, то цей факт повинен бути відображений в обох рядках. В іншому випадку буде мати місце протиріччя в даних, що є прикладом аномалії редагування, зумовленої явною надмірністю даних в відношенні.

Неявна надмірність щодо ВИКЛАДАЧ є в однакових окладах у всіх викладачів і в однакових добавках до окладу за однаковий стаж. Тому, якщо при зміні окладів за посаду з 5000 на 5100 це значення змінять у всіх викладачів, крім, наприклад, Сидорова, то база стане суперечливою. Це приклад аномалії редагування для варіанта з неявній надмірністю.

Засобом виключення надмірності у відносинах і, як наслідок, аномалій є нормалізація відносин, розглянемо її більш детально.

4.2. Метод нормальних форм

Проектування БД є одним з етапів життєвого циклу інформаційної системи. Основним завданням, що вирішується в процесі проектування БД, є завдання нормалізації її відносин. Розглянутий нижче метод нормальних форм є класичним методом проектування реляційних БД. Цей метод заснований на фундаментальному в теорії реляційних баз даних понятті залежності між атрибутами відносин.

Розглянемо основні види залежностей між атрибутами відношень: функціональні, транзитивні і багатозначні.

Поняття функціональної залежності є базовим, так як на його основі формулюються визначення всіх інших видів залежностей.

Атрибут B функціонально залежить від атрибута A, якщо кожному значенню A відповідає в точності одне значення B.

Математично функціональна залежність У від А позначається записом А→В. Це означає, що у всіх кортежей з однаковим значенням атрибута А атрибут В матиме також одне і те ж значення. Відзначимо, що А і В можуть бути складовими, тобто складатися з двох і більш атрибутів.

Відносно на рис. 4.4 можна виділити функціональні відношення між атрибутами ПІБ→Каф, ПІБ→Должн, Должн→Оклад і інші. Наявність функціональної залежності в відношенні визначається природою речей, інформація про яких представлена кортежами відносини. У відношенні на рис.4.4 ключ є складовим і складається з атрибутів ПІБ, Предмет, Група.

Якщо існує функціональна залежність виду A → B і B→A, то між A і B є взаємно однозначна відповідність, або функціональна взаємозалежність.

Наявність функціональної взаємозалежності між атрибутами А і В позначимо як А↔В або В↔А.

Приклад. Нехай є деяке відношення, що включає два атрибуту, функціонально залежні один від одного. Це серія, номер паспорта (N) та прізвище, ім'я та по батькові власника (ПІБ). Наявність функціональної залежністі поля ПІБ від N означає не тільки той факт, що значення поля N однозначно визначає значення поля ПІБ, але і те, що одному і тому ж значенню поля N відповідає тільки єдине значення поля ПІБ. Зрозуміло, що в даному випадку діє і зворотна функціональна залежність: кожному значенню поля ПІБ відповідає тільки одне значення поля N. В даному прикладі передбачається, що ситуація наявності повного збіги прізвищ, імен та по батькові двох людей виключена.

Якщо відношення знаходиться в нормальній формі, то все не ключові атрибути функціонально залежать від ключа з різним ступенем залежності.

Часткової функціональною залежністю називається залежність не ключового атрибута від частини складного ключа.

В даному відношенні атрибут Должн знаходиться в функціональної залежності від атрибута ПІБ, що є частиною ключа. Тим самим атрибут Должн знаходиться в частковій залежності від ключа відношення.

Альтернативним варіантом є повна функціональна залежність не ключового атрибута від всього складного ключа. У нашому прикладі атрибут ВідЗан знаходиться в повній функціональної залежності від складного ключа.

Атрибут С залежить від атрибута А транзитивне (існує транзитивна залежність), якщо для атрибутів А, В, С виконуються умови А→В і В→С, але зворотна залежність відсутня.

Між атрибутами може мати місце багатозначна залежність.

В відношенні R атрибут В багатозначно залежить від атрибута А, якщо кожному значенню А відповідає безліч значень В, не пов'язаних з іншими атрибутами з R.

Багатозначні залежності можуть бути «один до багатьох» (1:М), «багато до одного» (М:1) або «багато до багатьох» (М:М), що позначаються відповідно: А=>В, А<=В і А<=>В.

Наприклад, нехай викладач веде кілька предметів, а кожен предмет може вестися декількома викладачами, тоді має місце залежність ПІБ⇔Предмет. Так, з рис. 4.4, видне, що викладач Іванов І.М. веде заняття по двом предметам, а дисципліна СУБД - читається двома викладачами: Івановим І.М. і Петровим М.І.

У загальному випадку між двома атрибутами одного відношення можуть існувати залежності: 1:1, 1:М, М:1 і М:М. Оскільки залежність між атрибутами є причиною аномалій, намагаються розчленувати відносини з залежностями атрибутів на ніскільки відносин. В результаті утворюється сукупність пов’язаних відношень (таблиць) зі зв'язками виду 1:l, 1:М, М:1 і М:М. Зв'язки між таблицями відображають залежності між атрибутами різних відносин.

Два або більше атрибута називаються взаємно незалежними, якщо жоден з цих атрибутів не є функціонально залежним від інших атрибутів.

Відсутність залежності атрибута A від атрибута B можна позначати так: A¬ → B. Випадок, коли A¬ → B і B¬ → A, можна позначить A¬ =B.

Процес проектування БД з використанням методу нормальних форм є ітераційним і полягає в послідовному перекладі відносин з першої нормальної форми в нормальні форми більш високого порядку за певними правилами. Кожна наступна нормальна форма обмежує певний тип функціональних залежностей, усуває відповідні аномалії при виконанні операцій над відношеннями БД і зберігає властивості попередніх нормальних форм.

Виділяють наступну послідовність нормальних форм:

- перша нормальна форма (1НФ);
- друга нормальна форма (2НФ);
- третя нормальна форма (ЗНФ);
- посилена третя нормальна форма, або нормальна форма Бойса-Кодда (БКНФ);
- четверта нормальна форма (4НФ);
- п'ята нормальна форма (5НФ).

4.3. Перша нормальна форма

Відношення знаходиться в 1НФ, якщо всі його атрибути є простими (мають єдине значення). Початкове відношення будується таким образом, щоб воно било в 1НФ.

Переклад відносини в наступну нормальну форму здійснюється методом «декомпозиції без втрат». Така декомпозиція повинна забезпечить те, що запити (вибірка даних по умові) до вихідного ставлення і до відносин, що отримуються в результаті декомпозиції, дадуть однаковий результат.

Основною операцією методу є операція проекції. Пояснім її на прикладі. Початкове відношення ВИКЛАДАЧ, що використовується для ілюстрації методу, має складовою ключ ПІБ, Предм, Група і знаходиться в 1НФ, оскільки всі його атрибути прості.

У цьому відношенні можна виділити часткову залежність атрибутів Стаж, Д_Стаж, Каф, Должн, Оклад від ключа. Вказані атрибути знаходяться у функціональній залежності від атрибута ПІБ якій є частиною складеного ключа.

Ця часткова залежність від ключа приводить до наступного:

1. В відношенні присутня явне і неявне надмірне дублювання даних, наприклад:

• повторення відомостей про стаж, посади і оклади викладачів, які проводять заняття в декількох групах і / або по різним предметів;

• повторення відомостей про оклади для однієї і тієї ж посади або про надбавки за однаковий стаж.

2. Наслідком надмірного дублювання даних є проблема їх редагування. Наприклад, зміна посади у викладача Іванова І.М. зажадає перегляду всіх кортежів відносини і внесення змін в ті з них, які містять відомості про даного викладача.

Частина надмірності усувається при перекладі відносини в 2НФ.

4.4. Друга нормальна форма

Відношення знаходиться в 2НФ, якщо воно знаходиться в 1НФ і кожний не ключовий атрибут функціонально повно залежить від первинного ключа (складного).

Для усунення часткової залежності і перекладу відношення в 2НФ необхідно, використовуючи операцію проекції, розкласти його на кілька відношень наступним чином:

- побудувати проекцію без атрибутів, які перебувають у частковій функціональної залежності від первинного ключа;
- побудувати проекції на частини складеного первинного ключа і атрибут, що залежать від цих частин.

В результаті отримаємо два відношення R1 і R2 в 2НФ (рис. 4.5).

Відносно R1 первинний ключ є складовим і складається з атрибутів ПІБ, Предм, Група. Нагадаємо, що даний ключ щодо R1 отриманий в припущенні, що кожен викладач в одній групі з одного предмету може або читати лекції, або проводити практичні заняття. Відносно R2 ключ ПІБ.

Дослідження відношень R1 і R2 показує, що перехід до 2НФ дозволяє виключити явну надмірність даних в таблиці R2 - повторення рядків з відомостями про викладачів. У R2 і раніше має місце неявне дублювання даних.

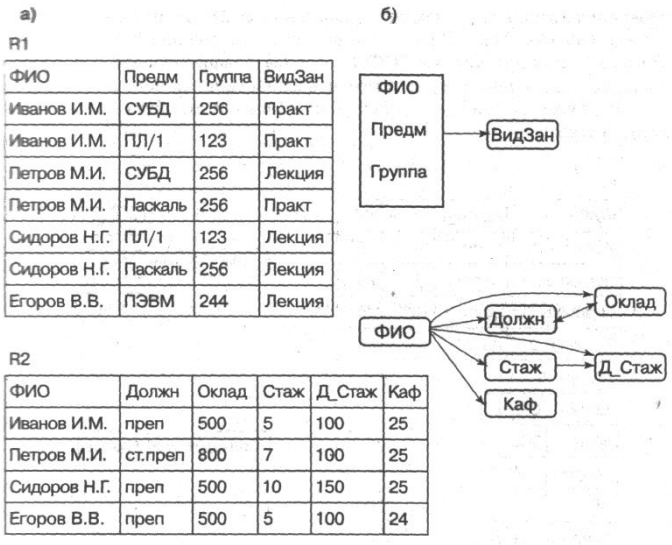


Рисунок 4.5 - Ставлення БД в 2НФ

Для подальшого вдосконалення відносини необхідно перетворити його в третю нормальну форму.

4.5. Третя нормальна форма

Відношення знаходиться в 3НФ в тому і тільки в тому випадку, якщо все не ключові атрибути відношення взаємно незалежні і повністю залежать від первинного ключа.

Взаємна незалежність атрибутів означає відсутність будь-якої залежності між атрибутами відносини, у тому числі і транзитивної залежності між ними.

Якщо по відношенню до R1 транзитивні залежності відсутні, то в відношенні R2 вони є:

ПІБ → Должн → Оклад,

ПІБ → Оклад → Должн,

ПІБ → Стаж → Д_Стаж,

Транзитивні залежності також породжують надлишкове дублювання інформації в відношенні. Для усунення їх виконують операцію проекції на атрибути, які є причиною транзитивних залежностей.

Графічно ці відносини представлені на рис.4.6б. Зауважимо, що ставлення R2 можна перетворити по-іншому, а саме: в відношенні R3 замість атрибута Должн взяти атрибут Оклад.

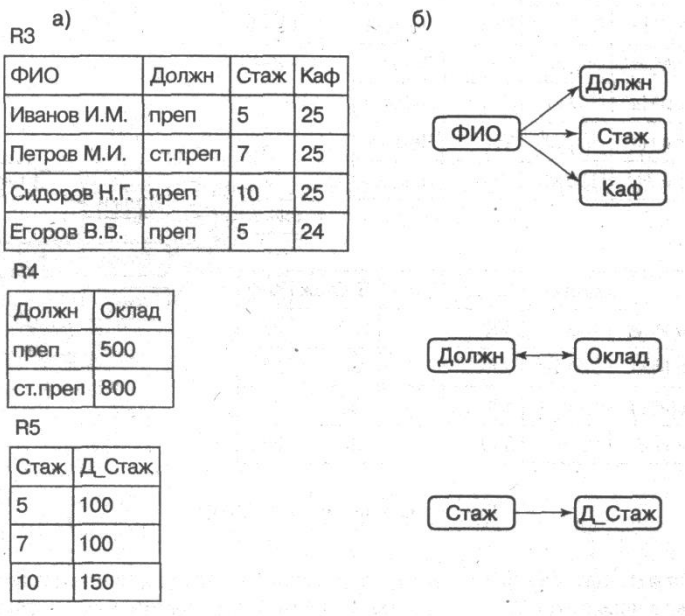


Рисунок 4.6 - Відносини БД в 3НФ

На практиці побудова 3НФ в більшості випадків є достатній. Дійсно, приведення відношення до 3НФ в нашому прикладі, призвело до усунення надмірного дублювання.

Якщо в плані є залежність атрибутів складеного ключа від не ключових атрибутів, то необхідно перейти до підсиленої 3НФ.

4.6. Четверта нормальна форма

Розглянемо приклад нового відношення ПРОЕКТИ, схема якого виглядає в такий спосіб: ПРОЕКТИ (Номер_проекту, Код_ співробітника, Завдання_ співробітника). Первинним ключем відношення є вся сукупність атрибутів: Номер_проекту, Код_ співробітника і Завданння_співробітника.

Відносно містяться номери проектів, для кожного проекту - список кодів співробітників-виконавців, а також список заданий, передбачених кожним проектом. Співробітники можуть брати участь в декількох проектах, і різні проекти можуть містити однакові завдання. Передбачається, що кожен співробітник виконує всі завдання з цього проекту.

При такій постановці питання єдиним можливим ключем відношення є складової атрибут Номір_проекту, Код_співробітника, Завдання_співробітника. Він, природно, і став первинним ключем відношення. Звідси слідує, що відношення ПРОЕКТИ, знаходиться в формі 3НФ.

Нехай вихідна інформація в цьому відношенні виглядає наступним чином:

ПРОЕКТИ

Номер_проєкту	Код_ співробітника	Завдання_ співробітника
001	05	1
001	05	2
001	05	3
004	02	1
004	02	2

004	03	1
004	03	2
004	05	1
004	05	2
007	06	1

Головний недолік відношення ПРОЕКТИ полягає в тому, що при добавленні / відсторонення від проекту співробітника припадає добавляти / виключати зі ставлення стільки кортежів, скільки завдань є в проекті. Внесення або виключення щодо одного факту про деякого співробітника вимагає серії елементарних операцій через дублювання значень в кортежі.

Звідси виникають питання: навіщо зберігати в кортежі повторювані значення кодів співробітників? Чи потрібно перераховувати всі завдання по кожному проекту, та ще для кожного співробітника-виконавця цього проекту? Чи не можна інформацію про прив'язку завдань до проекту помістити в окрему таблицю і виключити повторення в основній таблиці?

Зауважимо, що непряма ознака аномалії, як і раніше, - дублювання інформації в таблиці. Висловимо припущення, що причиною аномалії є наявність деякій залежності меж атрибутами відношення (як побачимо далі - багатозначною залежністю).

Дійсно, щодо ПРОЕКТИ існують дві багатозначні залежності:

Номер_проекту => Код_співробітника

Номер_проекту => Завдання_співробітника

У довільному відношенні R (A, B, C) може одночасно існувати багатозначна залежність A => B і A => C. Ця обставина позначається як A => B | C.

Під проектуванням без втрат розуміється такий спосіб декомпозиції відносини, при якому початкове ставлення повністю відновлюється шляхом з'єднання отриманих відносин.

Пояснимо проектування без втрат на прикладі.

Нехай є найпростіше відношення R (A, B, C), що має вигляд:

A	B	3
До	15	1
До	15	2
Л	10	1
М	20	1
М	20	2
М	20	3

Побудуємо проекції R1 і R2 на атрибути A, B і A, C відповідно. Вони будуть виглядати так:

R1 R2

A	B
До	15
Л	10
М	20

A	3
До	1
До	2
Л	1
М	1
М	2
М	3

Неважко бачити, що зв'язування R1 (A, B) і R2 (A, C) в точності народжує вихідне відношення R (A, B, C). Відносно R немає зайвих кортежів, немає і втрат.

Визначення четвертої нормальної форми. Відношення R знаходиться в четвертої нормальної формі (4НФ) в тому і тільки в тому випадку, коли є багатозначна залежність A => B, а все фундаментальні атрибути R функціонально залежать від A.

Наведене вище відношення ПРОЕКТИ можна представити у вигляді двох відносин: ПРОЕКТИ-СПІВРОБІТНИКИ і ПРОЕКТИ-ЗАВДАННЯ. Структура цих відносин і вміст відповідних таблиць виглядає наступним чином:

ПРОЕКТИ-СПІВРОБІТНИКИ (номер_проекту, Код_співробітники).

Первинний ключ відносини: Номер_проекту, Код_співробітника.

ПРОЕКТИ-СПІВРОБІТНИКИ

Номер_проекту	Код_ співробітника
001	05
004	02
004 '	03
004	05
007	06

ПРОЕКТИ-ЗАВДАННЯ (Номер_проекту, Завдання_співробітника).

Первинний ключ відносини: Номер_проекту, Завдання_співробітника.

ПРОЕКТИ-ЗАВДАННЯ

Номер_проекту	Завдання_співробітника
001	1
001	2
001	3
004	1
004	2
007	1

Як легко побачити, обидва цих відносини знаходяться в 4НФ і свобідні від помічених недоліків. Дублювання значень атрибутів кодів співробітників пропало.

У загальному випадку не будь-яке відношення можна відновити до вихідного. У нашому випадку відновлення можливо потому, що кожен співробітник виконував всі завдання по проекту (саме це вкладається в принцип 1:М з'єднання відносин). Самі ж співробітники брали участь в декількох проектах, і різні проекти могли містити однакові завдання.

4.7. П’ята нормальна форма

Результатом нормалізації всіх попередніх схем відносин були два нових відношення. Іноді це зробити не вдається, або одержувані відношення свідомо мають небажані властивості. В цьому випадку виконують декомпозицію вихідного відношення на відносини, кількість яких, перевищує два.

Розглянемо відношення СПІВРОБІТНИКИ-ВІДДІЛИ-ПРОЕКТИ. Первинний ключ відношення включає всі атрибути: Код_співробітника, Код_відділу і Номер_проекту. Нехай в цьому відношенні один співробітник може працювати в декількох відділах, причому в кожному відділі він може примати участь в декількох проектах. В одному відділі можуть працювати декілька співробітників, але кожен проект виконує тільки один співробітник. Функціональних і багатозначних залежностей між атрибутами нема.

СПІВРОБІТНИКИ-ВІДДІЛИ-ПРОЕКТИ

Код_співробітника	Код_відділу	Номер_проекту
01	РД	036
02	АТ	004
03	УП	004
04	АТ	019
05	ЛЗ	001
05	ЛЗ	004
06	УП	007
08	ВЦ	013
09	ВЦ	014
10	СЖ	013

Виходячи зі структури відносини СПІВРОБІТНИКИ-ВІДДІЛИ-ПРОЕКТИ можна зробити висновок, що воно знаходиться в формі 4НФ. Тим не менш, в відношенні можуть бути аномалії, пов'язані з можливістю повторення значень атрибутів в декількох кортежі. Наприклад, те, що співробітник може працювати в декількох відділах, при звільненні

працівника вимагає відшукання і подальшого видалення з вихідної таблиці декількох записів.

Визначення п'ятої нормальної форми. Відношення R знаходиться в 5НФ в тому і тільки тому випадку, коли будь-яка залежність з'єднання в R випливає з існування деякого можливого ключа в R.

Утворити складові атрибути відношення СПІВРОБІТНИКИ-ВІДДІЛИ-ПРОЕКТИ:

СО = {Код_співробітника, Код_ відділу }

СП = {Код_співробітника, Номер_проекту}

ОП = {Код_відділу, Номер_проекту}.

Покажемо, що якщо відношення СПІВРОБІТНИКИ-ВІДДІЛИ-ПРОЕКТИ спроектувати на складові атрибути СО, СП і ОП, то поєднання цих проекцій дає початкове відношення. Це означає, що в нашому відношенні є залежність з'єднання *(СО, СП, ОП). Проекції на складові атрибути назвемо відповідно СПІВРОБІТНИКИ-ВІДДІЛИ, СПІВРОБІТНИКИ-ПРОЕКТИ і ВІДДІЛИ-ПРОЕКТИ.

Раніше ми виконували з'єднання двох проекцій і відразу отримували шуканий результат. Для відновлення відносини з трьох (або декількох) проекцій треба отримати всі попарні з'єднання (тому що інформація про те, яке з них «краще», відсутня), над якими потім виконати операцію перетину множин. Перевіримо, чи це так.

СПІВРОБІТНИКИ-ВІДДІЛИ

Код_ співробітника	Код_ відділу
01	РД
02	АТ
03	УП
04	АТ
05	ЛЗ

СПІВРОБІТНИКИ-ПРОЕКТИ

Код_ співробітника	номер_ проекту
01	036
02	004
03	004
04	019
05	001

ВІДДІЛИ-ПРОЕКТИ

Код_ відділу	Номер_ проекту
АТ	004
АТ	019
ВЦ	013
ВЦ	014
ЛЗ	001

05	ЛЗ
06	УП
08	ВЦ
09	ВЦ
10	СЖ

05	004
06	007
08	013
09	014
10	013

ЛЗ	004
СЖ	013
РД	036
УП	004
УП	007

Отримаємо попарні з'єднання трьох наведених вище відношень, котре матимуть вигляд:

* (СО, СП)

Код_ співробітника	Код_ відділу	Номер_ проекту
01	РД	036
02	АТ	004
03	УП	004
04	АТ	019
05	ЛЗ	001
05	ЛЗ	004
06	УП	007
08	ВЦ	013
09	ВЦ	014
10	СЖ	013

* (СО, ОП)

Код_ співробітника	Код_ відділу	Номер_ проекту
01	РД	036
02	АТ	004
02	АТ	019
03	УП	004
03	УП	007
04	АТ	004
04	АТ	019
05	ЛЗ	001
05	ЛЗ	004
06	УП	004
06	УП	007
08	ВЦ	013

* (СП, ОП)

Код_ співробітника	Код_ відділу	Номер_ проекту
01	РД	036
02	АТ	004
02	ЛЗ	004
02	УП	004
03	АТ	004
03	ЛЗ	004
03	УП	004
04	АТ	019
05	ЛЗ	001
05	АТ	004
05	ЛЗ	004
05	УП	004

08	ВЦ	014
09	ВЦ	013
09	ВЦ	014
10	СЖ	013

06	УП	004
06	УП	007
08	ВЦ	013
08	ВЦ	014
09	ВЦ	013
09	ВЦ	014
10	СЖ	013

Неважко побачити, що перетин всіх відносин дає вихідне відношення СПІВРОБІТНИКИ-ВІДДІЛИ-ПРОЕКТИ.

Відносини СПІВРОБІТНИКИ-ВІДДІЛИ, СПІВРОБІТНИКИ-ПРОЕКТИ і ВІДДІЛИ-ПРОЕКТИ знаходяться в 5НФ. Ця форма є останньої з відомих. Умови її отримання досить нетривіальні і тому вона майже не використовується на практиці. Більш того, вона має певні недоліки.

Припустимо, необхідно дізнатися, де і які проекти виконує співробитник з кодом 02. Для цього в відношенні СПІВРОБІТНИКИ-ВІДДІЛИ знайдемо, що співробітник з кодом 02 працює у відділі АТ, а з відносини ВІДДІЛИ-ПРОЕКТИ знайдемо, що в відділі АТ виконуються проекти 004 і 019. А це означає, що співробітник 02 повинен виконувати проекти 004 і 019. На жаль, інформація про те, що співробітник з кодом 02 виконує проект 019, є хибною (див. вихідне відношення СПІВРОБІТНИКИ-ВІДДІЛИ-ПРОЕКТИ).

Такі протиріччя можна усунути тільки шляхом спільного розгляду всіх проекцій основного відношення. Саме тому після з’єднання проекцій і виконувалася операція їх пересічення.

На практиці зазвичай обмежуються структурою БД, відповідної 3НФ. Тому процес нормалізації вставлений методом нормальних форм передбачає послідовне видалення з вихідного відношення наступних між атрибутних залежностей:

- часткових залежностей не ключових атрибутів від ключа (задоволення вимог 2НФ);

- транзитивних залежностей ключових атрибутів від ключа (задоволення вимог 3НФ);
- залежність ключів (атрибутів складових ключів) від не ключових атрибутів (задоволення вимог БКНФ).

4.8. Забезпечення цілісності

Під цілісністю розуміють властивість бази даних, що означає, що вона містить повну, несуперечливу предметну область інформацію. Розрізняють фізичну і логічну цілісність.

Фізична цілісність означає наявність фізичного доступу до даних і те, що дані не втрачені.

Логічна цілісність означає відсутність логічних помилок в базі даних, до яких відносяться порушення структури БД або її об'єктів, видалення або зміна встановлених зв'язків між об'єктами і т. д.

В подальшому мова будемо вести про логічну цілісності.

Підтримка цілісності БД включає перевірку цілісності і її відновлення в разі виявлення суперечностей в базі. Цілісний стан БД задається за допомогою обмежень цілісності у вигляді умов, яким повинні задовольняти в базі дані.

Серед обмежень цілісності можна виділити два основних типи обмежень: обмеження значень атрибутів відносин і структурні обмеження на кортежі відносин.

Прикладом *обмежень значень атрибутів відносин* є вимоги не припустимості порожніх або повторюваних значень в атрибутах, а також контроль приналежності значень атрибутів заданому діапазону. Так, в записах відносин про кадрах значення атрибута Дата_народження ні можуть перевищувати значення атрибута Дата_прийому.

Найбільш гнучким засобом реалізації контролю значень атрибутів є збережені процедури і тригери, наявні в деяких СУБД.

Структурні обмеження визначають вимоги цілісності сутності і цілісності посилань. Кожному екземпляру сутності, представленому в відношенні, відповідає тільки один його кортеж. Вимога цілісності сутностей полягає в тому, що будь-який кортеж відносини повинен бути відмінним від будь-якого іншого кортежу цього відношення, т.е. іншими словами, будь-яке відношення повинен володіти первинним ключем.

Формулювання вимоги цілісності посилань тісно пов'язана з поняттям зовнішнього ключа. Нагадаємо, що зовнішні ключі служать для зв'язку відношень (таблиць БД) між собою. При цьому атрибут одного відношення називається зовнішнім ключем даного відношення, якщо він є первинним ключем іншого ставлення (дочірнього). Кажуть, що відношення, в якому визначений зовнішній ключ, посилається на відношення, в якому цей же атрибут є первинним ключем.

Вимога цілісності посилань полягає в тому, що для кожного значення зовнішнього ключа батьківської таблиці повинен знайтися рядок в дочірній таблиці з таким же значенням первинного ключа. Наприклад, якщо в відношенні R1 (рис. 4.7) містяться відомості про співробітників кафедри, а атрибут цього відношення повинен бути первинним ключем відношення R2, то в цьому відношенні для кожної посади з R1 повинен бути рядок з відповідальним їй окладом. У багатьох сучасних СУБД є засоби забезпечення контролю цілісності БД.

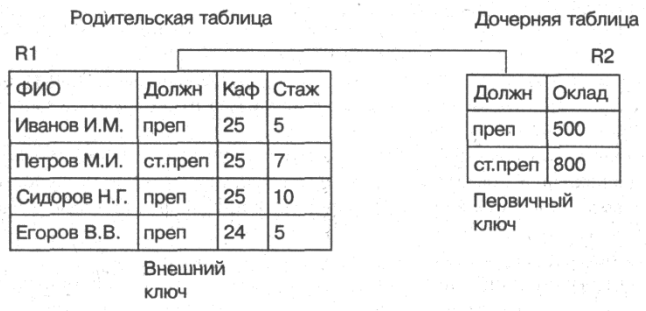


Рисунок 4.7. Зв'язок відносин за допомогою зовнішнього ключа

Контрольні питання.

1. Назвіть підходи до проектування структур даних.
2. У чому полягає надмірне і не надлишкове дублювання даних?
3. Назвіть і охарактеризуйте основні види аномалій.
4. Як формується вихідне відношення при проектуванні БД?
5. Наведіть приклади явної і неявної надмірності.
6. Назвіть основні види залежностей між атрибутами відношень.
7. Наведіть приклади функціональної і частково функціональної залежностей.
8. Наведіть приклади відносин з залежними атрибутами.
9. Охарактеризуйте нормальні форми.
10. Дайте визначення першої нормальної форми.
11. Дайте визначення другої нормальної форми.
12. Дайте визначення третьої нормальної форми.
13. Дайте визначення посиленою третьої нормальної форми.
14. Поясніть на прикладі що використовуються в розділі таблиць вимоги 4НФ.
15. Поясніть на прикладі що використовуються в розділі таблиць вимог 5НФ.
16. Дайте визначення фізичної та логічної цілісності БД.
17. Наведіть приклади обмежень значень і структурних обмежень.
18. Поясніть поняття зовнішнього і первинного ключів таблиць.

ЛЕКЦІЯ. КОНЦЕПТУАЛЬНЕ ПРОЕКТУВАННЯ БАЗ ДАНИХ

План:

1. Концептуальні моделі.
2. Модель «сутність-зв’язок».
3. Розширена модель "сутність – зв'язок".
4. Проблеми побудови моделей "сутність – зв'язок".
5. Приклад побудови моделі "сутність – зв'язок".

Література

1. Гайна Г.А. Основи проектування баз даних: Навчальний посібник. К.: КНУБА, 2005. – 204 с.
2. Гайна Г.А. Організація баз даних і знань. Мови баз даних: Конспект лекцій.−К.:КНУБА, 2002. – 64 с.
3. *Гайна Г.А., Попович Н.Л.* Організація баз даних і знань. Організація реляційних баз даних: Конспект лекцій.−К.:КНУБА, 2000. – 76 с.

1. Концептуальні моделі.

З концептуального проектування починається створення концептуальної схеми БД, в основі якої лежить концептуальна модель даних. Концептуальна модель представляє загальний погляд на дані. Розрізняють два головних підходи до моделювання даних при концептуальному проектуванні:

− семантичні моделі;

− об'єктні моделі.

Семантичні моделі головну увагу приділяють структурі даних. Найбільш поширеною семантичною моделлю є модель "сутність – зв'язок" (Entity Relationship model, ER-модель). ER-модель складається із сутностей, зв'язків, атрибутів, доменів атрибутів, ключів. Моделювання даних відображає логічну структуру даних, так само, як блок-схеми алгоритмів відображають логічну структуру програми.

Об'єктні моделі головну увагу приділяють поведінці об'єктів даних і засобам маніпуляції даними. Головне поняття таких моделей − об'єкт, тобто сутність, яка має стан і поведінку. Стан об'єкта визначається сукупністю його атрибутів, а поведінка об'єкта визначається сукупністю операцій специфікованих для нього.

Зближення цих моделей реалізується в розширеному ER-моделюванні (Extended Entity Relationship model, EER-модель).

2. Модель сутність-зв’язок.

ER-моделювання являє собою низхідний підхід до проектування БД, який починається з визначення найбільш важливих даних, які називаються сутностями (entities), і зв'язків (relationships) між даними, які повинні бути представлені в моделі. Потім в модель заноситься інформація про властивості сутностей і зв'язків, яка називається атрибутами (attributes), а також всі обмеження, які відносяться до сутностей, зв'язків і атрибутів. ER-модель дає графічне представлення логічних об'єктів і їх відношень в структурі БД. Послідовність проведення ER-моделювання показана на рис. 1.

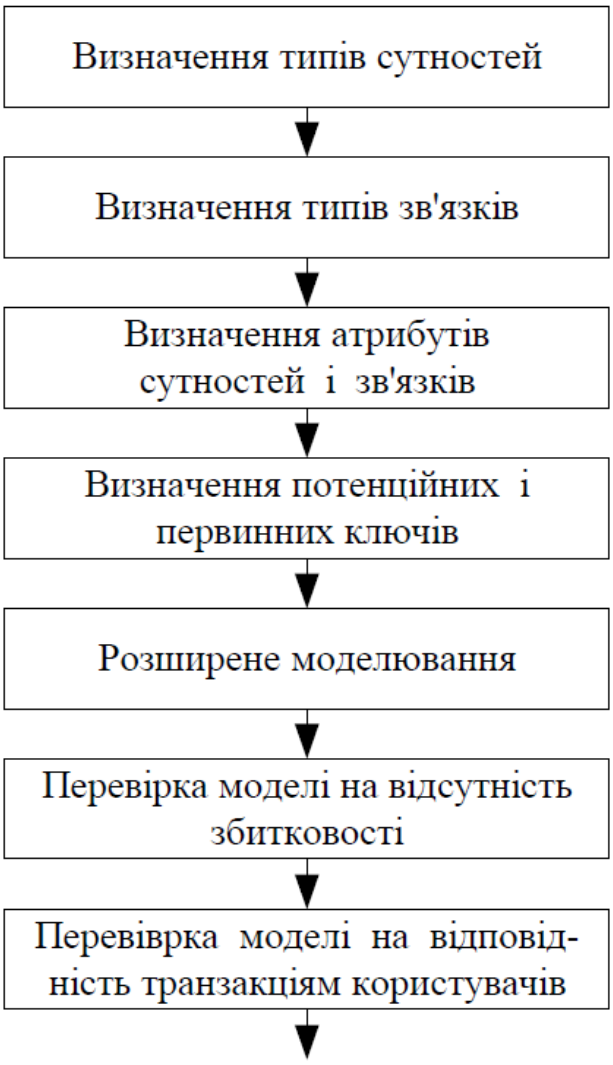


Рис. 1. Етапи побудови моделі "сутність – зв'язок".

Вперше поняття ER-моделі запровадив П.Чен. Підхід П.Чена дозволив концептуальне моделювання перевести в практичну площину проектування БД. У подальшому діаграми Чена набули розвитку у багатьох інших роботах з ER-моделювання. До них належать такі моделі:

− "пташина лапка", розроблена К.М. Бахманом;

− IDEF1X, розроблена Т.Ремеєм;

− на основі UML;

− модель Баркера і багато інших моделей.

Сутності

Сутність дозволяє моделювати клас однотипних об'єктів. Сутність має унікальне ім'я у межах системи, що моделюється. Оскільки сутність відповідає деякому класу однотипних об'єктів, то передбачається, що в системі існує багато екземплярів даної сутності. Об'єкт, якому відповідає сутність, має набір атрибутів, які характеризують його властивості. При цьому набір атрибутів повинен бути таким, щоби можна було розрізняти конкретні екземпляри сутності.

Приклад. Сутність Викладач може мати такі атрибути:

Табельний номер, Прізвище, Ім'я, По батькові, Посада, Вчений ступінь.

Набір атрибутів, що однозначно ідентифікує конкретний екземпляр сутності, називають ключовим. У наведеному прикладі для сутності Викладач ключем буде Табельний номер, оскільки для всіх викладачів табельні номери різні. Екземпляром сутності Викладач буде опис конкретного викладача. Загальноприйняте позначення сутності − прямокутник (рис. 2).

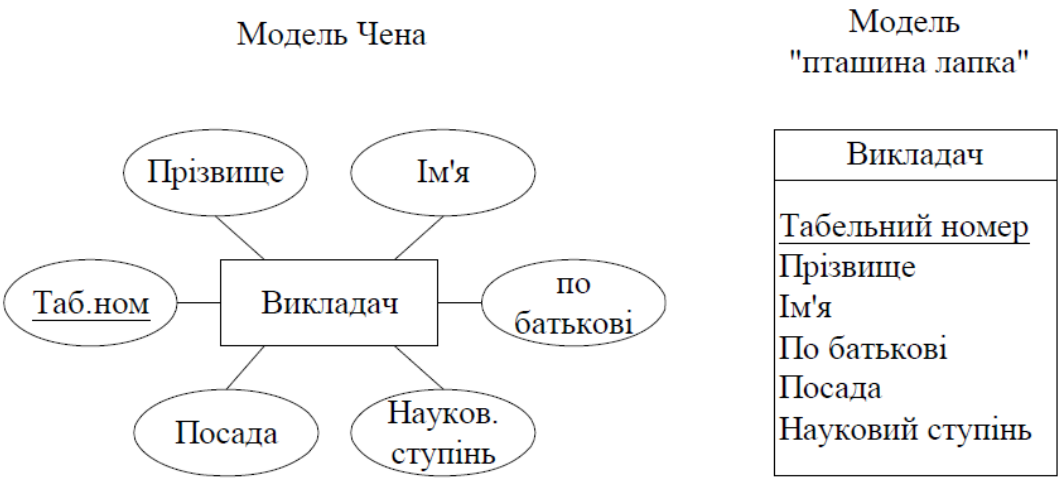


Рис. 2. Представлення сутностей і атрибутів у ER-діаграмах П. Чена і ER-діаграмах "пташина лапка"

Зв’язки

Між сутностями встановлюються зв'язки, які вказують яким чином сутності співвідносяться або взаємодіють між собою. Розрізняють такі зв'язки:

− між двома сутностями (бінарний зв'язок);

− між трьома сутностями (тернарний зв'язок);

− між N сутностями (N-арний зв'язок);

− між однією сутністю (рекурсивний зв'язок).

Найбільш поширеними є бінарні зв'язки. Зв'язок показує яким чином екземпляри сутностей зв'язані між собою. Бінарні зв'язки бувають:

− 1:1 (один до одного);

− 1:M (один до багатьох);

− N:M (багато до багатьох).

На рис. 3, 4 показані відображення цих зв'язків у різних ER-моделях.

Зв'язок "один до одного" (1:1): завідуючий кафедрою може керувати тільки однією кафедрою, а кожною кафедрою керує тільки один завідуючий

а

Зв'язок "один до багатьох" (1:М): на кафедрі працює багато викладачів, а кожен викладач працює тільки на одній кафедрі

б

Рис. 3. Представлення зв'язків між відношеннями на діаграмі Чена: а −1:1; б − 1:М

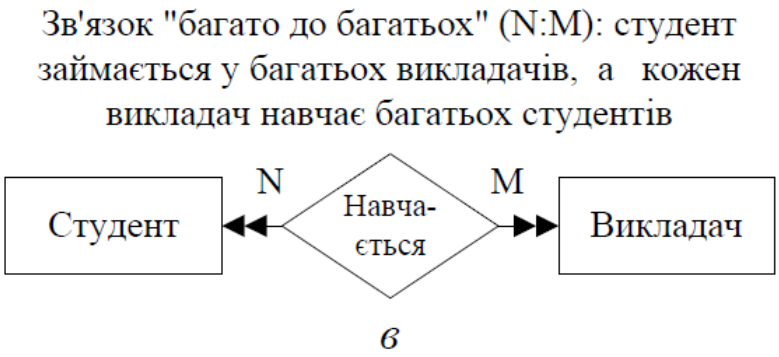


Рис. 5.3. Закінчення: *в* – N:M

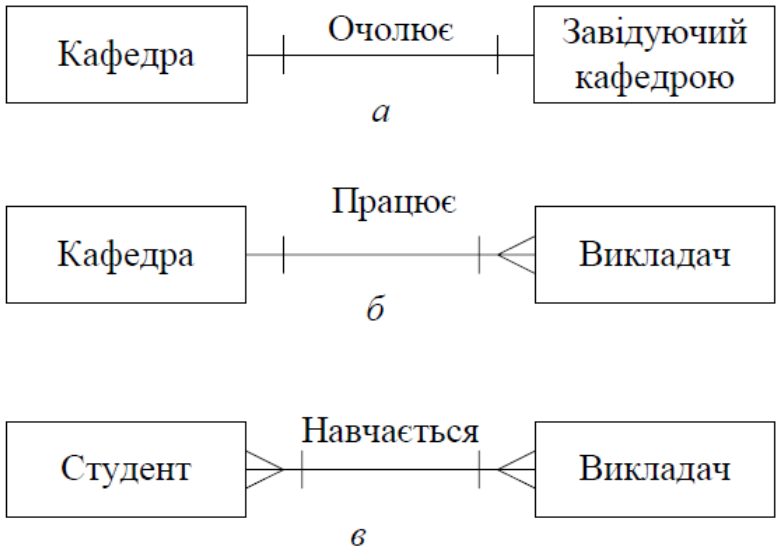


Рис. 4. Представлення зв'язків між відношеннями на діаграмі "пташина лапка": а −1:1; б − 1:M; в − N:M

Атрибути

Атрибути являють собою властивості сутності. Значення кожного атрибута вибирають з відповідної множини значень, яка включає всі потенційні значення, які можуть бути присвоєні атрибуту. Ця множина значень називається доменом.

Приклад. Атрибут Оцінка може приймати чотири значення: 2, 3, 4, 5. Ці значення і складають домен цього атрибута. Атрибути залежно від складності значень, які вони можуть приймати поділяються на певні категорії (табл. 1).

Таблиця 1

Типи атрибутів

Тип	**Властивість**
Простий	Атрибут, який не може бути поділений

	на інші атрибути. Приклад. Прізвище; посада
Складовий	Атрибут, який може бути поділений на інші атрибути. Приклад. Адреса; прізвище, ім'я, по батькові
Однозначний	Атрибут, який може приймати тільки одне значення. Приклад. Табельний номер, номер залікової книжки
Багатозначний	Атрибут, який може приймати багато значень. Приклад. Телефон, Адреса (постійне місце проживання і гуртожиток)
Похідний	Атрибут, який не зберігається в БД, а обчислюється за допомогою певного алгоритма Приклад. Вік (обчислюється по даті народження), кількість студентів в групі

Приклад. Розглянемо сутність *Студент* (рис. 5).

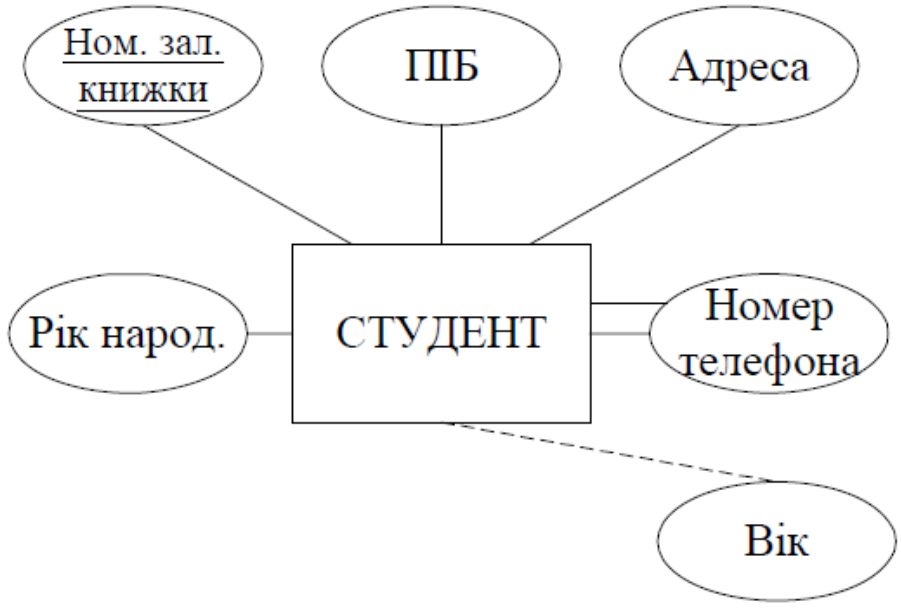


Рис. 5. ER-діаграма сутності Студент

Атрибути номер залікової книжки, рік народження є простими.

Атрибути ПІБ і Адреса є складовими. ПІБ може бути поділений на атрибути: прізвище, ім'я, по батькові, а Адреса − на індекс, місто, вулиця, будівля, квартира.

Атрибут Вік є похідним: він обчислюється за значенням атрибута Рік народження (зображається пунктирною лінією).

Атрибут Номер залікової книжки є однозначним: він не може приймати два значення для одного студента.

Атрибут Номер телефону є багатозначним: він може приймати декілька значень для одного студента (зображається подвійною лінією).

Атрибут або набір атрибутів сутності, які застосовуються для ідентифікації екземпляра сутності, називаються потенційним ключем. Сутність може містити декілька потенційних ключів. В прикладі в якості потенційних ключів можуть бути такі атрибути: Номер залікової книжки, Прізвище Ім'я по Батькові.

Потенційний ключ, який вибрано для однозначної ідентифікації кожного екземпляра сутності певного типу, називається первинним ключем. Первинний ключ вибирається за умови гарантії унікальності його значень, а також мінімальної довжини атрибутів, які входять в його склад. В прикладі в якості первинного ключа служить Номер залікової книжки.

Потужність зв'язків.

Потужність зв'язку (кардинальність) відображає певне число екземплярів сутностей, які зв'язані з одним екземпляром зв'язаної сутності. В моделі Чена потужність зв'язку відображається вказівкою відповідних чисел поруч з сутностями у форматі (x, y). Перше число визначає мінімальне значення потужності зв'язку, а друге − його максимальне значення. Потужність вказує на число екземплярів у зв'язаній сутності.

Відомості про максимальне і мінімальне значення потужності зв'язку може застосовуватися у прикладному програмному забезпеченні або за допомогою тригерів; на рівні таблиць СУБД не може оперувати з потужністю зв'язків.

Приклад. Розглянемо зв'язок Група-Студент (рис. 6).

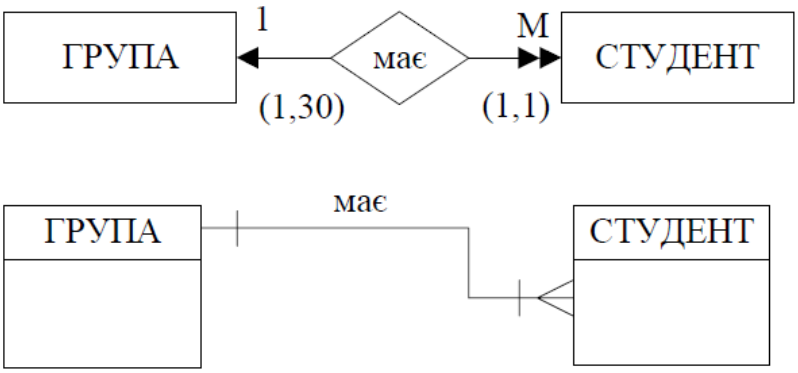


Рис. 6. Зв'язок і потужність в ER-діаграмах

Потужність (1,30) поруч із сутністю Група вказує, що в групі може займатися від 1 до 30 студентів. Потужність (1,1) поруч із сутністю Студент вказує, що студент може займатися тільки в одній групі (мінімум 1, максимум 1). У моделі "пташина лапка" числовий діапазон значень потужності не відображається в ER-діаграмах.

Сильні і слабкі зв'язки.

Якщо сутність може існувати незалежно від інших сутностей, то вона є незалежною від існування. Якщо сутність залежить від існування інших сутностей, то вона є залежною від існування. Наприклад, сутності Студент і Група можуть існувати незалежно одна від одної, а сутність Нагорода студента є залежною від сутності Студент й існувати без неї не може.

Якщо одна сутність незалежна від існування іншої сутності, то зв'язок між ними називається не ідентифікаційним зв'язком або слабким зв'язком. На ER-діаграмах "пташина лапка" слабкий зв'язок відображається штриховою лінією.

Ідентифікаційний зв'язок або сильний зв'язок має місце у тому випадку, коли одна зв'язана сутність залежить від існування іншої. На ER-діаграмах "пташина лапка" сильний зв'язок відображається суцільною лінією.

Приклад. Розглянемо зв'язки Група-Студент і Студент-Нагорода (рис. 7).

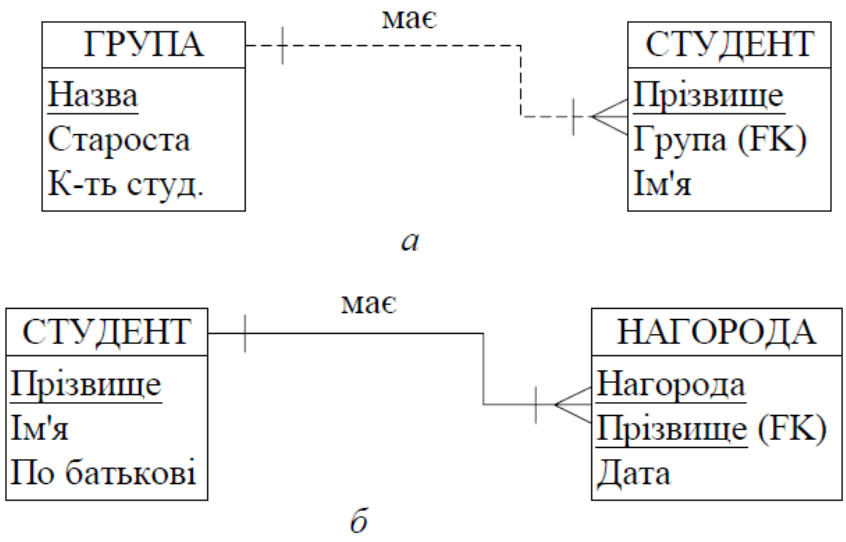


Рис. 7. Неідентифікаційний (а) та ідентифікаційний (б) зв'язки

Сутність Студент не залежить від сутності Група, в цьому випадку зв'язок між сутностями відображається штриховою лінією, а атрибут Назва групи в сутності Студент є зовнішнім ключем (Foreign Key, FK).

Сутність Нагорода залежить від сутності Студент, в цьому випадку зв'язок між сутностями відображається суцільною лінією, а атрибут Прізвище студента в сутності Нагорода є частиною первинного ключа (Primary Key, PK) і одночасно зовнішнім ключем (FK).

Атрибути зв'язків.

Так само як і сутності зв'язки можуть мати атрибути.

Приклад. Атрибути День і Аудиторія належать до зв'язку між сутностями Студент і Дисципліна (рис. 8).

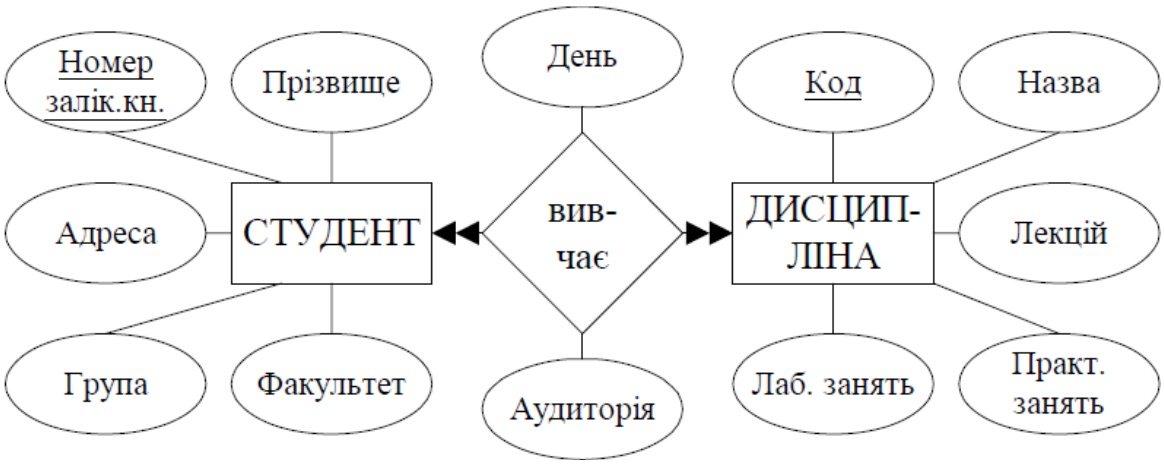


Рис. 8. ER-діаграма з атрибутами зв'язку День і Аудиторія

Обов'язкові і необов'язкові зв'язки.

Участь сутності у зв'язку може бути обов'язковою або необов'язковою. Якщо один екземпляр сутності не потребує наявності відповідного екземпляра сутності в окремому зв'язку, то участь сутності у зв'язку є необов'язковою. Наприклад, у зв'язку сутностей Студент-Нагорода − не кожен студент має нагороди. Тобто не кожен екземпляр в таблиці Студент потребує обов'язкової наявності екземпляра сутності в таблиці Нагорода. Сутність Нагорода розглядається як необов'язкова по відношенню до сутності Студент. Необов'язкова сутність позначається невеликим колом з боку необов'язкової сутності. Існування необов'язковості вказує на те, що для необов'язкової сутності мінімальне значення потужності зв'язку дорівнює 0.

Участь сутності у зв'язку буде обов'язковою, якщо кожен екземпляр сутності обов'язково потребує відповідного екземпляра сутності в окремому зв'язку. При обов'язковому зв'язку для обов'язкової сутності мінімальна потужність зв'язку дорівнює 1.

Приклад. Зв'язок між сутностями Студент і Нагорода є необов'язковим (рис. 9). Необов'язково кожен студент має нагороду, але якщо є нагорода, то вона обов'язково зв'язана з певним студентом.

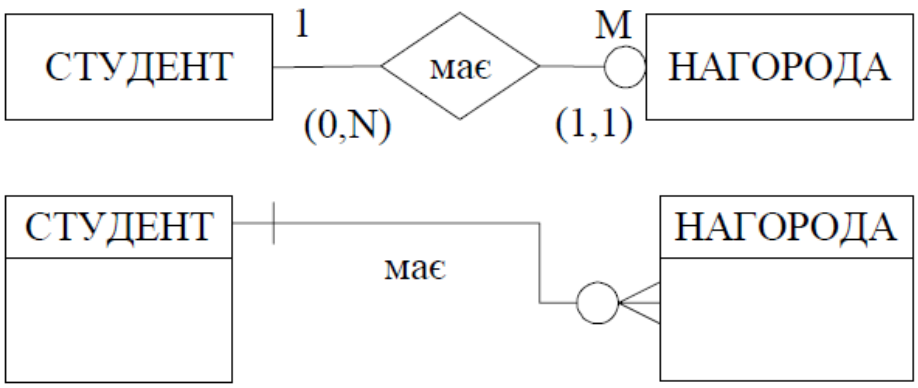


Рис. 9. Необов'язкова сутність Нагорода в ER-діаграмах П. Чена і "пташина лапка"

Слабкі сутності.

Слабкою сутністю називається сутність, яка задовольняє таким умовам:

− залежності від існування сутності з якою вона зв'язана;

− первинний ключ цієї сутності частково або повністю отриман з іншої сутності.

Слабка сутність на діаграмі Чена позначається подвійним прямокутником, а на діаграмі "пташина лапка" невеликими сегментами в кожному з кутів прямокутника.

Приклад. Сутність Нагорода є слабкою по відношенню до сутності Студент: вона залежить від існування цієї сутності і в її первинний ключ входить первинний ключ сутності Студент (рис. 10).

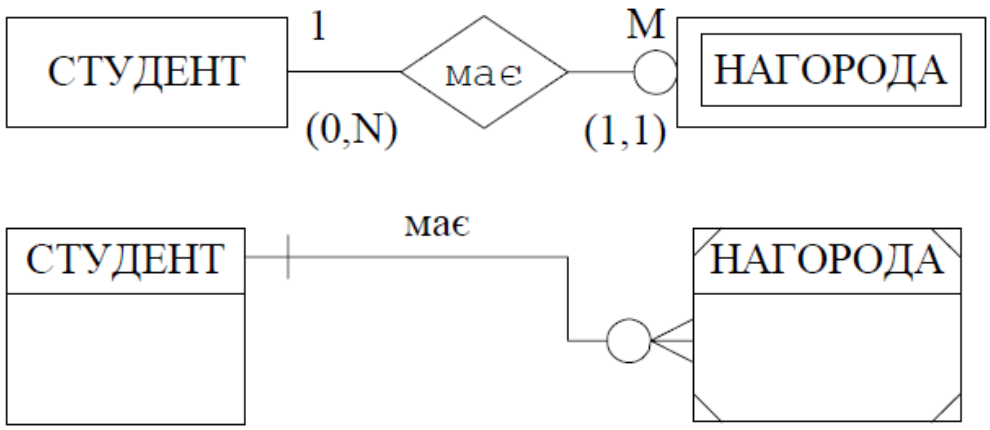


Рис. 10. Слабка сутність на діаграмах П. Чена і "пташина лапка"

Складні зв'язки.

Використання зв'язків більш високого порядку дає можливість у багатьох випадках краще відобразити семантику проблемної області.

Приклад. Сутності Викладач, Дисципліна і Екзамен утворюють тернарний зв'язок (рис. 11).

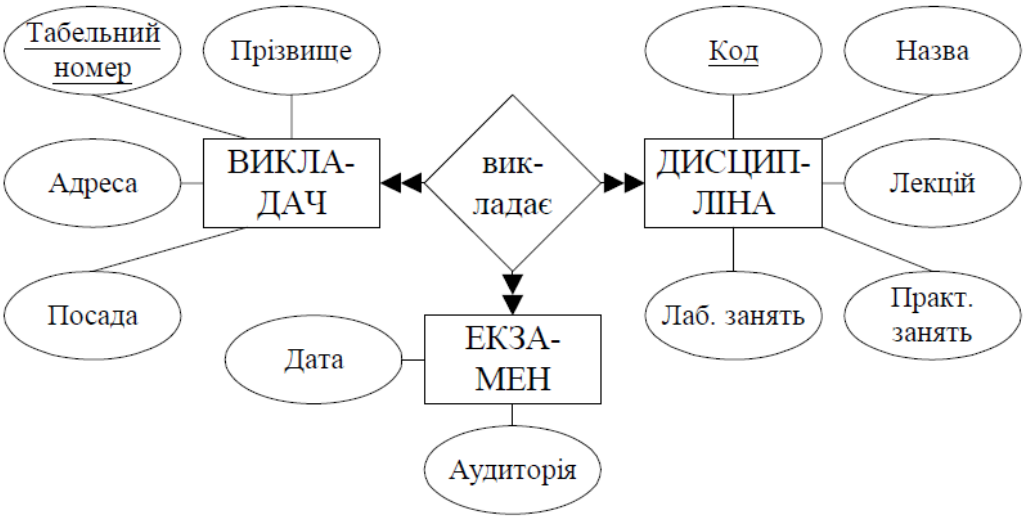


Рис. 11. Тернарний зв'язок між трьома сутностями на діаграмі П. Чена

Рекурсивні зв'язки/

Рекурсивний зв'язок має місце, коли є зв'язок між екземплярами одного і того ж набору сутностей.

Приклад. Розглянемо можливі варіанти рекурсивних зв'язків (рис. 12).

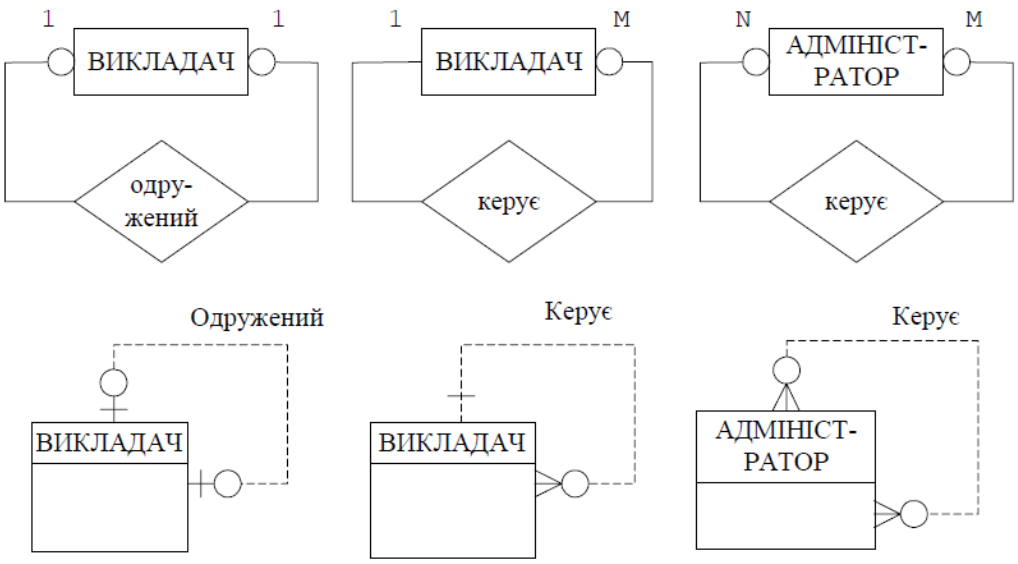


Рис. 12. Представлення рекурсивних зв'язків на діаграмі П. Чена і моделі "пташина лапка"

Зв'язок 1:1 представляє висловлювання: "викладач може бути одружений тільки з одним співробітником". Зв'язок 1:M представляє таке висловлювання: "викладач, якщо він є завідуючим кафедрою, керує декількома викладачами, а викладачі мають тільки одного керівника – завідуючого кафедрою". Зв'язок M:N представляє висловлювання: "адміністратор має декілька підлеглих-адміністраторів, і в свою чергу адміністратор має декілька керівників-адміністраторів".

Між двома сутностями може бути декілька зв'язків з різними змістовними навантаженнями.

Приклад. Між сутностями Викладач і Студент можна встановити такі змістовні зв'язки: Викладає і Керівництво дипломним проектуванням. Викладач викладає для багатьох студентів, для кожного студента викладає багато викладачів.

Кожен студент обов'язково повинен мати одного керівника дипломного проекту, але необов'язково кожен викладач керує дипломниками (рис. 13).

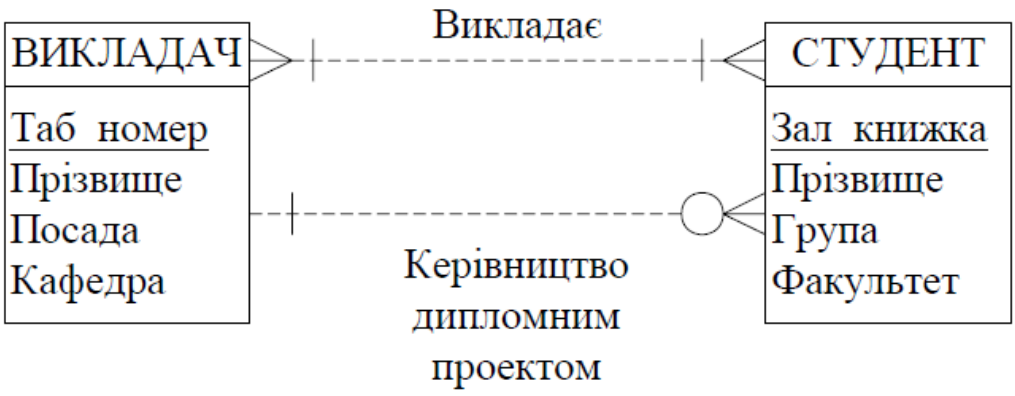


Рис. 13. Різні зв'язки між двома сутностями

3. Розширена модель "сутність – зв'язок".

Для задоволення нових потреб, що висуваються більш складними застосуваннями, в семантичне моделювання були введені додаткові концепції, що розширюють його можливості. Така модель має назву розширеної ER-моделі (Enhanced Entity Relationship, EER-модель). Вона включає всі концепції ER-моделі плюс концепції *уточнення*, *узагальнення*, *агрегування* і *композиції*. Додаткові концепції базуються на таких поняттях, як суперклас і підклас. Суперклас може мати декілька підкласів. Наприклад підкласи *Викладач*, *Керівник*, *Лаборант* є членами суперкласу *Співробітник*. Це означає, що кожен екземпляр підкласу є в той же час і екземпляром суперкласу. Зв'язок між суперкласом і підкласом відноситься до типу 1:1.

Використання понять суперклас і підклас дозволяє визначити для підкласів власні атрибути і атрибути, що наслідуються від суперкласу. Так, наприклад, підклас *Викладач* повинен мати ті ж атрибути, що і всі *Співробітники*. Однак він має і свої власні атрибути, які не визначені для інших категорій працівників університету. До цих атрибутів можна віднести вчене звання, номер диплому про вчене звання, кількість навчально-методичних праць тощо. При відсутності підкласів для об'єкту Співробітник слід було б вводити атрибути, які б мали невизначене значення для інших співробітників (наприклад для лаборантів). Підклас може мати свої власні зв'язки, які не підходять для всіх екземплярів суперкласу. Наприклад, *Викладач* може мати підкласи *Професор*, *Доцент*, *Асистент*. Підклас наслідує не тільки атрибути, але і всі зв'язки суперкласу.

Уточнення це процес збільшення різниці між окремими екземплярами об'єкта за рахунок визначення їхніх відмінних характеристик. Цей процес є низхідним. Наприклад, перехід від об'єкта Співробітник до об'єктів Викладач і Керівник.

Узагальнення це процес зведення відмінностей між об'єктами до мінімуму шляхом виділення їх спільних характеристик. Цей процес є висхідним. Наприклад, перехід від об'єктів *Викладач* і *Керівник* до об'єкта *Співробітник*.

У процесі проведення уточнення або узагальнення можуть застосовуватися обмеження:

− ступеня участі;

− неперетинання.

Підкласи набора сутностей можуть перетинатися і не перетинатися. Якщо підкласи суперкласу *не перетинаються*, то це означає, що кожен екземпляр сутності може бути елементом тільки одного з підкласів (позначається Or). Зв'язки, які не перетинаються позначаються символом "*G*". Наприклад, співробітник може працювати або на посаді доцента, або на посаді професора, і не може бути одночасно і професором, і доцентом.

Якщо підкласи суперкласу *перетинаються*, то це означає, що будь-який екземпляр сутності може бути елементом декількох з підкласів (позначається And). Зв’язки які перетинаються позначаються символом "Gs". Наприклад, завідуючий кафедрою проводить заняття, і одночасно виконує обов'язки викладача і керівника. На рис. 14 показана ієрархія сутностей з підкласами, що перетинаються і не перетинаються.

Приклад. Відношення суперклас – підклас.

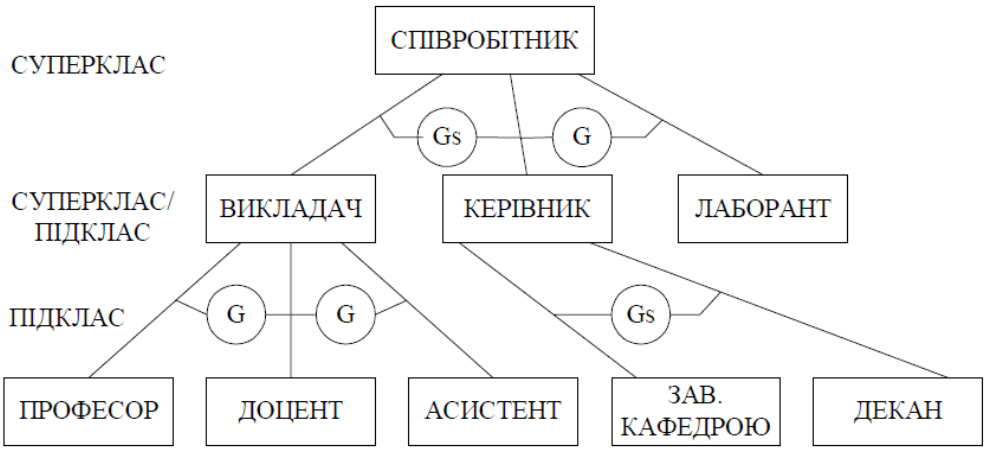


Рис. 14. Діаграма з використанням понять суперклас і підклас

Не кожен елемент суперкласу повинен бути елементом одного з підкласів.

Зв'язок суперкласу з підкласом з *обов'язковою участю*, вказує на те, що кожен елемент суперкласу повинен бути також елементом підкласу (*Mandatory*).

Приклад. Студент обов'язково займається або на денній, або на заочній, або на вечірній формі навчання, або екстернатом (рис. 15).

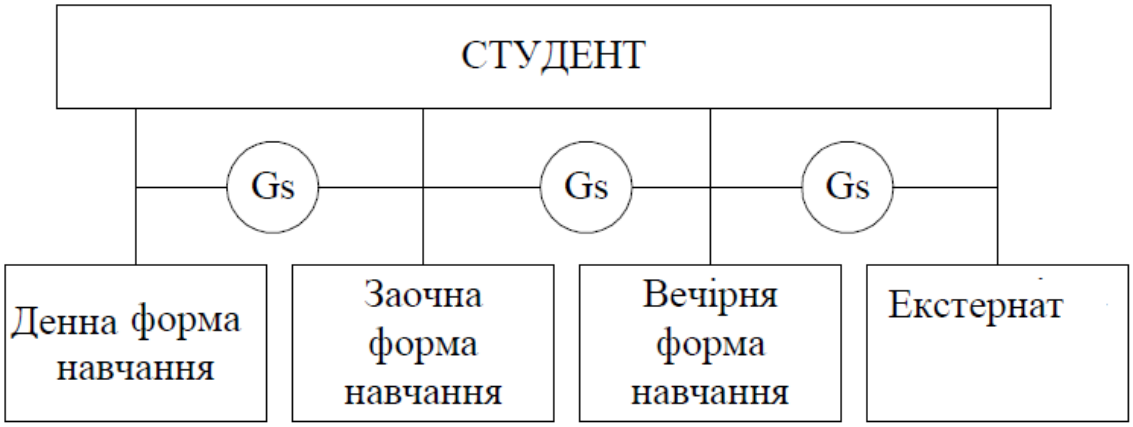


Рис. 15. Діаграма зв'язку суперкласу з підкласом з обов'язковою участю

Зв'язок суперкласу з підкласом з *необов'язковою участю*, вказує на те, що деякі елементи суперкласу можуть не належати жодному з підкласів (*Optional*). Наприклад, можуть бути викладачі, які не займають посади професора, доцента або асистента (наприклад старший викладач).

Введення понять суперкласів і підкласів дозволяє уникнути опису різних екземплярів сутності, які можуть мати різні атрибути. Це може привести до появи великої кількості незаповнених атрибутів. До того ж індивідуальні

атрибути можуть показати зв'язки, які притаманні тільки для конкретних екземплярів, а не всім екземплярам сутностей.

Приклад. Екземпляри сутності *Викладач*, які належать до викладачів на посаді професора, у порівнянні з викладачами на посаді асистента, мають такі додаткові атрибути: вчене звання, вчений ступінь і т.ін. Крім того, вони можуть бути зв'язані індивідуальними атрибутами із сутністю *Аспірант*.

Один підклас може бути зв'язаний з декількома суперкласами, які в свою чергу є підкласами одного спільного для них суперкласу.

Приклад. Підкласи *Викладач* і *Керівник* наслідують суперклас Співробітник. Підклас *Завідуючий кафедрою* є одночасно екземпляром суперкласу *Викладач* і суперкласу *Керівник* (рис. 16).

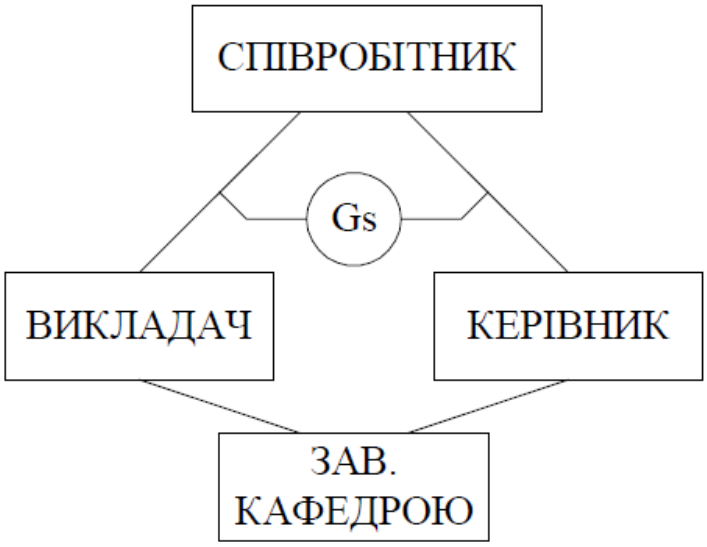


Рис. 16. Діаграма з підкласом, що сумісно використовується

Підклас є *категорією*, якщо він зв'язаний одразу з декількома суперкласами різних типів. Категорія може бути з повною участю і з частковою участю екземплярів. Повна участь передбачає, що кожен екземпляр всіх суперкласі повинен бути представлений у даній категорії. При частковій участі присутність в категорії всіх екземплярів всіх суперкласі необов'язкова.

Введення понять суперкласів і підкласів дозволяє ввести в проект більший обсяг семантичної інформації. Наприклад, твердження "Асистент є викладачем" дозволяє встановити певні ієрархічні відношення між об'єктами *Асистент* і *Викладач*. EER-діаграми повинні використовуватися, якщо структура даних є

занадто складною, для того щоби її можна було легко представити з використанням ER-діаграм.

Для моделювання інших видів зв'язків вводиться поняття агрегування і композиції.

Агрегування являє собою зв'язок типу "входить в склад" або "включає" між двома сутностями, одна з яких представляє "ціле", а інша − "частину".

Композиція − це особлива форма агрегування, яка представляє залежність між сутностями, що характеризуються повною приналежністю і співпаданням термінів існування між "цілим" і "частиною".

4. Проблеми побудови моделей "сутність – зв'язок".

При недостатньому розумінні суті встановлених зв'язків може бути створена модель, яка не буде повною мірою відображати зв'язки між реальними об'єктами. Визначають *дефекти з'єднання*, які виникають при невірній інтерпретації змісту деяких зв'язків: дефекти розгалуження і дефекти розриву.

Дефекти розгалуження мають місце, коли модель вірно відображає зв'язки між сутностями, але шлях між окремими сутностями визначений неоднозначно. Цей дефект виникає в тому випадку, коли два або більше зв'язків типу 1:M виходять з однієї сутності.

Приклад. Розглянемо такі зв'язки: на факультеті займається багато студентів, у склад факультету входить багато груп (рис. 17). Ці зв'язки вірно відображають зміст предметної області, але при спробі з'ясувати, в яких групах займаються конкретні студенти, виникають проблеми. Із сутності *Факультет* виходять два зв'язки 1:M.

Усунути цю проблему можна шляхом перебудови моделі для представлення вірної взаємодії цих сутностей (рис. 18).

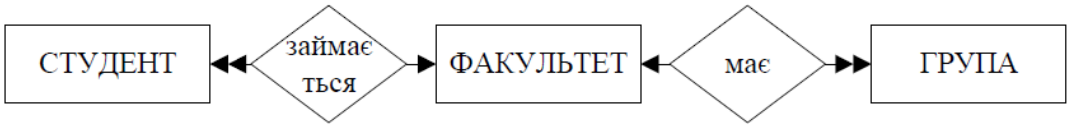


Рис. 17. Приклад дефекту розгалуження в ER-моделі

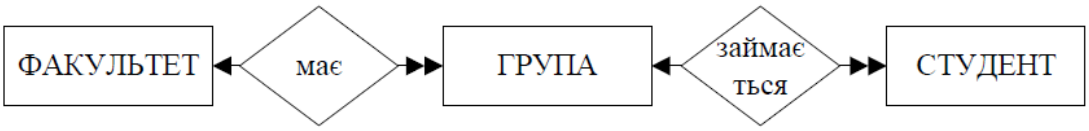


Рис. 18. Усунення дефекту розгалуження в прикладі ER-моделі

Отже, тепер відповідь на попереднє питання не є проблемою.

Дефекти розриву виникають у тому випадку, коли в моделі передбачається наявність зв'язку між декількома сутностями. Цей дефект виникає у разі, коли існує один або декілька зв'язків з мінімальною потужністю рівною 0, яка визначає необов'язкову участь, і ці зв'язки складають частину шляху між взаємозв'язаними сутностями.

Приклад. Розглянемо такі зв'язки: в склад факультету входить багато кафедр, кожна кафедра може відповідати за декілька комп'ютерних класів (від 0 до M) (рис. 19). Тобто деякі кафедри можуть не бути відповідальними за комп'ютерний клас. У свою чергу комп'ютерний клас може підпорядковуватися певній кафедрі, а може підпорядковуватися безпосередньо факультету (факультетський комп'ютерний клас). Ці зв'язки вірно відображають зміст предметної області, але при спробі з'ясувати, які комп'ютерні класи підпорядковані певному факультету, виникають проблеми. Зв'язок між сутностями *Кафедра* і *Комп'ютерний клас* передбачає необов'язкову участь сутностей, і він є частиною шляху між сутностями *Факультет* і *Кафедра*.

Усунути цю проблему можна шляхом введення додаткового зв'язку між сутностями *Факультет* і *Комп'ютерний клас* (рис. 20).

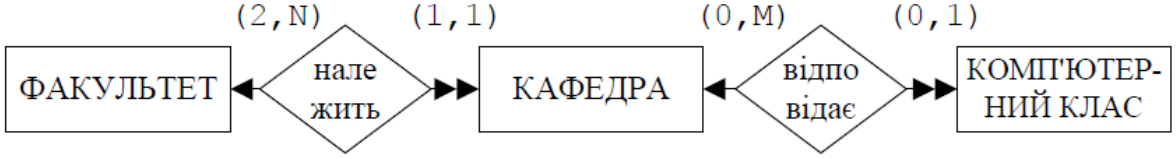


Рис. 19. Приклад дефекту розриву в ER-моделі

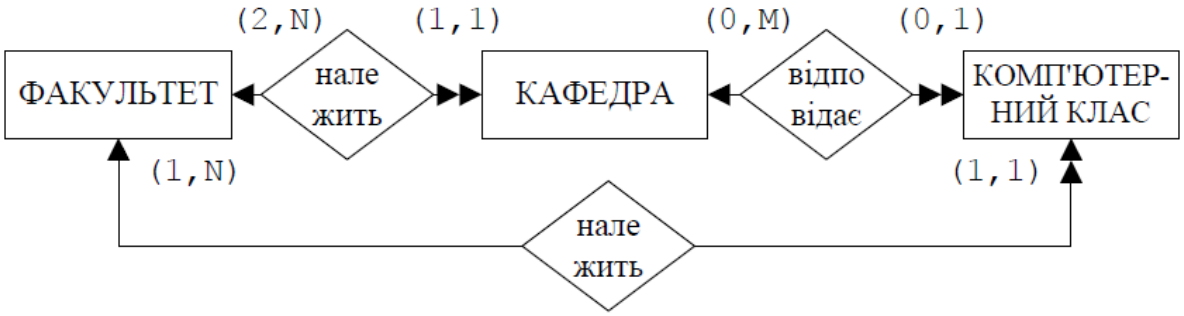


Рис. 20. Усунення дефекту розриву в прикладі ER-моделі

5. Приклад побудови моделі "сутність – зв'язок".

Процес концептуального проектування БД є ітеративний й заснований на операціях і процедурах, що повторюються. Спочатку створюється базова ER-модель певної предметної області. При дослідженні цієї моделі як правило з'являться додаткові сутності, атрибути і зв'язки. Після цього ER-модель буде змінюватися. Процес змін повторюється до тих пір, поки концептуальна модель не буде відображати предметну область.

Розробники БД на основі опитування фахівців визначають сутності, атрибути, зв'язки у предметній області, що розглядається, дослідженні документів, які застосовуються в роботі на підприємстві.

Розглянемо приклад створення ER-моделі предметної області ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД (ВНЗ).

Задачі інформаційної системи.

Мета створення бази даних полягає у такому:

− забезпечення адміністрації університету довідковою інформацією по факультетах, кафедрах, спеціальностях,

викладачах, студентах і дисциплінах, що викладаються у ВНЗ;

− контроль успішності студентів.

Аналіз предметної області.

У результаті дослідження й аналізу предметної області були визначені такі сутності, атрибути і первинні ключі (табл. 2).

Зв'язки сутностей визначаються на основі бізнес-правил,які побудовані з урахуванням організаційної структури та операцій, що виконуються в системі:

− в університеті існує декілька факультетів;

− на факультеті навчаються групи студентів за певними спеціальностями;

− у склад групи входять студенти;

− факультет об'єднує декілька кафедр;

− на кожній кафедрі працює декілька викладачів;

− на кожній спеціальності викладається ряд дисциплін, які проводять викладачі з різних кафедр;

− з кожної дисципліни своєї спеціальності студенти складають іспит або залік;

− не кожен викладач читає дисципліни (наприклад, асистент) і не з кожної дисципліни є викладач (наприклад, нова дисципліна, по якій викладача ще не призначено).

Таблиця 2

Сутності, атрибути і первинні ключі предметної області ВНЗ

Сутність	**Атрибути**	**Первинний ключ**
ФАКУЛЬТЕТ	*Код* *Назва* *Декан*	*Код факультету*
СПЕЦІАЛЬНІСТЬ	*Код* *Назва* *Вимоги*	*Код спеціальності*
ГРУПА	*Код* *Назва* *Кількість студентів* *Староста*	*Код групи*
СТУДЕНТ	*Номер залікової книжки* *Прізвище* *Адреса* *Рік народження*	*Номер залікової книжки*
КАФЕДРА	*Код* *Назва* *Завідуючий кафедрою* *Кількість викладачів*	*Код кафедри*
ВИКЛАДАЧ	*Табельний номер* *Прізвище* *Посада* *Науковий ступінь*	*Табельний Номер викладача*
ДИСЦИПЛІНА	*Код* *Назва* *Кількість годин* *Семестр*	*Код дисципліни*

Для спрощення концептуальної моделі бази даних цілий ряд об'єктів і бізнес-правил ВНЗ залишився нерозглянутим.

Дослідження предметної області виявило, що всі сутності є сильними, а зв'язки між сутностями − неідентифікуючими. Зв'язок між сутностями *Студент* і *Дисципліна* має атрибут *Оцінка*.

Побудова ER-діаграми.

На основі задач, які були поставлені перед інформаційною системою, і на основі аналізу предметної області, побудована ER-діаграма ЗВО (рис. 21).

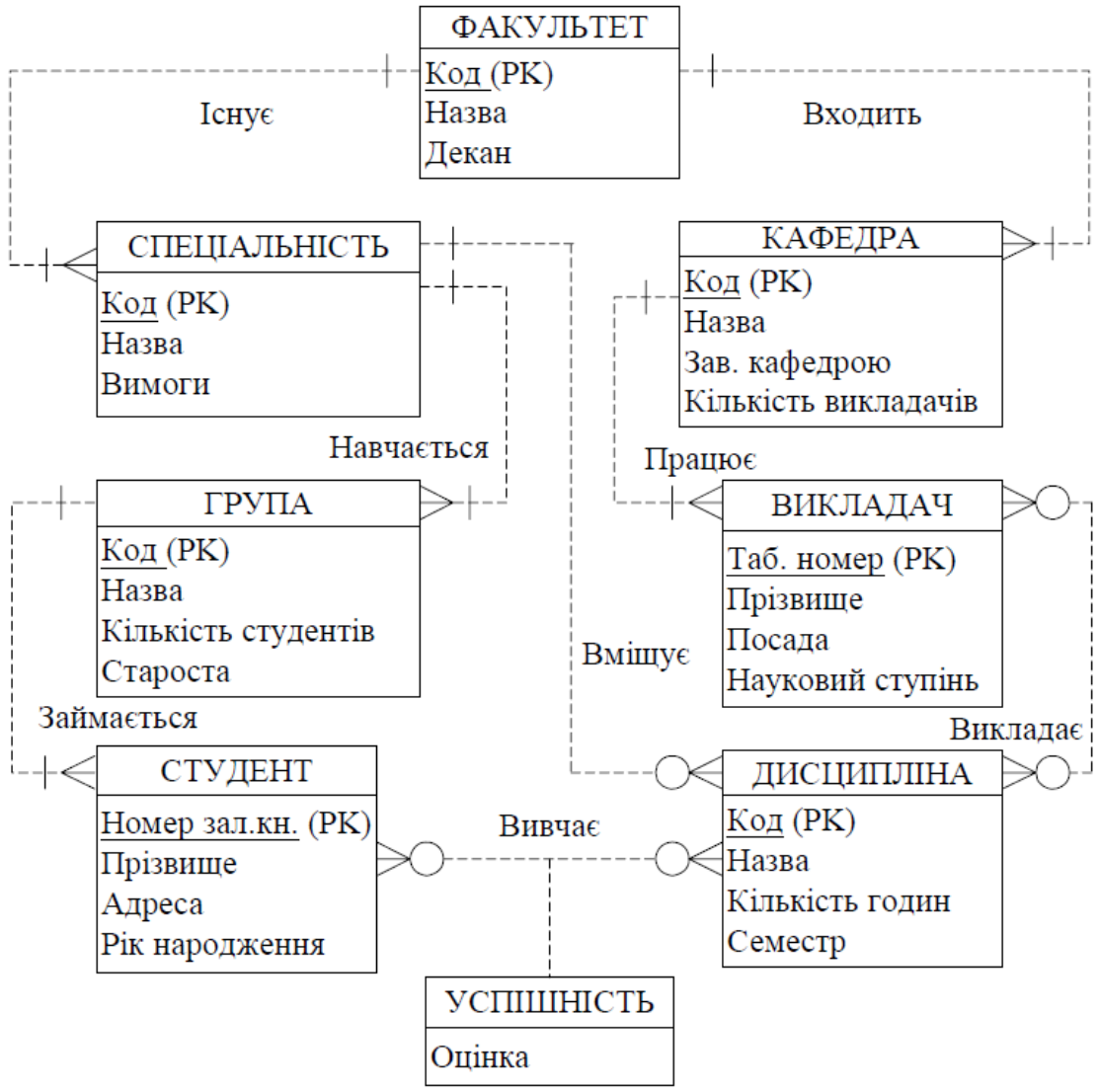


Рис. 21. ER-діаграма предметної області ЗВО

Розроблений концептуальний проект необхідно перевірити на збитковість та на відповідність транзакціям користувачів.

Перевірка на збитковість передбачає перевірку ER-моделі з метою виявлення збиткових даних і вилучення їх, в тому випадку, якщо вони визначені. Збиткові зв'язки виявляються в тому, що між двома сутностями є декілька шляхів і вони дублюють один одний (це не відноситься до зв'язків, які представляють різні асоціації).

Перевірка моделі на відповідність транзакціям користувачів виконується на основі таких підходів:

− перевірка того, чи представляє модель всю інформацію (сутності, атрибути, зв'язки), яка необхідна для кожної транзакції;

− перевірка по ER-діаграмі маршруту кожної транзакції.

Перевірка моделі на збитковість та на відповідність транзакціям користувачів дозволяє зробити висновок, що концептуальний проект відповідає всім необхідним вимогам.

Слід звернути увагу на те, що розроблений концептуальний проект не є єдиним проектом, який відповідає поставленій задачі. Можливі варіанти розробки системи із застосуванням інших зв'язків між сутностями, або із застосуванням розширеної ER-моделі.

Застосування ER-діаграм дозволяє забезпечити просте і наглядне уявлення про головні логічні об'єкти БД і про зв'язки, які між цими об'єктами існують. Також до переваг ER-діаграми слід віднести те, що вони добре інтегрують з реляційною моделлю.

Недоліком ER-моделей є те, що вони мають недостатні можливості для представлення відношень і обмежень, можуть бути складні при наявності багатьох об'єктів, не мають засобів для опису операцій маніпулювання даними.

Контрольні запитання

1. Дати визначення сутності. Що таке сильна сутність, слабка сутність?

2. Дати визначення атрибута. Що таке простий атрибут, складний атрибут, композитний атрибут?

3. Дати визначення ступеня зв'язку. Що таке кардинальне число?

4. Що таке ідентифікаційний і не ідентифікаційний зв'язок?

5. Навести символьні позначення, які застосовуються в діаграмах "сутність – зв'язок".

6. Пояснити, що таке підтипи сутностей і навести приклади.

7. Як відображається наслідування на діаграмах "сутність – зв'язок"?

8. Навести приклади зв'язків 1:N для таких різновидів зв'язків: необов'язково-необов'язково, необов'язково-обов'язково, обов'язково-необов'язково, обов'язково-обов'язково.

9. Дати визначення рекурсивного зв'язку і навести приклади рекурсивних зв'язків 1:1, 1:N, M:N.

10. Пояснити переваги і недоліки ER-моделі.

ЛЕКЦІЯ. ЛОГІЧНЕ ПРОЕКТУВАННЯ БАЗ ДАНИХ

План:

1. Етапи логічного проектування.
2. Спрощення концептуальної моделі.
3. Методика перетворення ER-діаграм в реляційні структури.
4. Перевірка відношень за допомогою правил нормалізації.
5. Перевірка відповідності відношень вимогам транзакцій користувачів.
6. Перевірка підтримки цілісності.
7. Приклад створення логічної моделі бази даних

Література

1. Гайна Г.А. Основи проектування баз даних: Навчальний посібник. К.: КНУБА, 2005. – 204 с.
2. Гайна Г.А. Організація баз даних і знань. Мови баз даних: Конспект лекцій.−К.:КНУБА, 2002. – 64 с.
3. *Гайна Г.А., Попович Н.Л.* Організація баз даних і знань. Організація реляційних баз даних: Конспект лекцій.−К.:КНУБА, 2000. – 76 с.

1. Етапи логічного проектування.

Логічне проектування виконується для певної моделі даних. Для реляційної моделі даних логічне проектування полягає у створенні реляційної схеми, визначенні числа і структури таблиць, формуванні запитів до БД, визначенні типів звітних документів, розробці алгоритмів обробки інформації, створенні форм для вводу і редагування даних в БД і рішенні цілого ряду інших задач. Концептуальні моделі за певними правилами перетворюються в логічні моделі даних. Коректність логічних моделей перевіряється за допомогою *правил нормалізації*, які дозволяють переконатися в структурній узгодженості, логічній цілісності і мінімальній збитковості прийнятої моделі даних. Модель також перевіряється з метою виявлення можливостей *виконання транзакцій*, які будуть задаватися користувачами. Проектування являє собою циклічний процес. Етапи логічного проектування наведені на рис. 1.

Рис. 1. Етапи логічного проектування бази даних

2. Спрощення концептуальної моделі.

Першим кроком спрощення концептуальної моделі є попередні перетворення з метою усунення зв'язків, які є несумісними з реляційною моделлю.

На цьому етапі виконуються такі операції:

− вилучення двосторонніх зв'язків M:N;

− вилучення складних зв'язків;

− вилучення багатозначних атрибутів;

− вилучення рекурсивних зв'язків;

− вилучення зв'язків з атрибутами.

Вилучення двосторонніх зв'язків "багато до багатьох"

Перетворення зв’язку "багато до багатьох" виконується шляхом введення проміжної сутності із заміною одного зв'язку M:N двома зв'язками 1:N з новою сутністю.

Приклад. Викладач може викладати багато *Дисциплін*, одну *Дисципліну* викладає багато *Викладачів* (рис. 2).

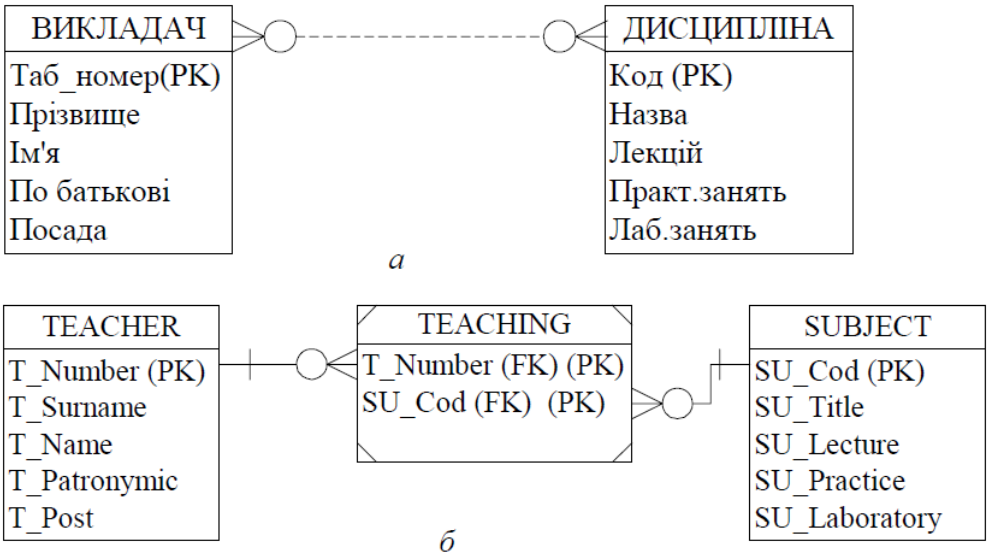


Рис. 2. Перетворення зв'язку "багато до багатьох": а – зв'язок M:N; б – результат перетворення − два зв'язку 1:N

У результаті перетворення отримана нова сутність, яка є слабкою і залежить від двох інших сутностей. Її первинний ключ складається з первинних ключів двох сутностей, а кожен атрибут окремо є вторинним ключем.

Вилучення складних зв'язків

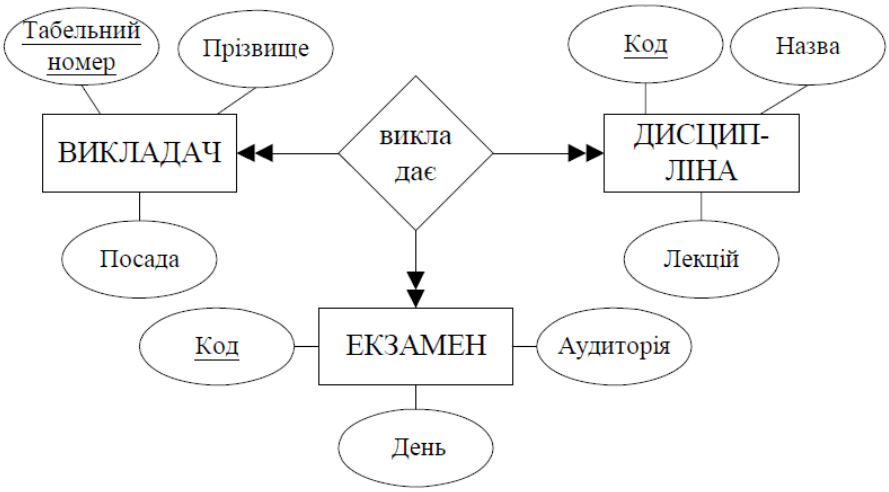
Для вилучення складних зв'язків виконуються такі операції:

− у модель вводиться нова сутність;

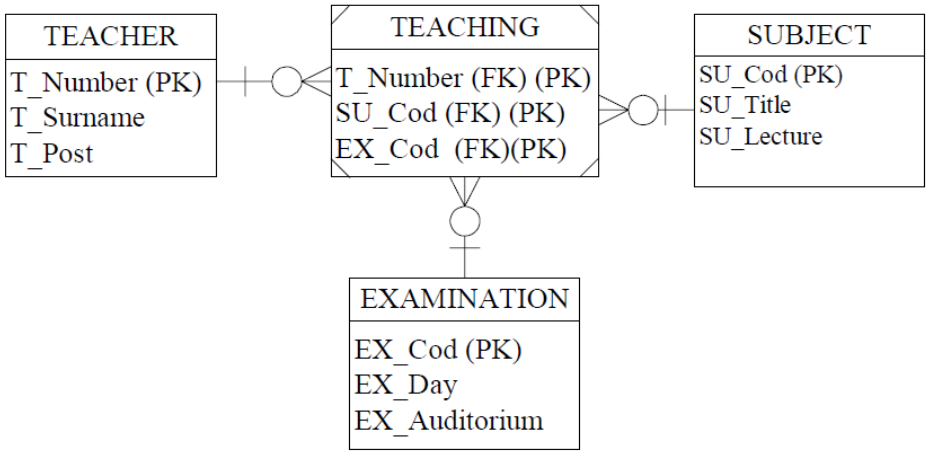
− складний зв'язок замінюється бінарними зв'язками "один до багатьох" зі знов створеною сутністю;

− кількість бінарних зв'язків дорівнює ступеню складності зв'язку.

Приклад. Викладач може викладати багато *Дисциплін*, одну *Дисципліну* викладає багато *Викладачів*. З *Дисципліни Викладач* проводить *Екзамен* (рис. 3).



а



б

Рис. 3. Представлення тернарного зв'язку бінарними зв'язками: а – складний зв'язок *Викладає*; б – декомпозиція складного зв'язку на три двосторонні зв'язки і нову сутність *Екзамен*

Вилучення багатозначних атрибутів

Якщо в концептуальній моделі даних присутній багатозначний атрибут, то може бути виконана декомпозиція цього атрибуту для визначення деякої сутності.

Приклад. Студент може мати декілька телефонів (рис. 4).

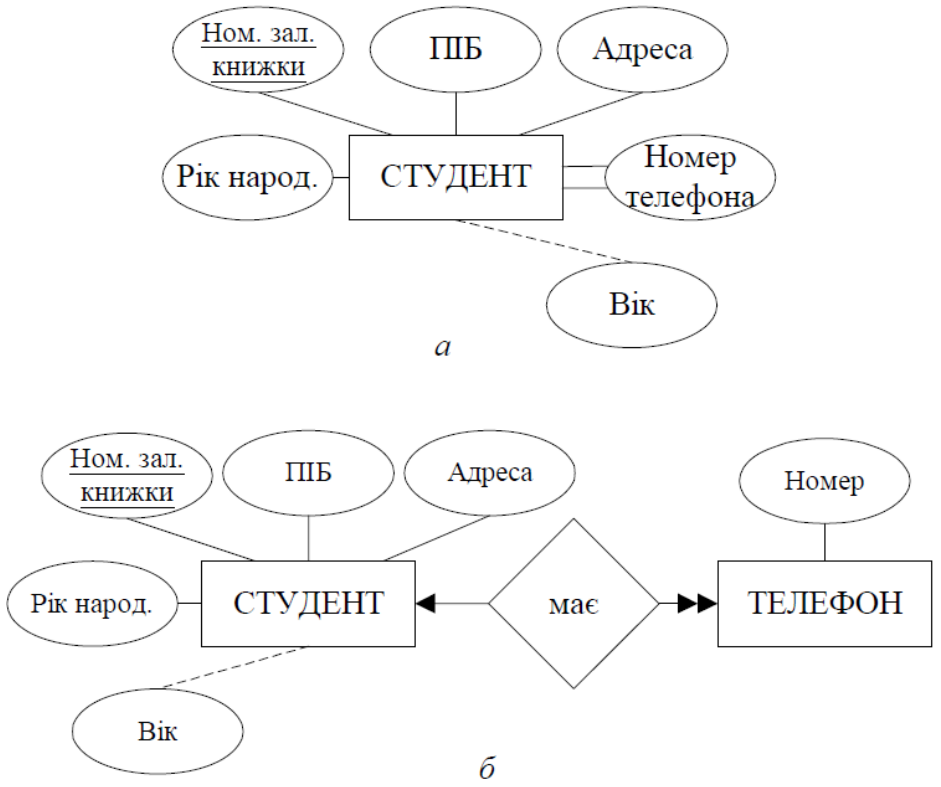


Рис. 4. Вилучення багатозначного атрибуту: а – сутність *Студент* з багатозначним атрибутом *Номер телефону*; б – нова сутність *Телефон*

Вилучення рекурсивних зв'язків

На етапі спрощення концептуальної моделі рекурсивні зв'язки 1:1 і 1:M (рис. 5) можуть бути перетворені у одне відношення. У випадку, коли є необов'язкова сутність з боку "багато" для зв'язку 1:M для зменшення пустих значень створюється нове відношення. Зв'язок M:N перетворюється на дві сутності (рис.6).

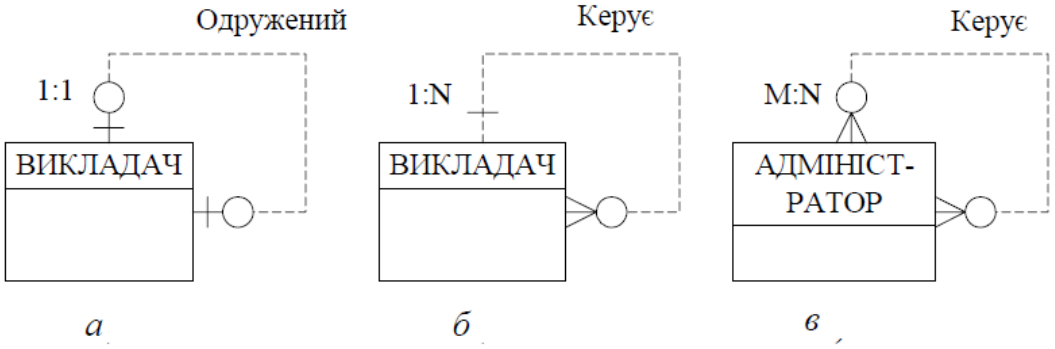


Рис. 5. Види рекурсивних зв'язків: а – 1:1; б – 1:N: в – M:N

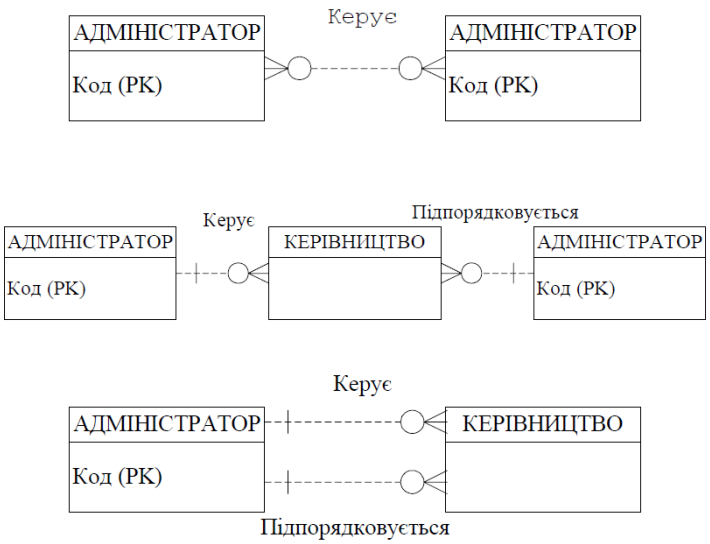


Рис. 6. Етапи послідовного перетворення рекурсивного зв'язку M:N

Вилучення зв'язків з атрибутами

Вилучення зв'язків з атрибутами виконується шляхом додавання у модель нової сутності для відношення M:N з атрибутами зв'язку. Для відношення 1:M атрибути зв'язку передаються у сутність "багато" без створення нової сутності.

Приклад. Розглянемо сутності *Студент-Дисципліна* (рис.7).

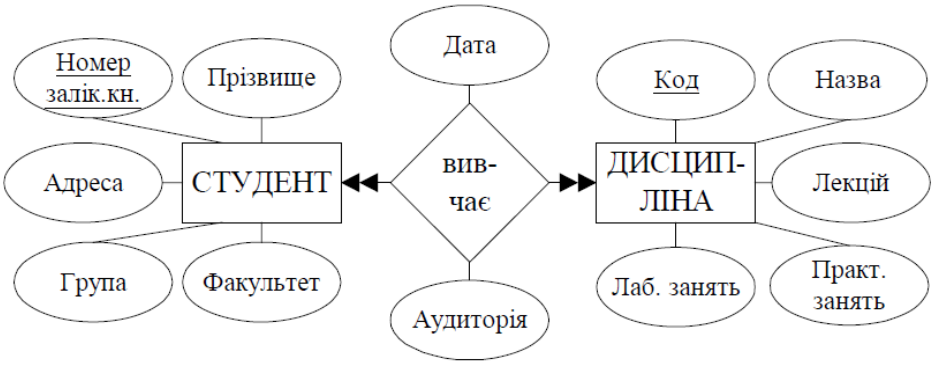


Рис. 7. Зв'язок "багато до багатьох" з атрибутами зв'язку Дата і Аудиторія

У результаті перетворення зв'язку з атрибутами отримано реляційну схему (рис. 8).

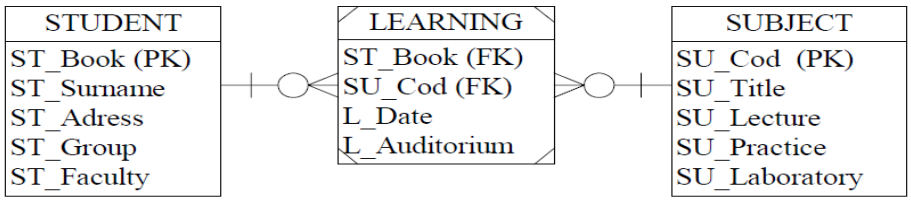


Рис. 8. Представлення атрибутів зв'язку *L_Date* і *L_Auditorium* у новому відношенні *Learning*

Спрощення концептуальної моделі передбачає також вилучення збиткових зв'язків. Збиткові зв'язки характеризуються тим, що одна і та ж інформація може бути отримана не тільки через них, але і через інші зв'язки.

Після спрощення в концептуальній моделі можуть бути присутні тільки такі елементи:

− об'єкти і атрибути;

− зв'язки типу 1:1 і 1:M;

− зв'язки типу суперклас-підклас.

3. Методика перетворення ER-діаграм в реляційні структури

Для ER-моделі існує алгоритм однозначного перетворення її в реляційну модель даних.

Розглянемо правила перетворення ER-моделі в реляційну модель.

Сутності і атрибути

Для кожної сутності створюється відношення, кожен атрибут сутності стає атрибутом відповідного відношення.

Для *сильних сутностей* первинний ключ сутності стає PRIMARY KEY (PK) відповідного відношення.

Приклад. Розглянемо сутність *Студент* (рис. 9).

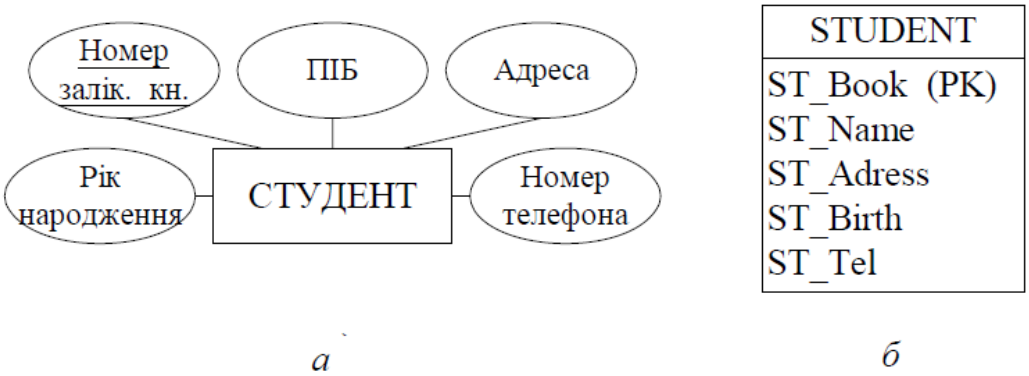


Рис. 9 . Перетворення сутності Студент (а) у відношення Student (б)

Для *слабких сутностей* первинний ключ частково або повністю залежить від ключа сутності володаря (декількох володарів), тобто PK визначається тільки тоді, коли визначені всі PK сутностей володарів.

Приклад. Розглянемо перетворення сильної сутності *Студент* і слабкої сутності *Нагорода* (рис. 10).

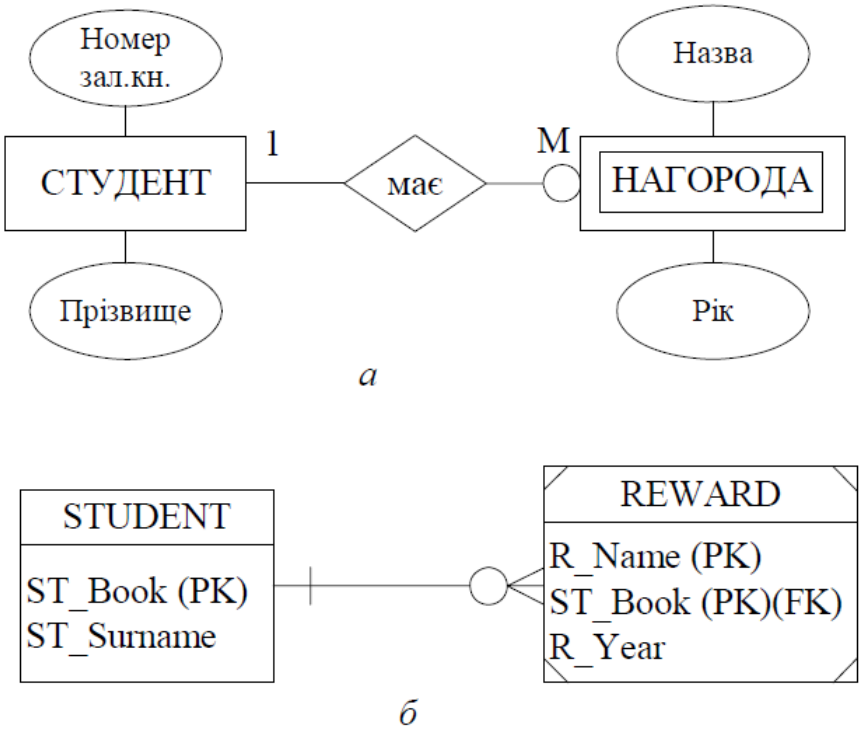


Рис. 10. Перетворення сильної сутності Студент і слабкої сутності Нагорода з ідентифікуючим зв'язком між ними (а) у зв'язані відношення Student і Reward (б)

Зв'язки

Після перетворення концептуальної моделі залишаються такі типи зв'язків:

− "один до одного";

− "один до багатьох";

− рекурсивні зв'язки;

− суперклас – підклас.

Для кожного типу зв'язку залежно від умов зв'язування існують свої різновиди. Зв'язки між відношеннями в реляційній моделі реалізуються шляхом використання первинних і зовнішніх ключів.

Зв'язки "один до одного"

В концептуальних моделях даних визначають такі обмеження ступеня участі сутностей:

− обов'язкова участь для обох сутностей;

− обов'язкова участь для однієї сутності;

− необов'язкова участь для обох сутностей.

Залежно від обмежень перетворення на реляційну модель будуть різні.

Приклад. Розглянемо можливі варіанти перетворення зв'язку між сутностями *Викладач* і *Дисципліна* (рис. 11).

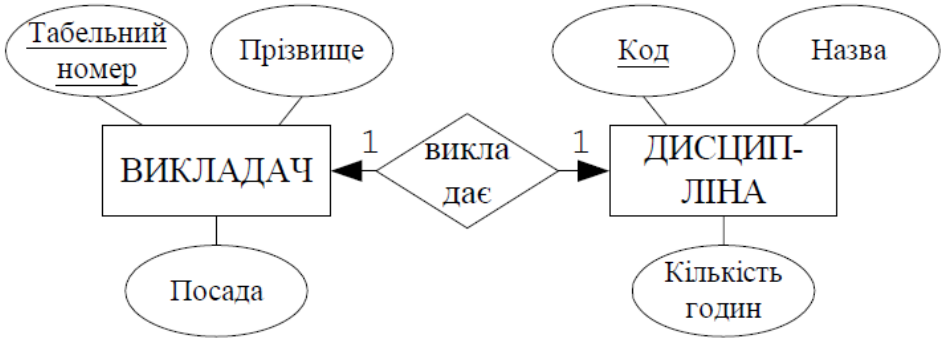


Рис. 11. Зв'язок 1:1 між сутностями *Викладач* і *Дисципліна*

1. *Обов'язкова участь для обох сутностей*

Припустимо, що кожен викладач обов'язково викладає одну дисципліну і кожну дисципліну обов'язково викладає один викладач. В цьому випадку реляційна структура буде складатися з одного відношення і мати один з таких варіантів (рис. 12).

TEACHING
T_Number (PK)
T_Name
T_Post
SU_Cod
SU_Name
SU_Hours

а

TEACHING
SU_Cod (PK)
T_Number
T_Name
T_Post
SU_Name
SU_Hours

б

Рис. 12. Варіанти відношень для перетворення зв'язку 1:1 з обов'язковою участю для обох сутностей

2. *Обов'язкова участь для однієї сутності*

Припустимо, що кожен викладач обов'язково викладає одну дисципліну, а за кожною дисципліною необов'язково закріплений викладач. В цьому випадку сутність, яка є необов'язковою (*Дисципліна*) виступає в якості батьківської сутності, а обов'язкова сутність визначається як дочірня (*Викладач*). Реляційна структура показана на рис. 13.

Зовнішній атрибут SU_Cod також може бути ключем для *Викладача*.

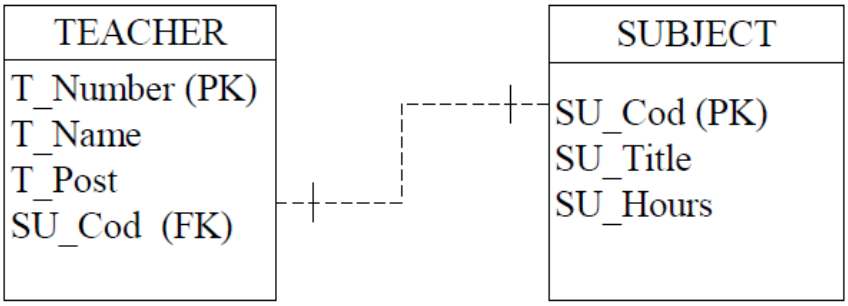
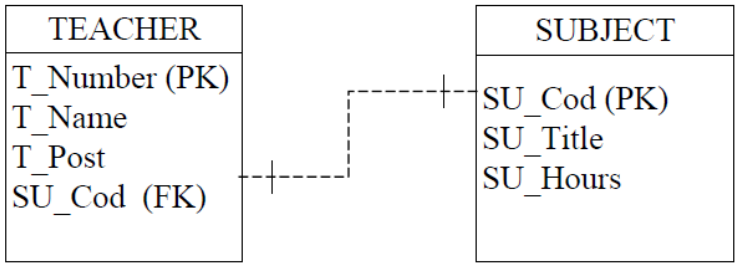


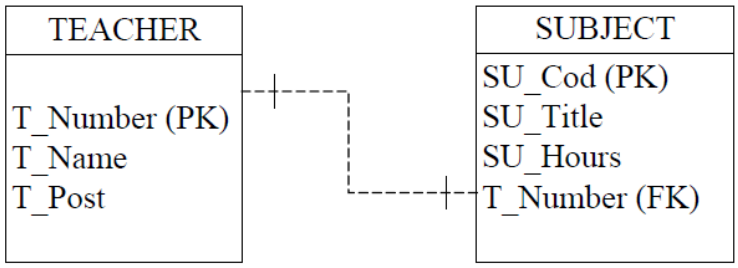
Рис. 13. Перетворення зв'язку 1:1 з обов'язковою участю сутності *Викладач* і необов'язковою участю сутності *Дисципліна*

3. *Необов'язкова участь для обох сутностей*

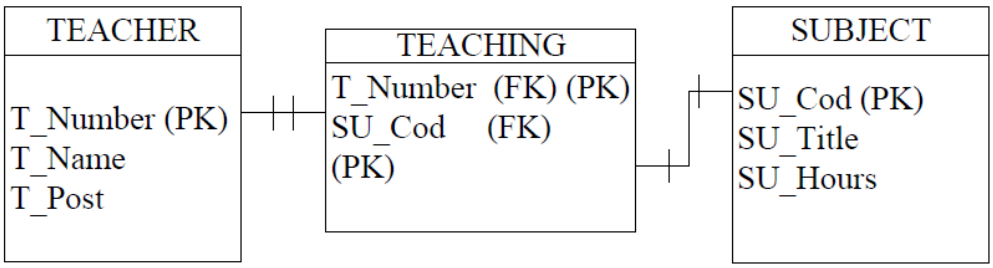
Припустимо, що необов'язково кожен викладач викладає дисципліни і необов'язково за кожною дисципліною закріплений викладач. В цьому випадку можливі три реляційні структури (рис. 14).



а



б



в

Рис. 14. Варіанти реляційних схем відношень для перетворення зв'язку 1:1 з необов'язковою участю для обох сутностей

Зв'язки "один до багатьох"

У кожне відношення, яке відповідає підлеглій (дочірній) сутності, додається набір атрибутів основної (батьківської) сутності, який складає первинний ключ основної сутності. У відношенні, що відповідає підлеглій сутності, цей набір атрибутів стає зовнішнім ключем (FOREIGN KEY, FK).

Для моделювання необов'язкового типу зв'язку у атрибутів, що відповідають зовнішньому ключу, встановлюється властивість допустимості невизначених значень (NULL). У разі обов'язкового типу зв'язку атрибути набувають властивості відсутності невизначених значень (NOT NULL).

Приклад. Розглянемо можливі варіанти перетворення зв'язку між сутностями *Викладач* і *Дисципліна* (рис. 15).

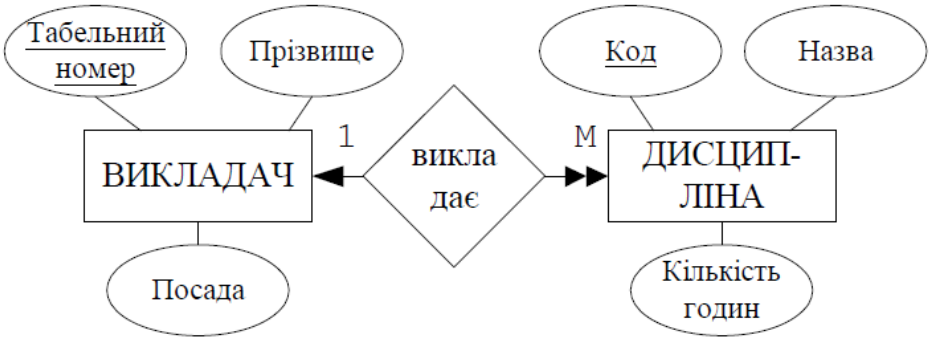


Рис. 15. Зв'язок 1:M між сутностями *Викладач* і *Дисципліна*

1. Необов'язкова участь сутності *Викладач* і обов'язкова участь сутності *Дисципліна* (рис. 16).

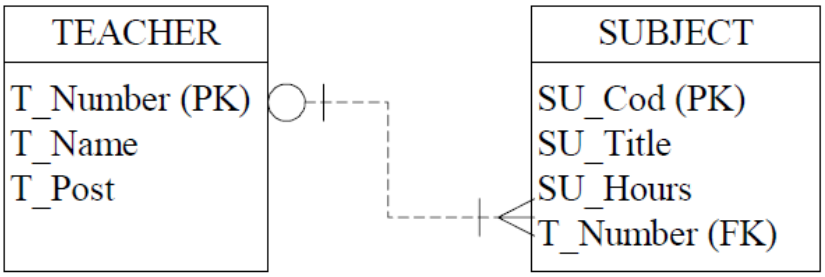


Рис. 16. Перетворення зв'язку 1:M з необов'язковою участю сутності *Викладач* і обов'язковою участю сутності *Дисципліна*

2. Необов'язкова участь сутності *Викладач* і необов'язкова участь сутності *Дисципліна* (рис. 17).

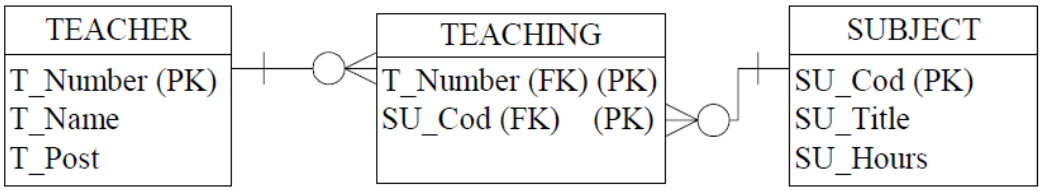


Рис. 17. Перетворення зв'язку 1:M з необов'язковою участю сутності *Викладач* і необов'язковою участю сутності *Дисципліна*

Зв'язки "багато до багатьох"

Для кожного зв'язку M:N необхідно створювати додаткове відношення, яке представляє цей зв'язок і включати в нього всі атрибути, які входять в склад цього зв'язку. Копії атрибутів первинного ключа сутностей, які беруть участь у зв'язку, передаються у нове відношення для використання в якості зовнішніх ключів. Ці зовнішні ключі утворюють також первинний ключ нового відношення.

Приклад. Розглянемо можливі варіанти перетворення зв'язку між сутностями Викладач і Дисципліна (рис. 18).

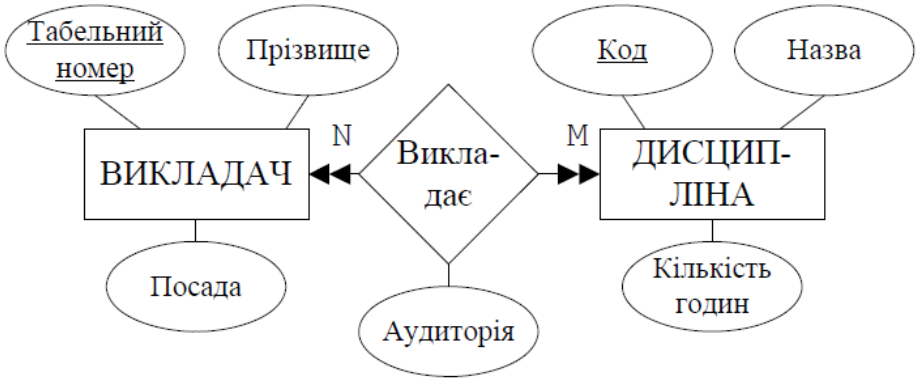


Рис. 18. Зв'язок N:M між сутностями *Викладач* і *Дисципліна*

В цьому випадку існує єдина схема перетворення (рис. 19).

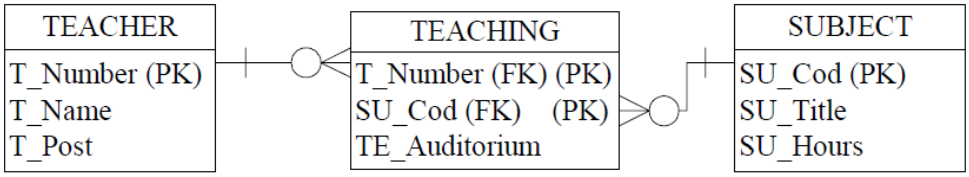


Рис. 19. Перетворення зв'язку N:M у реляційну схему

Для рекурсивних зв'язків 1:1 виконуються правила

визначені раніше для зв'язку між двома сутностями 1:1. Для рекурсивного зв'язку 1:1 з обов'язковою участю двох сторін, реляційна схема представляється

у вигляді одного відношення з двома копіями первинного ключа (див. рис. 12). Одна копія відповідає зовнішньому ключу. Для рекурсивного зв'язку 1:1 з обов'язковою участю тільки однієї сторони створюється або одне відношення, або нове відношення, яке відображає цей зв'язок (див. рис. 13). Для рекурсивного зв'язку 1:1 з необов'язковою участю обох сторін створюється нове відношення (див. рис. 14).

Для складних типів зв'язків створюється відношення, яке відображає цей зв'язок і включає всі атрибути, які входять в склад цього зв'язку. Копії атрибутів первинного ключа сутностей, які беруть участь у зв'язку, передаються у нове відношення для використання в якості зовнішніх ключів. Ці зовнішні ключі утворюють також первинний ключ нового відношення (див. рис. 3).

Для багатозначного атрибуту створюється нове відношення, яке відповідає багатозначному атрибуту, і в це нове відношення передається первинний ключ сутності для використання в якості зовнішнього ключа (див. рис. 4).

Зв'язки "суперклас – підклас"

Для виконання перетворення зв'язку типу суперклас – підклас у реляційну модель необхідно враховувати також обмеження ступеня участі у зв'язку (Mandatory або Optional) і обмеження неперетинання (And або Or). Можливі чотири сполучення, перетворення яких дає чотири реляційні схеми. На схему також впливає те, чи беруть участь підкласи в різних зв'язках, кількість сутностей в зв'язку і т.ін. Діапазон можливих варіантів рішення є достатньо великим і конкретна схема вибирається в кожному конкретному випадку з урахуванням багатьох факторів.

Приклад. Розглянемо суперклас *Викладач*, який має атрибути *Табельний номер, Прізвище, Посада.* Підкласами суперкласу виступають об'єкти *Професор, Доцент, Асистент* (рис. 20). Кожен екземпляр підкласу може бути екземпляром суперкласу, тобто суперклас може мати свої екземпляри (Optional). Кожен викладач обов'язково належить тільки одному підкласу (Or). Ця діаграма перетворюється в реляційну схему відношень показану на рис. 21. Зв'язок між відношеннями виконується за допомогою ключа суперкласу (*Табельний номер*).

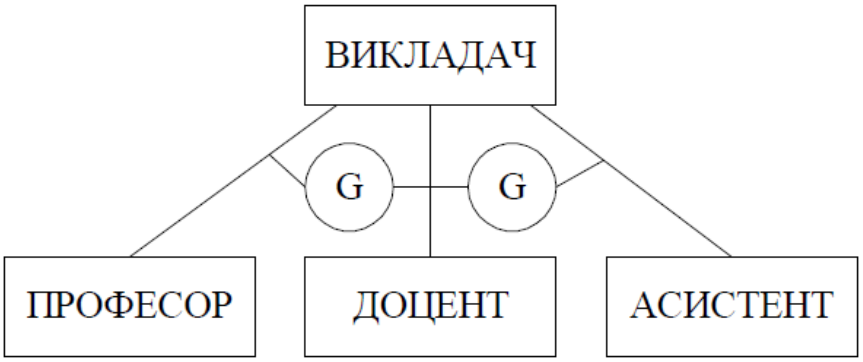


Рис. 20. Зв'язок суперклас – підклас з обмеженнями *Optional* і *Or*

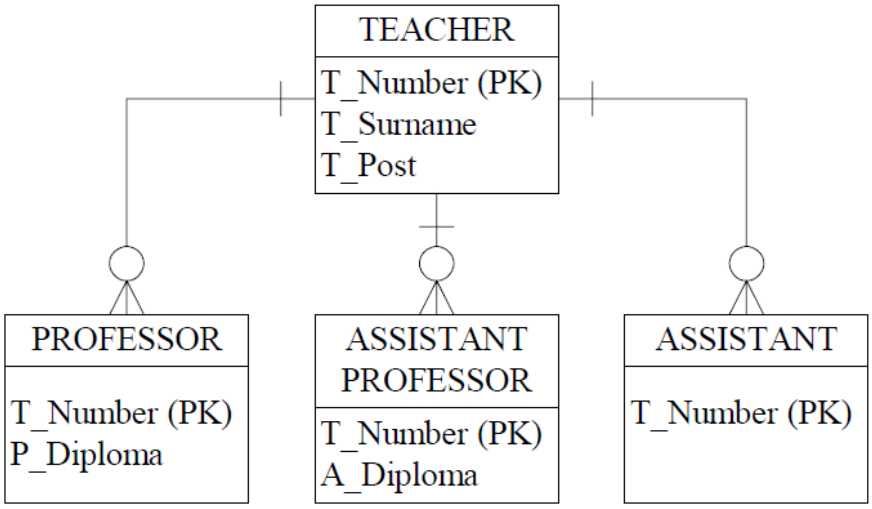


Рис. 21. Реляційна схема, яка відповідає попередньому зв'язку суперклас – підклас

Приклад. Розглянемо суперклас *Студент*, який має атрибути *Номер залікової книжки, Прізвище, Група*. Підкласами суперкласу виступають об'єкти *Очна, Заочна, Вечірня і Дистанційна форми навчання* (рис. 22). Кожен екземпляр підкласу є одночасно екземпляром суперкласу (Mandatory). Кожен студент може належати до декількох підкласів, тобто одночасно може займатися на різних формах навчання (And). Ця діаграма перетворюється в наступну реляційну схему відношень (рис. 23).

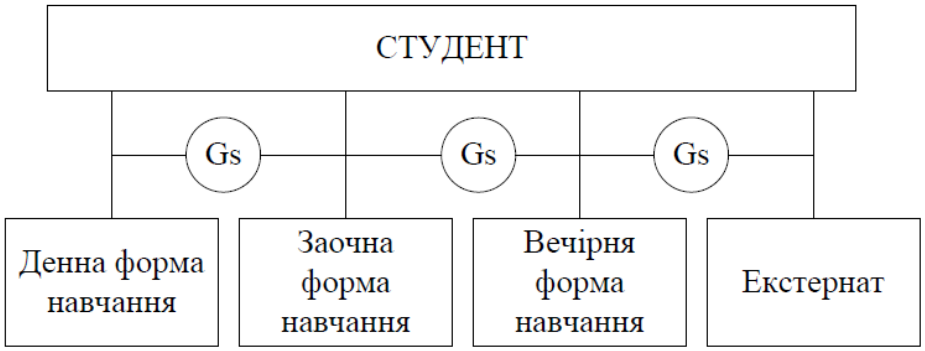


Рис. 22. Зв'язок суперклас–підклас з обмеженнями *Mandatory* і *And*

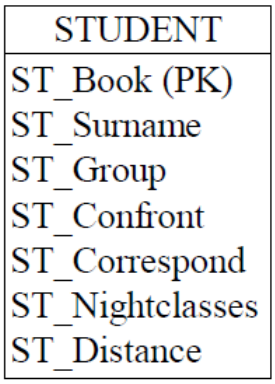
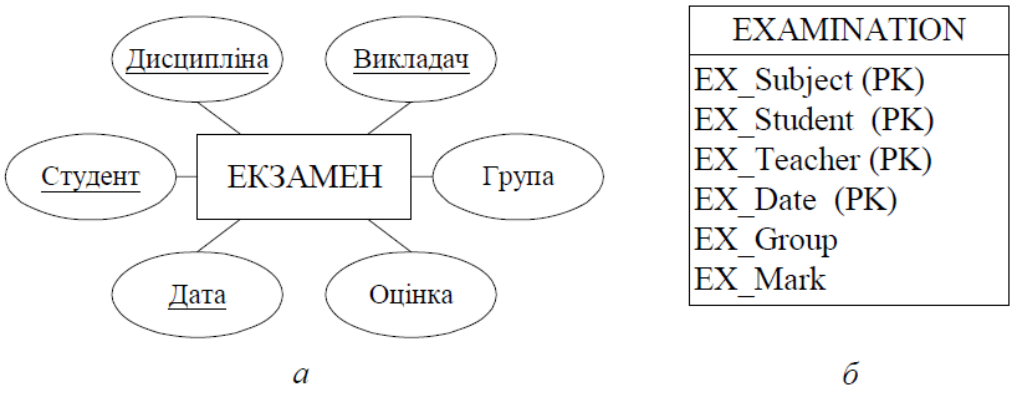


Рис. 23. Реляційна схема, яка відповідає попередньому зв'язку суперклас–підклас

4. Перевірка відношень за допомогою правил нормалізації

Створений на попередніх етапах набір відношень логічної моделі БД повинен бути перевірений на коректність об'єднання атрибутів у кожному відношенні. Перевірка виконується шляхом застосування до кожного відношення процедури послідовної нормалізації. Нормалізація гарантує, що отримана модель не буде мати протирічь і буде мати мінімальну збитковість. Атрибути в результаті нормалізації будуть згруповані відповідно до існуючих між ними логічних зв'язків. Для забезпечення коретності логічної моделі, у разі виявленнявідношень, які не відповідають вимогам нормалізації, необхідно повернутися на попередні етапи проектування і перебудувати помилково створені елементи моделі.

Приклад. В результаті проектування отримано відношення показана на рис. 24.



а *б*

Рис. 24. Перетворення сутності *Екзамен* (а) у відношення *Examination* (б)

При дослідженні даного відношення були виявлені такі функціональні залежності:

Дисципліна, Викладач, Студент, Дата → Оцінка Студент → Група

У наведеній схемі існують аномалії і необхідно продовжити нормалізацію. В результаті декомпозиції вихідного відношення буде отримана схема показана на рис. 25.

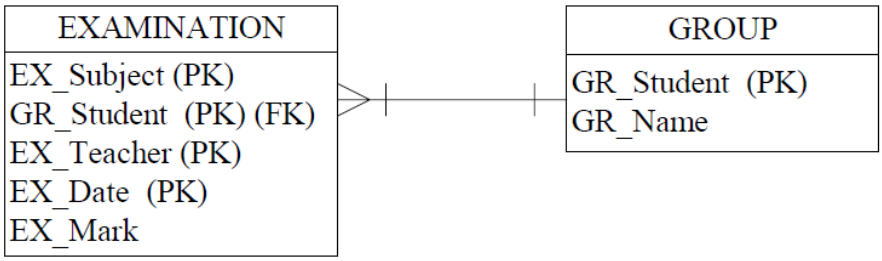


Рис. 25. Реляційна схема, яка відповідає сутності *Екзамен*

5. Перевірка відповідності відношень вимогам транзакцій користувачів

Перевірка полягає в нанесенні безпосередньо на ER-діаграму всіх шляхів, які потрібні для виконання кожної з транзакцій. Якщо таким чином вдається виконати всі транзакції, то перевірка на цьому завершується. У протилежному випадку необхідно повернутися до попередніх етапів і перевірити, а у разі потреби і змінити ті фрагменти моделі, які не відповідають необхідній роботі транзакцій.

Якщо в результаті перевірки будуть виявлені області, які не беруть безпосередньої участі у роботі транзакцій, то можливо їх вилучення з моделі.

6.6. Перевірка підтримки цілісності

Обмеження цілісності запобігають появі в БД суперечливих даних. Вирішення цієї проблеми на стадії проектування полягає у такому:

− наявність обов'язкових і необов'язкових значень даних для атрибутів (NULL, NOT NULL);

− наявність обмежень для доменів атрибутів (визначення області значень або діапазону значень);

− цілісність сутностей (обов'язкова наявність Primary Key в кожному відношенні);

− посилкова цілісність (зв'язування таблиць за допомогою Foreign Key);

− обмеження предметної області (бізнес правила), які реалізуються як засобами БД, так і на рівні застосувань.

У табл. 1 наведені правила зовнішнього ключа для відношення "один до багатьох" для сильної сутності.

Таблиця 1

Підтримка посилкової цілісності для сильної сутності

Тип зв'язку	Вимоги до зовнішнього ключа
Обов'язкова наявність значень відповідних екземплярів у батьківській і залежній таблицях	NOT NULL ON DELETE RESTRICT ON UPDATE CASCADE
Необов'язкова наявність значень відповідних екземплярів у батьківській і залежній таблицях	NULL ALLOWED ON DELETE SET NULL ON UPDATE CASCADE
Обов'язкова наявність значень відповідних екземплярів у залежній таблиці і необов'язкова наявність значень в батьківській таблиці	NULL ALLOWED ON DELETE SET NULL ON DELETE RESTRICT ON UPDATE CASCADE
Обов'язкова наявність значень відповідних екземплярів у батьківській таблиці і необов'язкова наявність значень в залежній таблиці	NOT NULL ON DELETE RESTRICT ON UPDATE CASCADE

Для слабкої сутності використовуються ті ж самі правила за винятком обмежень на зовнішній ключ: NOT NULL, ON DELETE CASCADE, ON UPDATE CASCADE.

6. Приклад створення логічної моделі бази даних

У попередній лекції (*Концептуальне проектування баз даних* п.4) розроблено концептуальний проект бази даних для предметної області ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД (ВНЗ). ER-діаграма відображає всі бізнес правила, які в свою чергу визначають сутності, атрибути, зв'язки і т.д.

Наступним етапом проектування бази даних є створення логічної моделі бази даних на основі створеної ER-моделі (рис. 21). Створення логічної моделі бази даних виконується шляхом застосування правил перетворення ER-діаграми в логічну модель (рис. 26).

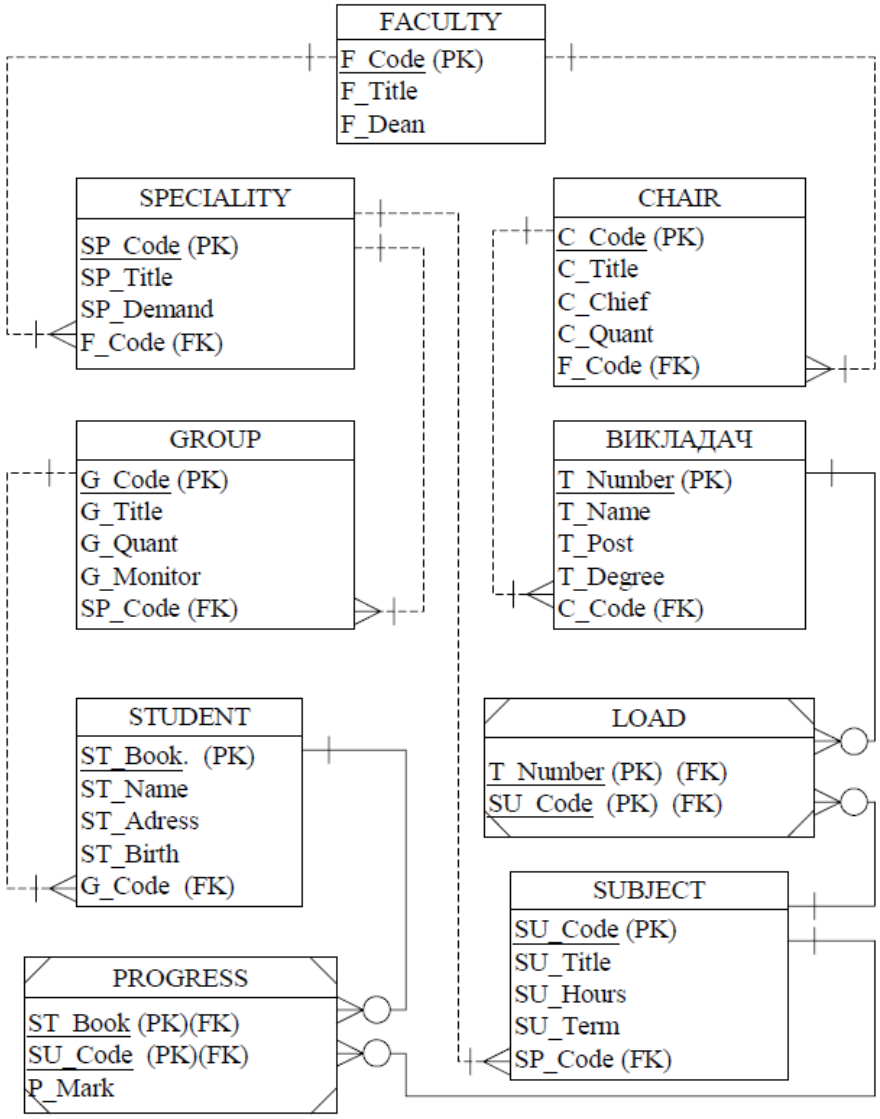


Рис. 26. Логічна модель бази даних для предметної області ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

Після створення логічної моделі даних реляційна схема аналізується на коректність об'єднання атрибутів в одному відношенні. Перевірка коректності виконується шляхом застосування послідовної нормалізації до кожного з відношень. Метою цієї перевірки є отримання гарантій того, що схема бази даних щонайменше знаходиться в 3-й нормальній формі або в нормальній формі Бойса-Кодда. Якщо ця умова не виконується, то необхідно повернутися на попередні етапи проектування і перебудувати помилково створені

фрагменти моделі. Перевірка логічної моделі бази даних ВНЗ показує, що реляційна схема знаходиться в 4-й нормальній формі й корегування моделі не потрібно.

Після перевірки логічної моделі за допомогою правил нормалізації система аналізується на предмет виконання транзакцій користувачів, які задаються на початкових етапах проектування. У разі неможливості виконання певних транзакцій необхідне корегування моделі бази даних (див. рис. 1).

Подальша перевірка моделі вимагає перевірки підтримки цілісності даних. На основі матеріалів прикладу можна тільки визначити, що підтримується посилкова цілісність. Всі інші перевірки, включаючи і перевірку транзакцій користувачів, вимагають більш детально опрацьованого проекту бази даних.

Правила перетворення ER-діаграми в логічну модель наведені в е-матеріалі «*Правила перетворення ER-діаграми в логічну модель*» дистанційного курсу «*Проектування баз даних та знань для інформаційно-комунікаційних систем*» у віртуальному університеті ЛДУ БЖД.

Контрольні запитання

1. Що називається логічним проектуванням?
2. Яка інформація є вихідною для логічного проектування?
3. Перелічити етапи логічного проектування.
4. Які зв'язки ER-діаграм не підтримуються в реляційній схемі?
5. Як відображаються сильні і слабкі сутності та їх атрибути в реляційну схему?
6. Як відображається необов'язковість зв'язку між сутностями в реляційну схему?
7. Як відображається зв'язок "багато до багатьох" в реляційну схему?
8. Як відображаються складні зв'язки і зв'язки з атрибутами в реляційну схему?
9. Як відображаються рекурсивні зв'язки в реляційну схему?
10. Як відображається зв'язок "один до одного" залежно від рівня участі сутностей в реляційну схему?

11. Як відображається зв'язок "один до багатьох" залежно від рівня участі сутностей в реляційну схему?

12. Як відображається зв'язок суперклас–підклас залежно від ступеня участі сутностей і обмеження неперетинання в реляційну схему?

13. Навіщо потрібно виконувати перевірку реляційної схеми на відповідність правилам нормалізації?

14. Як виконується перевірка реляційної схеми на відповідність вимогам транзакцій користувачів?

15. Як перевіряється цілісність реляційної бази даних?

16. Навести приклади створення логічної моделі бази даних.

ЛЕКЦІЯ. НОРМАЛІЗАЦІЯ

План:

1. Постановка задачі.
2. Нормальні форми.
3. Денормалізація.

Література

1. Гайна Г.А. Основи проектування баз даних: Навчальний посібник. К.: КНУБА, 2005. – 204 с.
2. Гайна Г.А. Організація баз даних і знань. Мови баз даних: Конспект лекцій.−К.:КНУБА, 2002. – 64 с.
3. *Гайна Г.А., Попович Н.Л.* Організація баз даних і знань. Організація реляційних баз даних: Конспект лекцій.−К.:КНУБА, 2000. – 76 с.

1. Постановка задачі.

Нормалізація – це процедура визначення того, які атрибути зв'язані у відношенні. Одна з головних задач при розробці реляційної БД – об'єднання в одному відношенні тих атрибутів, які зв'язані між собою (між якими є функціональні залежності). Нормалізація являє собою поетапний процес заміни сукупності відношень іншою сукупністю (схемою), в якій відношення мають просту і регулярну структуру. Результатом нормалізації є логічна модель БД.

Надлишковість даних в БД є небажаним явищем, оскільки призводить до збільшення об'єму пам'яті, уповільнює роботу БД. Надлишковість даних є результатом в першу чергу дублювання даних. Розрізняють *незбиткове* та *збиткове* дублювання даних. Повністю усувати надлишковість не потрібно, оскільки при цьому неможливо буде підтримувати БД як єдине ціле. Слід тільки мінімізувати надлишковість, залишивши необхідне дублювання даних.

Дублювання даних створює проблеми при виконанні операцій з БД. Ці проблеми виникають при спробі зробити операції: редагування, додавання або вилучення даних.

Аномаліями називається така ситуація в БД, яка призводить до протирічь у БД, або суттєво ускладнює обробку даних. Розрізняють аномалії модифікації, додавання і вилучення.

Приклад. Розглянемо відношення *Студент* (табл. 1).

Таблиця 1

Студент

Номер залікової книжки	Прізвище	Група	Факультет	Декан
1010	Бойко	ІТП-31	АІТ	Барков
2020	Лемешко	ІУСТ-22	АІТ	Барков
1030	Шевченко	ІТП-31	АІТ	Барков
1121	Петренко	БМО-32	АІТ	Барков
2231	Іванченко	ТБ-21	БТ	Тимчук

Аномалія модифікації виникає при спробі змінити прізвище декана. В цій ситуації необхідно переглянути всі кортежі. При великих розмірах БД це

потребує значного часу, при цьому можливі помилки (у разі невірного введення прізвища), які порушать цілісність БД.

Аномалія додавання виникає при додаванні інформації про нового студента, при цьому необхідно вводити інформацію, яка вже є в БД: назва факультету, прізвище декана. Крім того неможливо створити нову групу поки не існує студентів, які в ній займаються.

Аномалія вилучення виникає при спробі вилучити дані про студента, який в групі поки ще один, наприклад Лемешко. В цьому випадку зникне інформація про групу ІУСТ-22. Виконання декомпозиції наведеного відношення дозволяє позбутися вищеозначених аномалій (табл. 2...4).

Таблиця 2

Студент

Номер залікової книжки	Прізвище	Група
1010	Бойко	ІТП-31
2020	Лемешко	ІУСТ-22
1030	Шевченко	ІТП-31
1121	Петренко	БМО-32
2231	Іванченко	ТБ-21

Таблиця 3

Група

Група	Факультет
ІТП-31	АІТ
ІУСТ-22	АІТ
БМО-32	АІТ
ТБ-21	БТ

Таблиця 4

Факультет

Факультет	Декан
АІТ	Барков
БТ	Тимчук

Процес проектування БД з використанням декомпозиції являє собою процес послідовної нормалізації схем відношень, при цьому кожна наступна ітерація відповідає нормальній формі більш високого рівня і має кращі властивості у порівнянні з попередньою. Кожній нормальній формі (НФ) відповідає деякий набір обмежень. Визначають такі нормальні форми: 1НФ, 2НФ, 3НФ, НФБК (нормальна форма Бойса-Кодда), 4НФ, 5НФ.

При виконанні декомпозиції зберігається множина вихідних функціональних залежностей між атрибутами і виконується зворотність. *Зворотність* означає можливість відновлення вихідної схеми. *Функціональні залежності* відображають зв'язки між атрибутами, які властиві реальному об'єкту.

Атрибут *B* *функціонально* залежить від *A*, якщо кожному значенню *A* відповідає в точності одне значення *B*.

Математичний запис функціональної залежності (ФЗ): $A \rightarrow B$

Приклад. Функціональні залежності:

Студент → Група; *Група → Факультет*;

Викладач, Студент, Дисципліна → Оцінка.

Якщо існує ФЗ $A \rightarrow B$, то це означає, що у всіх кортежах з однаковим значенням атрибуту *A* атрибут *B* буде мати також одне й те ж значення. *A* і *B* можуть складатися з декількох атрибутів.

2. Нормальні форми.

Перша нормальна форма. Відношення знаходиться в 1НФ тоді і тільки тоді, коли всі його атрибути є атомарними.

Значення атрибуту вважається *атомарним*, якщо воно є неподільним у всіх застосуваннях.

Приклад. Представлення даних у таблицях може вважатися як атомарним, так і неатомарним залежно від використання. Засіб представлення визначається необхідним ступенем деталізації і повинен підтримуватися у всіх застосуваннях (табл. 5).

Таблиця 5

Дата народження

Прізвище	Дата народження
Бойко	15 лютого 1991

Прізвище	Дата і місяць	Рік
Бойко	15 лютого	1991

Прізвище	День	Місяць	Рік
Бойко	15	лютий	1991

Друга нормальна форма. Відношення знаходиться в 2НФ, якщо воно знаходиться в 1НФ і кожен його непервинний атрибут функціонально повно залежить від первинного ключа.

Неповною функціональною залежністю називається залежність неключового атрибуту від частини ключа, що складається з декількох атрибутів. Повна функціональна залежність передбачає залежність неключового атрибуту від всіх атрибутів одночасно, що входять до складу ключа.

Приклад. Розглянемо відношення *Студент* (табл. 6).

Таблиця 6

Студент

Номер залікової книжки	Прізвище	Група	Дисципліна	Оцінка
1010	Бойко	ІТП-31	Бази даних	5

Функціональні залежності:

№ залік. кн., Дисципліна → Прізвище, Група, Оцінка

№ залік. кн. → Прізвище, Група

Для приведення даного відношення до 2НФ необхідно розбити його на проекції, при цьому повинна бути виконана умова відновлення вихідного відношення без втрат. Проекції мають такий вигляд (табл. 7...8).

Таблиця 7

Студент

Номер залікової книжки	Дисципліна	Оцінка
1010	Бази даних	5

Таблиця 8

Група

Номер залікової книжки	Прізвище	Група
1010	Бойко	ІТП-31

Відсутність втрат при декомпозиції відношення R(x,y,z) на відношення R1(x,y) і R2(x,z) виконується, якщо від спільного атрибуту двох отриманих відношень (x) залежить хоча б один атрибут з двох, що залишилися (y або z), тобто якщо виконується (x → y) або (x → z).

Третя нормальна форма. Відношення знаходиться в 3НФ, якщо воно знаходиться в 2НФ і жоден з непервинних атрибутів у відношенні не є транзитивно залежним від первинного ключа.

Атрибут C *транзитивно* залежить від атрибуту A, якщо для атрибутів A, B, C виконуються такі умови A → B і B → C, але зворотня залежність відсутня.

Приклад. Розглянемо відношення *Студент* (табл. 9).

Таблиця 9

Студент

Номер залікової книжки	Прізвище	Група	Факультет
1010	Бойко	ІТП-31	АІТ

Функціональні залежності:

№ залік. кн. → Прізвище, Група, Факультет

Група → Факультет

Між атрибутами існує транзитивна залежність. Для того щоби запобігти цьому необхідно виконати декомпозицію відношення (табл. 10, 11):

Таблиця 10

Студент

Номер залікової книжки	Прізвище	Група
1010	Бойко	ІТП-31

Таблиця 11

Група

Група	Факультет
ІТП-31	АІТ

Нормальна форма Бойса-Кодда. Відношення знаходиться в НФБК, якщо воно знаходиться в 3НФ і у ньому відсутні залежності атрибутів первинного ключа від неключових атрибутів.

Приклад. Розглянемо відношення *Спеціальність* (табл. 12).

Таблиця 12

Спеціальність

Спеціальність	Дисципліна	Викладач
ІТП	Бази даних	Барко
ІУСТ	Бази даних	Шевченко

Припустимо, що на кожній спеціальності певну дисципліну може викладати тільки один викладач і кожен викладач викладає тільки одну дисципліну. У цьому випадку мають місце такі залежності:

Спеціальність, Дисципліна → Викладач

Викладач → Дисципліна

Відношення знаходиться в 3НФ, але неключовий атрибут *Викладач* визначає атрибут *Дисципліна*, що входить у ключ.

Для того щоби позбутися аномалій необхідно виконати декомпозицію відношення (табл. 13, 14).

Таблиця 13

Спеціальність

Спеціальність	Дисципліна
ІТП	Бази даних
ІУСТ	Бази даних

Таблиця 13

Дисципліна

Викладач	Дисципліна
Барко	Бази даних
Шевченко	Бази даних

Четверта нормальна форма. Відношення знаходиться в 4НФ тоді і тільки тоді, коли у випадку існування багатозначної залежності A→→B всі інші атрибути відношення функціонально залежать від A.

У відношенні R(A,B,C) існує багатозначна залежність A→→B в тому і тільки в тому випадку, коли множина значень B, що відповідає парі значень A і C залежить тільки від A і не залежить від C.

Відношення R(A,B,C) можна розбити без втрат інформації на відношення R1(A,B) і R2(A,C) в тому і тільки в тому випадку, якщо існують багатозначні залежності A→→B і A→→C.

Приклад. Розглянемо відношення *Кафедра* (табл.15).

Таблиця 15

Кафедра

Кафедра	Викладач	Група
ІТ	Барко	ІТП-31
ІТ	Барко	ІТП-32
ІТ	Шевченко	ІТП-31
ІТ	Шевченко	ІТП-32

У даному відношенні існують дві багатозначні залежності:

Кафедра →→ Викладач

Кафедра →→ Група

Це означає, що кожній кафедрі відповідає перелік викладачів, які на ній працюють і кожній кафедрі відповідає перелік груп, яким ця кафедра викладає дисципліни.

Для того, щоби звести відношення до 4НФ, необхідно виконати його декомпозицію (табл. 16, 17).

Таблиця 16

Кафедра

Кафедра	Викладач
ІТ	Барко
ІТ	Шевченко

Таблиця 16

Група

Кафедра	Група
ІТ	ІТП-31
ІТ	ІТП-32

П'ята нормальна форма. Відношення знаходиться в 5НФ тоді і тільки тоді, коли будь-яка залежність з'єднання у відношенні виходить з існування деякого можливого ключа у відношенні.

Відношення R(X,Y,…,Z) задовольняє залежності з'єднання (X,Y,…,Z) тоді і тільки тоді, коли R відновлюється без втрат інформації шляхом з'єднання своїх проекцій на X, Y,…, Z. Залежність з'єднання є узагальненням функціональної і багатозначної залежностей.

Приклад. Розглянемо відношення *Заняття*:

Заняття (*Студент, Викладач, Дисципліна*)

Кожен студент слухає лекції багатьох викладачів, кожен викладач викладає для багатьох студентів, кожен студент вивчає багато дисциплін, кожен викладач викладає багато дисциплін. У відношенні відсутні багатозначні і функціональні залежності й воно знаходиться в 4НФ. У відношенні можливі аномалії, які пов'язані з повтором значень атрибутів в декількох кортежах. Наприклад, якщо студент навчається у багатьох викладачів, то при його відрахуванні з університету необхідно знайти і вилучити декілька записів з відношення.

Утворимо такі складені атрибути відношення:

СВ (*Студент, Викладач*)

СД (*Студент, Дисципліна*)

ВД (*Викладач, Дисципліна*).

Якщо відношення R спроектувати на складені атрибути СВ, СД, ВД, то з'єднання цих проекцій дасть вихідне відношення. Це означає, що у відношенні *Заняття* існувала залежність з'єднання. Результатом декомпозиції відношення *Заняття* буде отримання таких відношень: R1(*Студент, Викладач*), R2(*Студент, Дисципліна*), R3(*Викладач, Дисципліна*).

Для зведення вихідного відношення до 5НФ виконують його декомпозицію на відношення, кількість яких перевищує два.

Результати зведення до нормальних форм наведені в табл. 18.

Таблиця 18

Правила формування нормальних форм

Нормальні форми	Приклад
1НФ	$R(ABCD)$ – відношення A,B,C,D – атомарні атрибути
2НФ	$R(\underline{AB}CD)$ – відношення, $\underline{AB}$ – ключ, $\underline{AB} \to CD$, неможливі залежності: $\underline{A} \to CD$, $\underline{A} \to C$, $\underline{A} \to D$, $\underline{B} \to CD$, $\underline{B} \to C$, $\underline{B} \to D$
3НФ	$R(\underline{AB}CD)$ – відношення, $\underline{AB}$ – ключ, $\underline{AB} \to CD$, неможливі залежності: $C \to D$, $D \to C$

Нормальні форми	Приклад
НФБК	$R(\underline{AB}CD)$ – відношення, $\underline{AB}$ – ключ, $\underline{AB} \to CD$, неможливі залежності: $C \to \underline{A}$, $C \to \underline{B}$, $D \to \underline{A}$, $D \to \underline{B}$, $C \to \underline{AB}$, $D \to \underline{AB}$
4НФ	$R(\underline{A}BC)$ – відношення, $A \to\to B$, $A \to C$, неможливі залежності: $A \to\to C$
5НФ	$R(ABC)$ – вихідне відношення; результат декомпозиції: $R_1(\underline{AB})$, $R_2(\underline{AC})$, $R_3(\underline{BC})$

3. Денормалізація.

Денормалізація – модифікація реляційної моделі, при якій ступінь нормалізації модифікованого відношення стає нижче, ніж ступінь нормалізації щонайменше одного з вихідних відношень.

Денормалізація застосовується у тих випадках, коли нормалізована БД не задовольняє вимогам, що висуваються до продуктивності системи. Денормалізація може застосовуватися у таких випадках:

− об'єднання таблиць зі зв'язками "один до одного";

− дублювання неключових атрибутів у зв'язках "один до багатьох" для зменшення кількості з'єднань;

− дублювання атрибутів зовнішнього ключа у зв'язках "один до багатьох" для зменшення кількості з'єднань;

− дублювання атрибутів "багато до багатьох" для зменшення кількості з'єднань;

− створення таблиць з даних, що містяться в інших таблицях;

− введення груп полів, що повторюються.

Застосовуючи денормалізацію слід враховувати, що цей процес має такі негативні наслідки:

− призводить до появи аномалій БД;

− знижує гнучкість системи;

− може зменшити час на відповіді до БД, але при цьому уповільнює операції оновлення даних;

− може ускладнити фізичну реалізацію системи.

Контрольні запитання

1. Дати визначення терміну функціональна залежність. Навести приклади функціональних залежностей.

2. У чому полягає збиткове і незбиткове дублювання даних?

3. Що таке аномалії додавання, оновлення, вилучення?

4. Дати визначення 2НФ. Навести приклад відношення, яке знаходиться в 1НФ, але не знаходиться у 2НФ. Звести його до 2НФ.

5. Дати визначення 3НФ. Навести приклад відношення, яке знаходиться в 2НФ, але не знаходиться у 3НФ. Звести його до 3НФ.

6. Дати визначення НФБК. Навести приклад відношення, яке знаходиться в 3НФ, але не знаходиться у НФБК. Звести його до НФБК.

7. Дати визначення терміну багатозначна залежність. Навести приклади багатозначних залежностей.

8. Дати визначення 4НФ. Навести приклад відношення, яке знаходиться в НФБК, але не знаходиться у 4НФ. Звести його до 4НФ.

9. Дати визначення 5НФ. Навести приклад 5НФ.

10. Яке місце займає нормалізація в процесі проектування бази даних?

11. Що таке денормалізація бази даних і які її переваги і недоліки?

12. Навести приклади проекту бази даних, коли доцільно виконати денормалізацію.

Лекція. СУБД MySQL

План лекції.

1. Логічна архітектура MySQL
2. Конфігурація сервера
3. Типи даних
4. Основні команди

15.1 Логічна архітектура MySQL

На найвищому рівні містяться служби, які не є чимось унікальним для MySQL. Ці служби необхідні більшості мережних клієнт – серверних інструментів і серверів: вони забезпечують підтримку з'єднань, ідентифікацію, безпеку тощо.

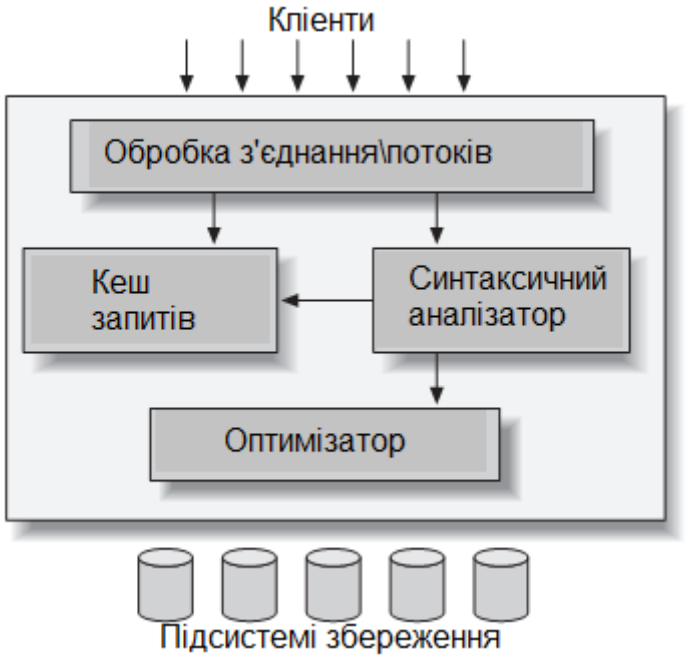


Рисунок 15.1 – Логічна архітектура сервера MySQL

Другий рівень. Тут зосереджена значна частина інтелекту MySQL: синтаксичний аналіз запитів, оптимізація, кешування і всі вбудовані функції (наприклад, функції роботи з датами і часом, математичні функції, шифрування). На цьому рівні реалізується будь-яка незалежна від підсистеми зберігання даних функціональність, наприклад, процедури подання, збережені процедури, зокрема тригери. *Тригер – це процедура, що зберігається, яка не викликається безпосередньо, а виконується із настанням певної події (вставка, видалення, оновлення рядка).*

Підтримка тригерів у MySQL почалася з версії 5.0.2.

Третій рівень містить підсистеми зберігання даних. Вони відповідають за збереження і вилучення всіх даних, що зберігаються в MySQL. Подібно до

різних файлових систем NTFS / wind GNU / Linux, кожна підсистема зберігання даних має свої сильні й слабкі сторони. Сервер взаємодіє з ними за допомогою API (інтерфейс прикладного програмування) підсистеми зберігання даних.

Цей інтерфейс приховує відмінності між підсистемами зберігання даних і робить їх майже прозорими на рівні запитів. Крім того, даний інтерфейс містить пару десятків низькорівневих функцій, які виконують операції типу «почати транзакцію» або «витягти рядок з таким первинним ключем». Підсистеми зберігання не роблять синтаксичний аналіз коду SQL і не взаємодіють один з одним, вони просто відповідають на запити, що виходять від сервера.

15.2 Конфігурація сервера

Конфігурація починається зі стартового вікна рис. 15.2.

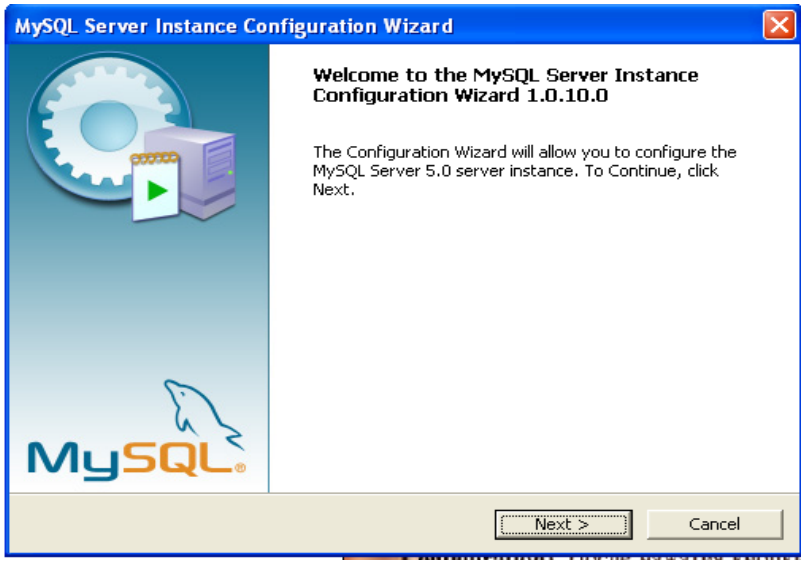


Рисунок 15.2 – Запуск конфігурування MySQL Server-5

У вікні рис. 15.3 пропонується обрати режим налаштування

Detailed Configuration (Детальний);

Standard Configuration (Стандартний).

Для більш гнучкого настроювання системи слід обрати перший пункт (Detailed Configuration).

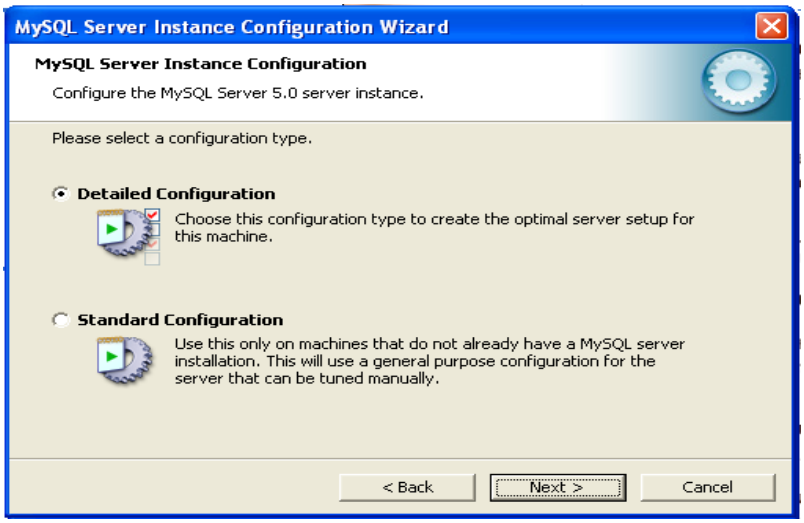


Рисунок 15.3 – Вибір режиму конфігурування

На рис 15.4 подано вікно ,де проводиться налаштування продуктивності MySQL з такими опціями:

Developer Machine (Машина розробника);

Server Machine (Сервер);

Dedicated MySQL Server Machine (Виділений сервер).

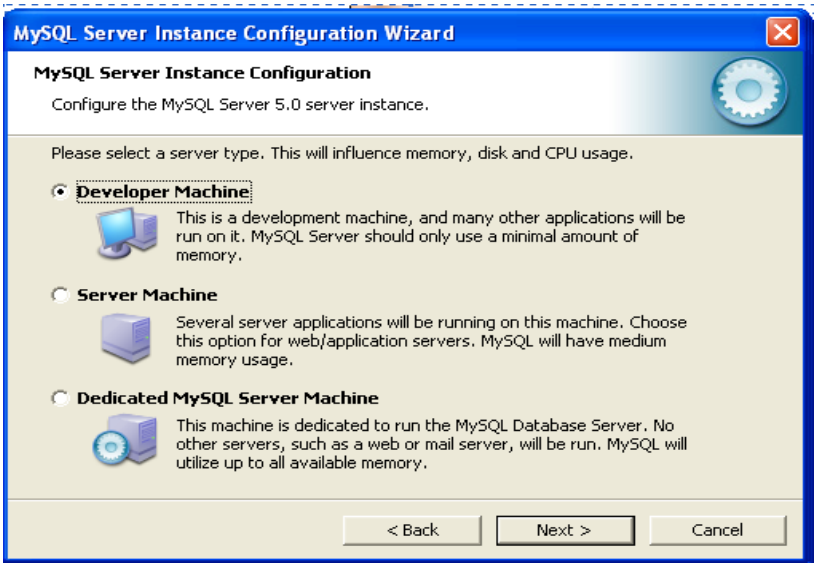


Рисунок 15.4 – Вибір інтенсивності завантаження процесора

Всі три опції різняться за інтенсивністю використання процесора, обсягом оперативної пам'яті і твердого диска. Слід обрати перший пункт, тому що при цьому MySQL займає найменший об'єм оперативної пам'яті, не заважаючи роботі інших додатків. Другий пункт призначений для серверів, на яких крім MySQL працюватимуть інші сервери, наприклад, Web-сервер або транспортний поштовий агент. Третій пункт призначений для виділеного сервера MySQL, на якому не виконуватимуть жодного додатку.

Наступне вікно (рис. 15.5) дозволяє обрати кращий тип для таблиць, який призначається за замовчуванням. Слід залишити перший пункт.

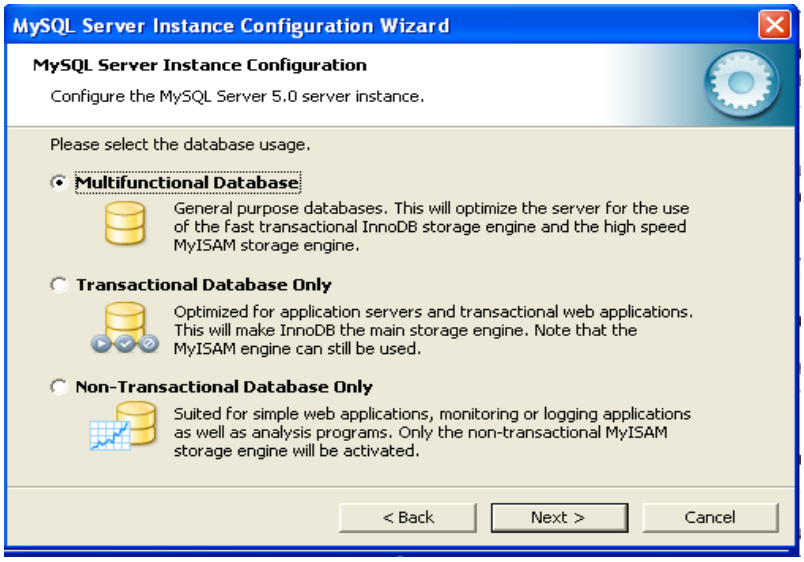


Рисунок 15.5 – Вибір типу створюваних таблиць

У цьому вікні обрати диск для зберігання datafile (рис. 15.6)

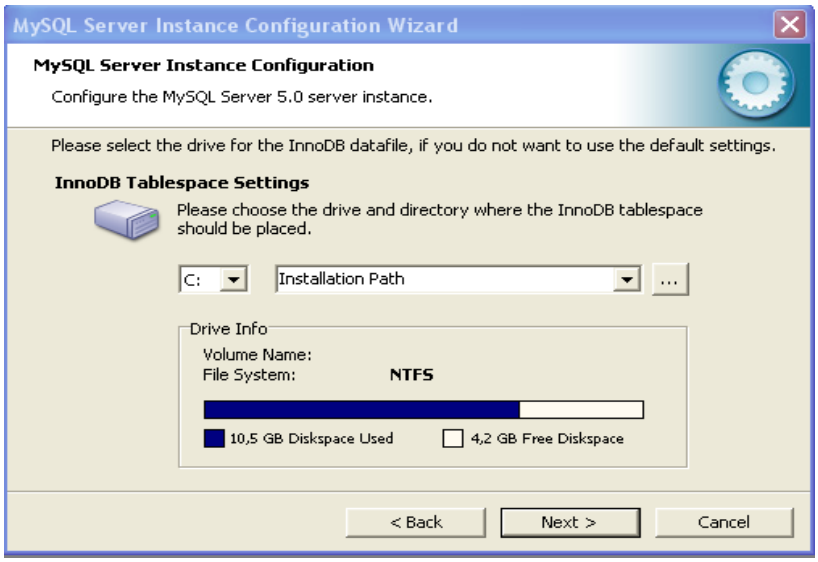


Рисунок 15.6 – Вибір диска для збереження datafile

У даному вікні (рис. 15.7) обрати максимальну кількість клієнтів, які можуть водночас підключитися до сервера. Перший пункт (рекомендується обрати саме його) передбачає, що кількість таких з'єднань не перевищуватиме 20, другий пункт допускає 500 з'єднань з сервером, третій пункт дозволяє призначати власну межу для числа активних сполук.

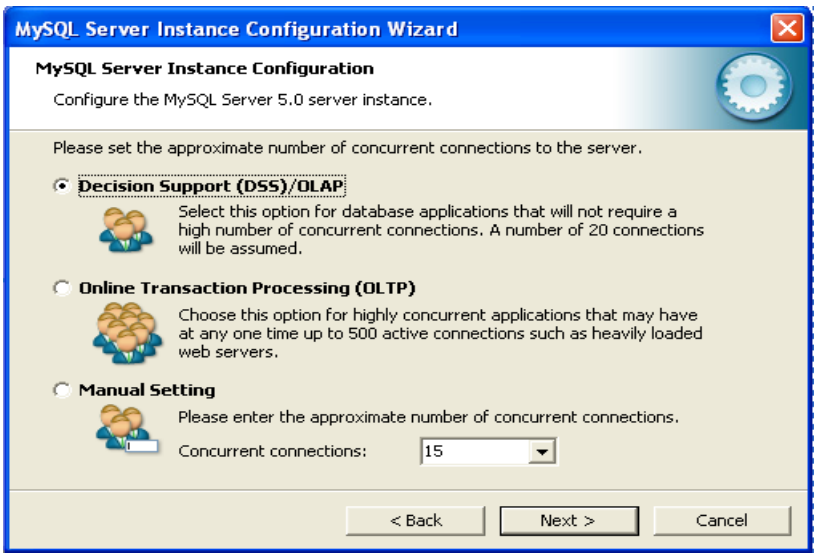


Рисунок 15.7 – Вибір числа клієнтів водночас підключених до сервера

У наступному вікні (рис. 15.8) встановити номер порту, за яким відбуватиметься з'єднання клієнтів з MySQL сервером (за замовчуванням − 3306). Якщо на комп'ютері немає інших баз даних MySQL або інших серверів, які працюють з цим портом, рекомендується залишити це значення за замовчуванням, тому що порт 3306 є стандартним для MySQL.

У тому випадку, якщо необхідно запускати MySQL 5 спільно з іншими версіями MySQL, можна обрати інший номер порту, відмінний від тих, за якими відбувається звернення до інших MySQL-серверів.

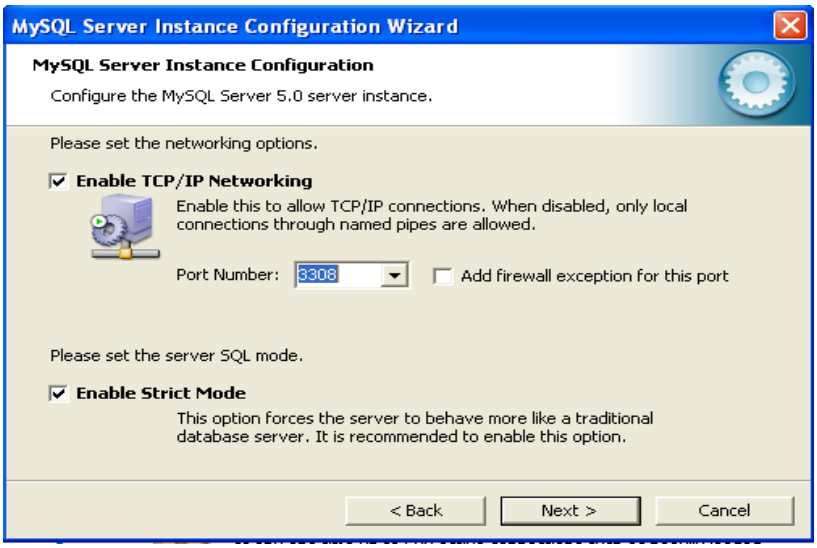


Рисунок 15.8 – Вибір робочого порту для MySQL 5

У наступному вікні (рис. 15.9) пропонується вказати кодування за замовчуванням – latin1. Для вибору коду «ср 1251» Windows-кодування. Необхідно обрати третій пункт (ручний вибір кодування) і в випадаючому вікні виділити «ср 1251».

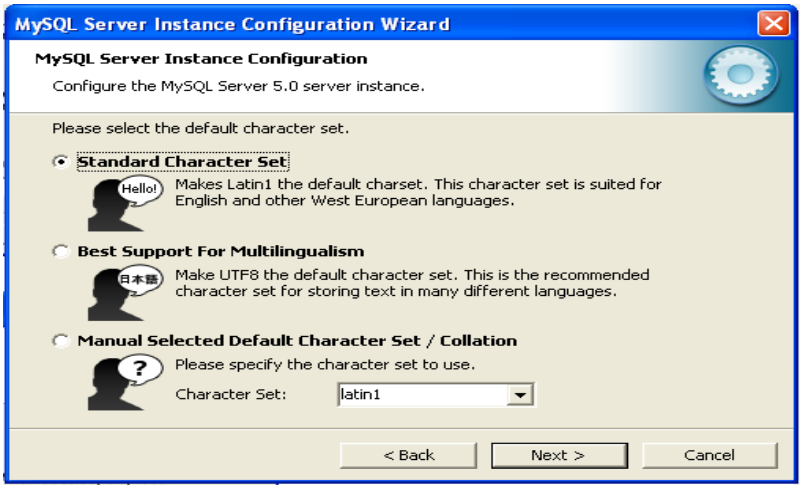


Рисунок 15.9 – Вибір коду кодування

Вікно, яке відкриється далі (рис. 15.10), призначено для інсталяції MySQL як сервісу, що забезпечить запуск mysqld зі стартом системи і коректне завершення роботи сервера із вимиканням комп'ютера. Установка прапорця «Install As Window Service» дозволяє встановити сервіс з ім'ям, яке слід обрати у списку, що випадає «Service Name». Можна змінити ім'я сервісу, особливо якщо в системі є встановлений MySQL – сервер більш ранньої версії, це дозволить уникнути конфліктів при запуску серверів як MySQL 5, так і більш ранніх версій сервера.

Відмітка прапорця «Launch the MySQL Server automatic» дозволяє налаштувати сервіс на автоматичний режим роботи, коли запуск сервера відбувається зі стартом системи. А зупинка – із завершенням роботи системи. В іншому випадку буде потрібно ручний запуск і зупинка сервера (змінити режим роботи сервісу в будь-який момент можна в консолі управління сервісами).

Рисунок 15.10 – Організація режиму запуску і зупинки MySQL Server

Прапорець «Include Bin Directory in Windows PATH» дозволяє прописати шлях до каталогу C: \ mysql5 \ bin і системної директорії PATH, що може бути зручним при частому використанні утиліт з цієї директорії.

У наступному вікні (рис. 15.11) проводиться налаштування облікових записів. Рекомендується зняти прапорець «Modify Security Settings» і залишити ці налаштування. І далі натиснути кнопку «Next».

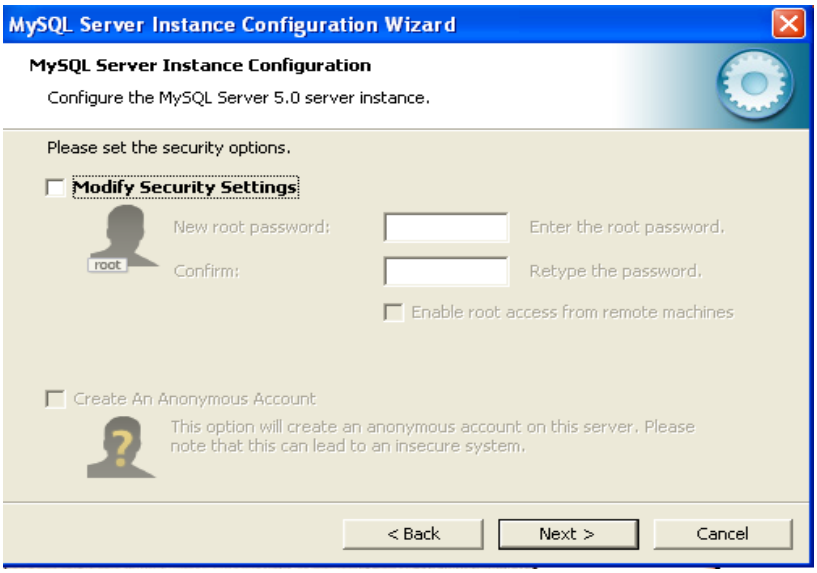


Рисунок 15.11 – Налаштування облікових записів

Після натиска кнопки «Execute» у цьому вікні (рис. 15.12) буде створений конфігураційний файл C:\mysql\my.ini і запущено сервер MySQL.

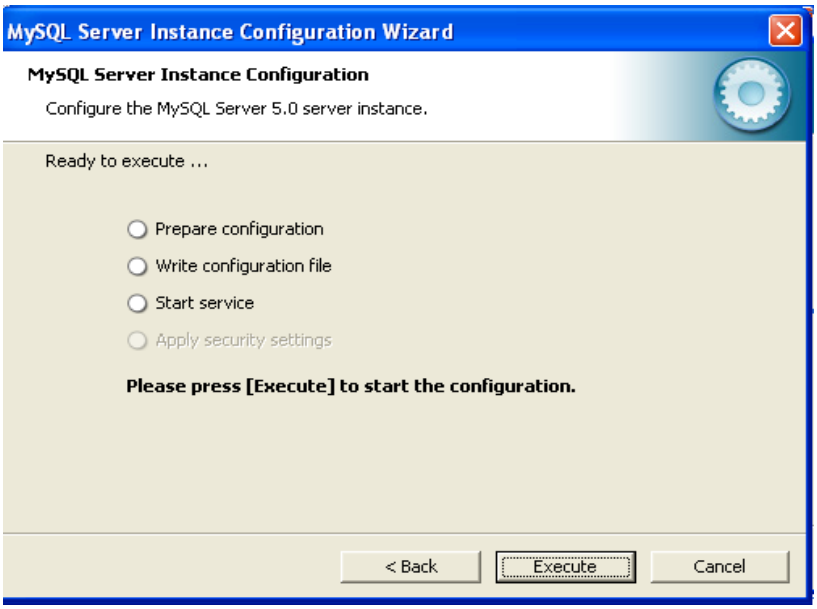


Рисунок 15.12 – Завершення конфігурування MySQL сервера

15.3 Типи даних

Типи даних регламентовані стандартом ISO SQL 92, це такі типи.

Числові дані - до них належать цілі числа, які не містять дробової частини (наприклад, 124), а також дійсні числа, що складаються з послідовності цифр, розділених крапкою (наприклад, 56.45). Числові дані поділяються на точкові (BOOLEAN, INTEGER I DECIMAL) I наближені (FLOAT, REAL I DOUBLE PRECISION).

Строкові дані – послідовність символів, поміщених в одинарні або подвійні лапки : 'Hello world', '123', " MySQL". Як стандарт в SQL визначаються одинарні лапки, тому для сумісності з іншими базами даних рекомендується використати саме їх.

Найпоширеніші типи текстових даних – це CHAR і VARCHAR.

У VARCHAR додається ще один байт, який вказує реальну довжину поля

Під час вибору рядкового типу даних для стовпця слід брати до уваги, що для змінних рядків VARCHAR потрібно знати кількість символів, що дорівнює довжині рядка плюс один байт, в той час як тип CHAR (M), незалежно від довжини рядка, використовує для її зберігання всі M символів . Водночас тип CHAR обробляється ефективніше змінних типів, оскільки завжди заздалегідь відомо, де закінчується черговий блок даних.

Зі створенням таблиці не можна змішувати стовпці типу CHAR і VARCHAR, якщо таке відбувається, СУБД MySQL змінить тип стовпців відповідно до правила: у разі, коли в таблиці є хоча один стовпець змінної довжини, всі стовпці типу CHAR приводяться до типу VARCHAR.

Таблиця 15.1 – Порівняння типів CHAR і VARCHAR

Значення	CHAR(4)		VARCHAR(4)	
	наведено	число байт	наведено	число байт
‘ ’	‘ ‘	4 байта	‘ ’	1 байт
‘ab’	'ab’	4 байта	'ab'	3 байта
'abcd'	‘abcd’	4 байта	'abcd'	5 байт
'abcdfqh'	‘abcd'	4 байта	‘abcd'	5 байт

Типи в BLOB і TEXT в СУБД MySQL в усьому аналогічні і відрізняються тільки в деталях. Наприклад, під час виконання операцій над стовпцями типу TEXT враховується кодування, а типу BLOB – немає.

Тип TEXT зазвичай використовується для зберігання великих обсягів тексту, в той час як BLOB – для великих двійкових об'єктів, таких як електронні документи, зображення, звуки тощо.

До особливих типів даних відносяться ENUM I SET. Рядки цих типів приймають значення із заздалегідь визначеного списку допустимих значень. Основна відмінність між ними полягає в тому, що значення типу ENUM має

містити саме одне значення із зазначеної безлічі, тоді як стовпці SET можуть містити будь-який або всі елементи заздалегідь заданої множини водночас.

Календарні дані – спеціальний тип для позначення дати і часу, може приймати різну форму, наприклад рядкову "2005-04-28" або числову 20050128 єдиному внутрішньому форматі, що дозволяє здійснювати операції додавання і віднімання, незалежно від зовнішнього уявлення.

NULL – спеціальний тип даних, що позначає відсутність інформації.

15.4 Основні команди

Робота з метаданими.

1. Дізнатися, які бази існують на сервері даних

« SHOW DATABASES; »

У відповідь буде отримана таблиця зі списком базами

2. Визначити, яка з баз обрана для роботи

« SHOW DATABASES ();»

У відповідь буде отримана таблиця із зазначенням працюючої бази, якщо такої немає, то список буде порожній.

3. З'ясувати, які таблиці існують у поточній базі даних

« SHOW TABLES; »

У відповідь буде отримана таблиця зі списком таблиць.

Дізнатися про структуру таблиць можна за допомогою команди DESCRIBE.

4. « DESCRIBE ім'я таблиці; »

У відповідь буде отримана таблиця з такими полями:

Field – ім'я стовпця (атрибута);

Type – тип даних для цього стовпця;

Null – вказується, чи може даний стовпець містити значення;

Null Key – чи є цей стовпець ключем;

Default – вказується значення цього стовпця за замовчуванням;

5. Якщо в таблиці створені індекси (ключі), то інформацію про них можна отримати за допомогою

« SHOW INDEX FROM ім'я таблиці; »

Створення бази даних

« CREATE DATABASE [IF NOT EXIST] ім'я бази ; »

Створити базу даних, якщо не існує

Видалити базу даних

« DROP DATABASE [IF EXIST] ім'я бази; »

Видалити базу даних, якщо існує – видаляється вся база даних з існуючими в ній таблицями.

Створення таблиць.

Перед початком створення таблиці необхідно викликати базу даних, для якої створюватиметься таблиця командою « USE ім'я БД ; » Або явно вказати ім'я бази даних –

Імя_БД.Імя_табл

« CREATE DATATABLE [IF NOT EXIST] Імя_БД.Імя_табл (col_name type [NULL | NOT NULL] [DEFAULT default_value] [AUTO_INCREMENT]
[primary або foreign KEY (col_name)]); "

Більш складні завдання із введенням значень для поля можна організувати за допомогою enum

enum_field ENUM (' first ', ' second ', ' third ')

Enum_field– поле типу ENUM, яке може приймати одне з трьох значень :
' first ', ' second ' або ' third ' ;

I SET

set_field SET (' a ', ' b ', ' c ', ' d ') Set_field – поле типу SET, яке може приймати одну з комбінацій множини (' a ', ' b ', ' з ', ' d ').

Показати таблицю.

SELECT * FROM ім'я таблиці

Додавання даних в таблицю.

INSERT INTO ім'я таблиці (атрибут 1, атрибут 2,... атрибут n) values (' значення1 ', ' значення2 ',..., ' значення n ') ;

Видалити таблицю.

« DROP TABLE [IF EXIST] Імя_БД.Імя_табл »

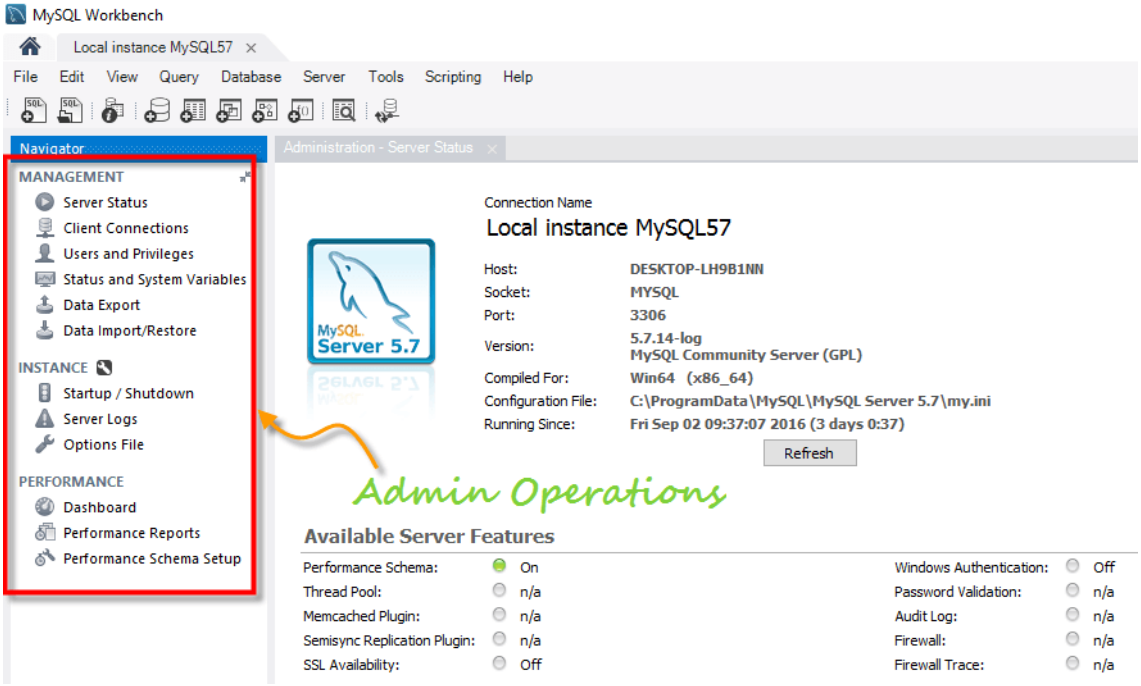
15.3 Запитання та завдання для самоконтролю

1. Скільки рівнів має логічна архітектура MySQL? Назвіть їх.
2. Чим відрізніється процедура інсталяції додатку від процедури конфігурування додатку?
3. Назвіть оптимальне місце інсталяції сервера, клієнтських програм, ….
4. На що впливає значення опції «Concurrent connections»?
5. Які номери портів можна підключити для роботи MySQL?
6. З якої причини необхідно активувати опцію «add firewall …»?

Основи роботи з MySQL Workbench

Для проектування складної бази даних з великою кількістю таблиць і зв'язків, можливостей phpMyAdmin катастрофічно не вистачає. Тому зазвичай використовується продукт MySQL Workbench – чудова безкоштовна програма для роботи з БД MySQL.

MySQL Workbench – інструмент для візуального проектування баз даних, що інтегрує проектування, моделювання, створення та експлуатацію БД в єдине безшовне оточення для системи баз даних MySQL.



Особливості MySQL Workbench дозволяють користувачеві створювати та керувати підключеннями до сервера баз даних та запускати SQL-запити на цих підключеннях до бази даних за допомогою вбудованого редактора SQL.

MySQL Workbench базується на п’яти важливих моментах:

- **Розробка SQL:** дозволяє створювати та керувати підключеннями до серверів баз даних. На додаток до того, що дозволяє користувачеві налаштовувати параметри підключення, MySQL Workbench надає можливість запускати запити SQL на підключеннях до бази даних за допомогою вбудованого редактора SQL.
- **Моделювання даних (дизайн):** дозволяє графічно моделювати схему бази даних, здійснювати зворотне та пряме проектування між схемою та активною базою даних та редагувати всі аспекти бази даних за допомогою всеосяжного редактора таблиць. Редактор таблиць надає зручні засоби для редагування таблиць, стовпців, індексів, тригерів, розділів, параметрів, вставок та привілеїв, процедур та подань.
- **Адміністрування сервера:** Дозволяє керувати екземплярами сервера MySQL, керуючи користувачами, виконуючи резервне копіювання та відновлення, перевіряючи дані аудиту, переглядаючи стан бази даних та контролюючи продуктивність сервера MySQL.
- **Міграція даних:** дозволяє переходити з Microsoft SQL Server, Microsoft Access, Sybase ASE, SQLite, SQL Anywhere, PostreSQL та інших таблиць, об’єктів та даних RDBMS до MySQL. Міграція також підтримує перехід зі старих версій MySQL на останні версії.
- **Підтримка MySQL Enterprise:** підтримка корпоративних продуктів, таких як MySQL Enterprise Backup, MySQL Firewall та MySQL Audit.

Все це дозволяє архітекторам даних візуалізувати вимоги, спілкуватися із зацікавленими сторонами та вирішувати проблеми проектування до того, як будуть зроблені великі інвестиції часу та ресурсів. Це дозволяє моделювати базу даних на основі моделі, яка є найефективнішою методологією для створення дійсних і ефективних баз даних, забезпечуючи при цьому гнучкість відповідати зростаючим вимогам бізнесу.

Утиліти перевірки моделей та схем дотримуються стандартів найкращої практики моделювання даних, а також застосовують специфічні для MySQL стандарти фізичного проектування, щоб не було помилок під час створення нових ER діаграм або створення фізичних баз даних MySQL.

Стартовий екран програми відображає основні напрямки її функціональності – проектування моделей баз даних та їх адміністрування (рис. 4.19).

Рис. 4.19. Стартове вікно MySQL Workbench

MySQL Workbench надає можливість роботи як з локальними БД, як з БД на SQL-сервері, так і хмарними серверами SQL, розміщених в кластерах.

Оскільки клієнт-серверні БД передбачають розміщення власне БД на віддаленому SQL-сервері, тому роботу з локальними БД опустимо.

Якщо БД на SQL-сервері створена, тоді для роботи необхідно встановити зв’язок з нею. Якщо ж БД на сервері немає, тоді необхідно, щоб її створив користувач з правами системного адміністратора зазвичай за допомогою phpMyAdmin – безкоштовного додатка, який призначений для адміністрування СКБД MySQL.

Проектування моделі бази даних

Після того, як логічна модель БД розроблена і нормалізована, MySQL Workbench надає декілька сервісів для фізичної реалізації таблиць БД на SQL-сервері:

- при відкритому з'єднанні безпосередньо на сервері за допомогою конструктора (рис. 4.20);

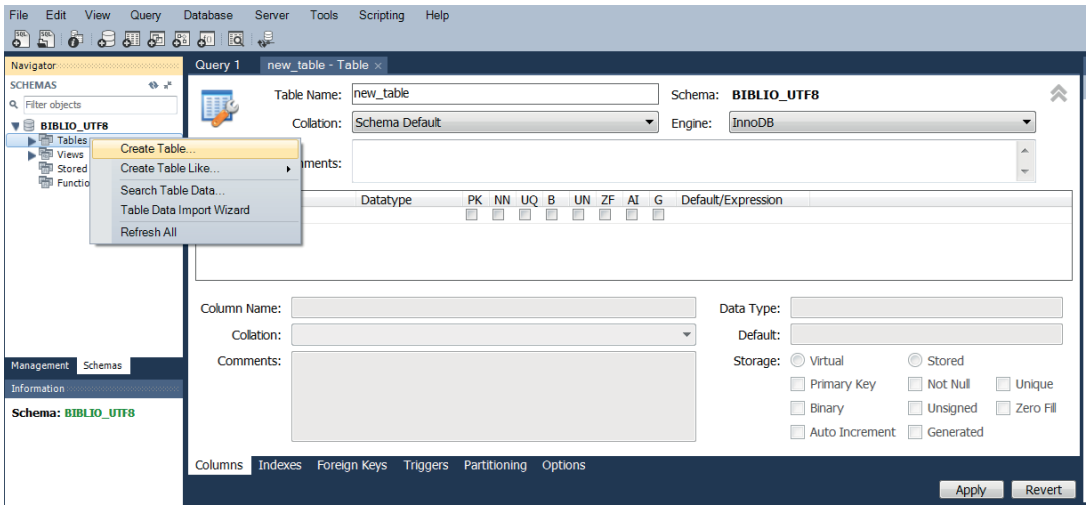


Рис. 4.20. Фізична реалізація БД при відкритому з'єднанні безпосередньо на сервері за допомогою конструктора

- при відкритому з'єднанні безпосередньо на сервері у вікні Query за допомогою SQL-команди CREATE TABLE… (рис. 4.21);

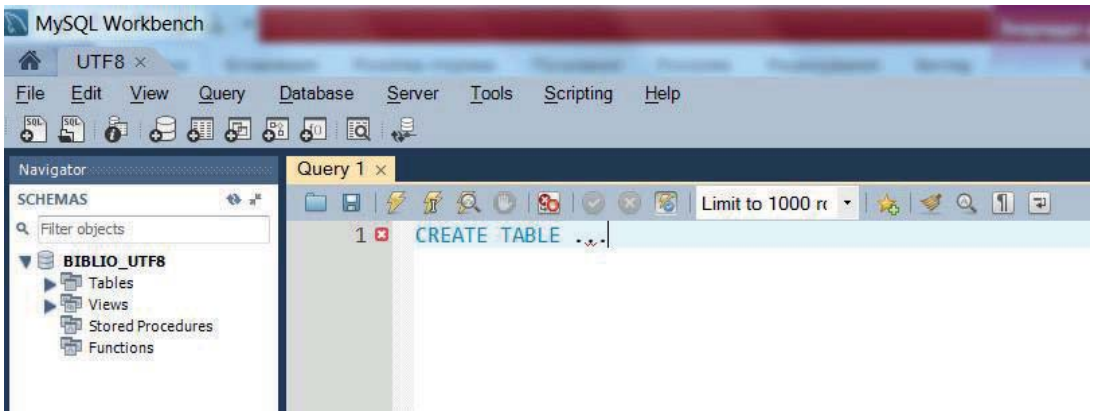


Рис. 4.21. Фізична реалізація БД при відкритому з'єднанні безпосередньо на сервері за допомогою виконання SQL-команд CREATE TABLE…

- при закритому з’єднанні у конструкторі моделі БД (рис 4.22) з наступним імпортом цієї схеми даних з на SQL-сервер.

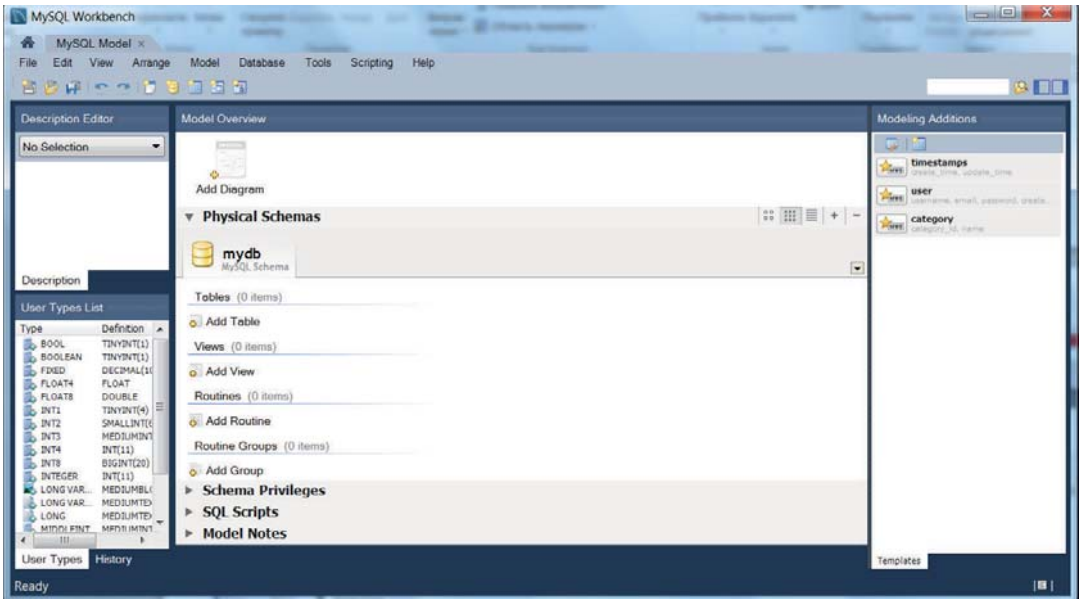


Рис. 4.22. Фізична реалізація БД при закритому з’єднанні з SQL-сервером

Реалізовану таким чином БД можна імпортувати на SQL-сервер за допомогою таких сервісів:

- синхронізувати створену модель з базою даних на сервері;
- використати майстер побудови бази даних;
- почергово скопіювати код створення таблиць у буфер обміну та запустити його на виконання у вікні SQL Editor, тощо.

Таким чином, можливе поєднання способів фізичної реалізації БД, зокрема, можна створити модель БД при закритому з’єднанні та запустити на виконання скопійовані (та відредаговані) команди створення таблиць CREATE TABLE… вже при відкритому з’єднанні з SQL-сервером.

Варто зазначити, що таке поєднання способів фізичної реалізації БД надає додаткові можливості:

- засоби для візуального конструювання таблиць, стовпців, індексів, тригерів, привілеїв, процедур та переглядів;
- візуального конструювання зв’язків між таблицями;

- створення ER діаграми з створеної моделі даних, що дозволяє наочно уявити модель бази даних у графічному вигляді.

Як вже зазначалося вище, все це дозволяє візуалізувати вимоги, спілкуватися із зацікавленими сторонами та вирішувати проблеми проектування до того, як будуть зроблені великі інвестиції часу та ресурсів.

Адже від того, як спроектована база даних, значно залежить в подальшому робота з нею.

4.3.3.1. Створення нової моделі БД

Першим кроком роботи у MySQL Workbench є створення нової моделі БД за командою File – New Model або комбінацією клавіш Ctrl + N. При цьому відкриється конструктор нової моделі mydb (рис. 4.23).

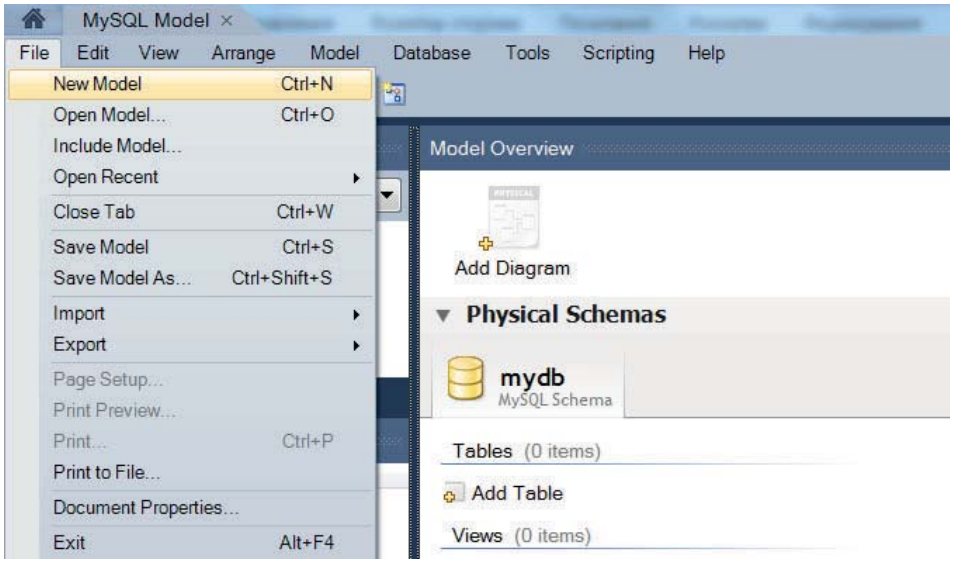


Рис. 4.23. Створення нової моделі бази даних у MySQL Workbench

У зв'язку з тим, що код створення таблиць буде генеруватися, необхідно, щоб ім'я моделі співпадало з іменем бази даних на SQL-сервері. У нашому випадку ім'я БД та ім'я користувача user співпадатимуть.

Тому необхідно для нової схеми mydb відкрити вікно редагування параметрів схеми подвійним кліком або за допомогою контекстного меню Edit Schema…, у якому задати необхідне ім'я БД (рис. 4.24).

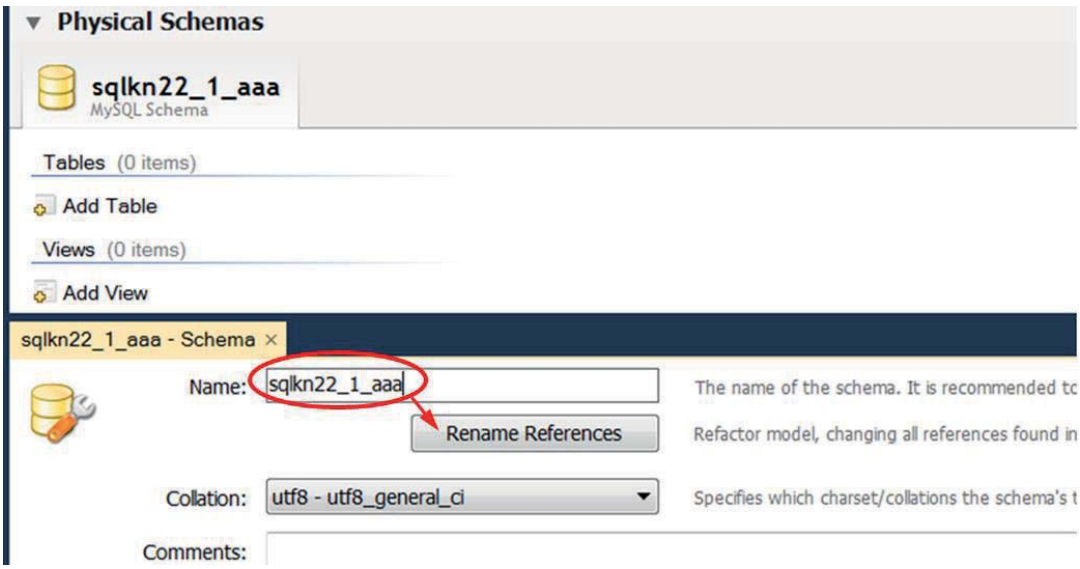


Рис. 4.24. Перейменування моделі бази даних

Як видно з рис. 4.24, кодова таблиця символів, що підтримує кирилицю, за замовчуванням utf8. Забігаючи наперед, варто зазначити, що при створенні таблиць необхідно вказувати ту ж кодову таблицю символів.

4.3.3.2. Додавання таблиць

Список баз даних проекту і список таблиць в межах бази даних буде розташовуватися у вкладці Physical Schema.

Імена таблиць можуть мати до 128 символів. Вони повинні бути унікальними по відношенню до користувача – тобто дві таблиці з однаковими іменами не можуть належати одній БД, проте в іншій БД може бути таблиця з тим же іменем.

Ім’я таблиці повинно записуватися згідно синтаксису запису імен змінних у будь-якій мові програмування:

- починатися з латинської букви, далі іде послідовність латинських букв, цифр і знаку підкреслення;
- не містити розділових і спеціальних символів, наприклад, %, (, /, «, !, ?, тощо;
- не містити пробілів, які можна замінити знаком підкреслення;
- не дублювати команди мови SQL, наприклад, SELECT, CREATE, DROP, ALTER, INSERT, UPDATE, DELETE і т.д.;

- не дублювати імена системних змінних, таких як USER, DATABASE, PASSWORD, FLUSH, PORT, SOCKET, VERSION і т.д.;
- не дублювати назви типів даних – INT, DECIMAL, REAL, FLOAT, DATE, TIME і т.д.;
- не дублювати назви функцій – MIN. SUM, COUNT, SIN, YEAR, LENGTH, TRIM, UPPER, тощо.

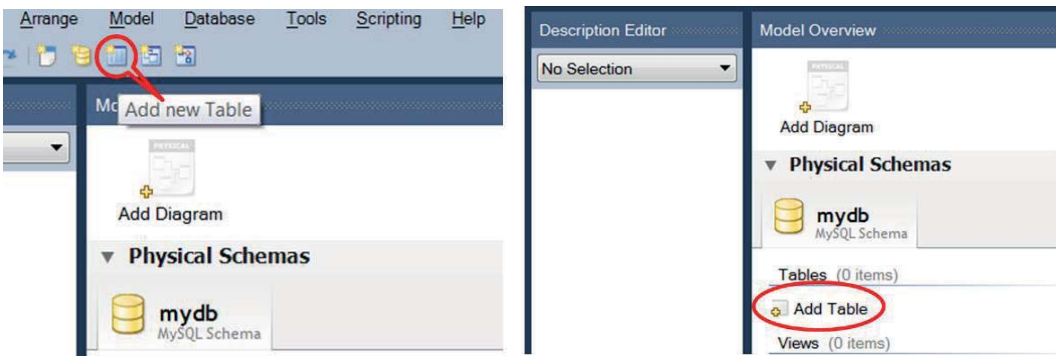
Приклади імен таблиць: Katalog001, TypeWare, Class_Animal тощо.

Варто також зазначити, що на імена полів поширюються ці ж обмеження!

Кожне поле визначається набором властивостей:

- Column Name – ім'я поля;
- Datatype – тип поля (тип даних);
- PK – PRIMARY KEY (поле первинного ключа);
- NN – NOT NULL (поле не може бути порожнім);
- UQ – UNIQUE INDEX (поле повинно містити унікальні значення);
- BIN – BINARY (дані у вигляді двійкових рядків (binary string). Сортування і порівняння засноване на числових значеннях байтів);
- UN – UNSIGNED (цілі без знакові значення);
- ZF – ZEROFILL (дозаповнення нулями – добавляє до значення нулі ліворуч, якщо довжина значення менше довжини поля. Наприклад, якщо довжина поля INT(5), тоді кожне значення дозаповнюється до 5 символів, 12 = 00012, 400 = 00400 і т.д.);
- AI – AUTO INCRIMENT (лічильник++);
- Default – значення за замовчуванням.

Для створення нової таблиці необхідно вибрати команду з панелі інструментів Add new Table (рис. 4.25 а) або двічі клацнути кнопкою мишки на команді "+Add Table", розміщеної в області фізичної схеми (рис. 4.25 б).



а) *б)*

Рис. 4.25. Створення нової таблиці у схемі БД:

а) кнопкою з панелі інструментів; б) кнопкою в області фізичної схеми

У відкритій закладці конструктора необхідно ввести ім’я таблиці, а також назви полів (стовпців, атрибутів) та задати їх властивості (рис. 4.26).

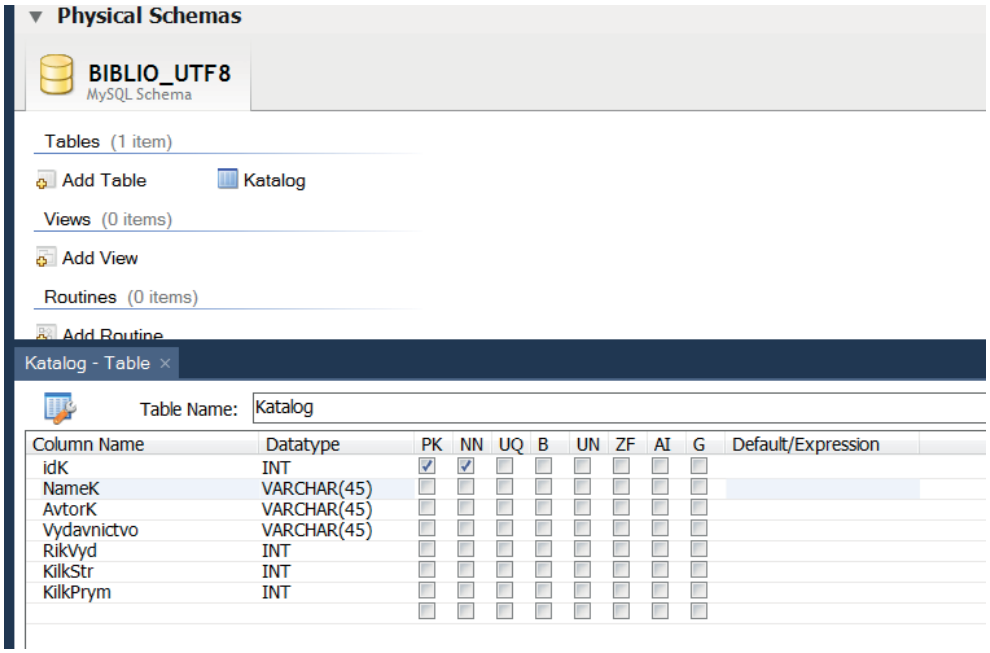


Рис. 4.26. Створення нової таблиці у конструкторі MySQL Workbench

Аналогічно створюються всі решту таблиць БД.

4.3.3.3. Управління індексами

У СКБД MySQL індекси є основним засобом прискорення доступу до вмісту таблиць, особливо це стосується запитів, що включають об'єднання декількох таблиць.

СКБД MySQL використовує індекси в декількох аспектах:

- для пошуку рядків, що відповідають певним умовам або рядків, що мають відповідності в інших таблицях при виконанні об'єднання;
- для прискорення пошуку максимального або мінімального значення індексованого стовпця при роботі з функціями MIN() або MAX();
- для прискорення сортування.

Додавати, видаляти і редагувати індекси таблиць можна у вкладці "Indexes" інтерфейсу управління таблицею (рис. 4.27): вводимо назву індексу, вибираємо його тип, потім помічаємо в потрібному порядку список полів, що беруть участь в даному індексі. Порядок полів буде відповідати порядку, в якому були проставлені галочки.

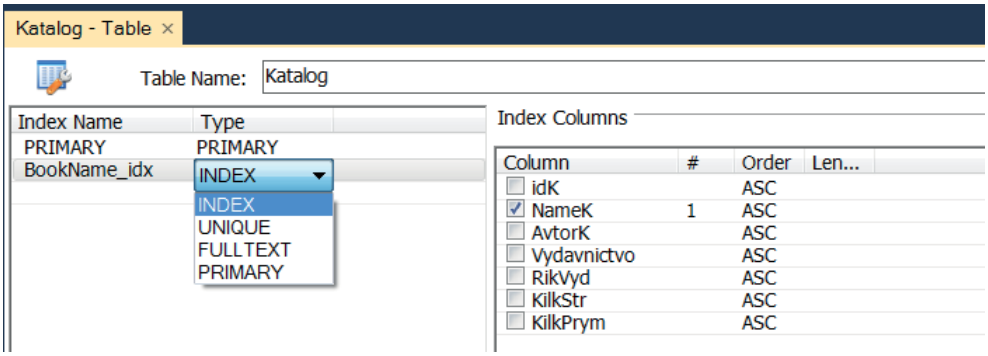


Рис. 4.27. Створення індексів у конструкторі MySQL Workbench

В даному прикладі додано унікальний індекс userIndex до поля nameuser.

Індекс з назвою PRIMARY і типом PRIMARY був автоматично створений за ключовим полем idK, оскільки під час створення структури таблиці поле idK було помічено як поле первинного ключа даної таблиці (рис. 4.26).

Існує декілька типів індексів:

- INDEX або KEY;
- UNIQUE;

- FULLTEXT;
- SPATIAL;
- PRIMARY.

Індекси типу PRIMARY, INDEX і UNIQUE зберігаються у вигляді B-дерев. Рядки автоматично стискаються з видаленням пропусків в префіксах і кінцевих пробілів.

Тип INDEX або KEY відноситься до нормального неоднозначною індексу. Індекс може містити рядки з однаковими значеннями. Цей тип індексів використовується тільки для того, щоб деякі запити працювали швидко.

UNIQUE відноситься до типу, де всі рядки індексу повинні бути унікальним. А також використовується для прискорення запитів. Тип UNIQUE можна використовувати до полів, в яких дозволено значення NULL (вважається, що NULL не дорівнює самому собі), в цьому випадку два рядки можуть бути ідентичними, якщо вони обидва містять значення NULL. Щоб запобігти цьому, необхідно при створенні таблиці встановити для поля атрибут NOT NULL.

PRIMARY діє точно так же, як UNIQUE індекс, за винятком того, що він завжди називається «PRIMARY» і може бути тільки одним (і завжди повинен бути один). Первинний індекс призначений в якості основного засобу, щоб однозначно ідентифікувати будь-який рядок в таблиці. Деякі СКБД (наприклад, InnoDB в MySQL) будуть зберігати записи на диску в тому порядку, в якому вони знаходяться в первинному індексі.

Спеціальні індекси FULLTEXT застосовуються для повнотекстного пошуку. Ці індекси підтримуються тільки таблицями типу MyISAM і вони можуть бути створені тільки з стовпців типу CHAR, VARCHAR і TEXT. Індексування завжди виконується для всього стовпця цілком, часткова індексація не підтримується.

Індекси типу SPATIAL можуть використовуватися тільки в таблицях типу MyISAM і тільки для індексування просторових даних, тобто для індексації багатовимірної інформації, такої, наприклад, як географічні дані – з двовимірними координатами широтою та довготою.

4.3.3.4. Зв'язки між таблицями

Установка *зовнішніх ключів* і зв'язування таблиць можливо тільки для таблиць InnoDB (ця система зберігання даних вибирається за замовчуванням). Для управління зв'язками в кожній таблиці знаходиться вкладка Foreign Keys.

Для додавання зв'язку між таблицями необхідно відкрити вкладку Foreign Keys дочірньої таблиці, ввести у поле Foreign Key Name ім'я зовнішнього ключа та вибрати у полі Referenced Table таблицю-батька і відповідне поле.

Крім того, у полі Foreign Key Options необхідно налаштувати поведінку зовнішнього ключа при зміні відповідного поля (ON UPDATE) і видаленні (ON DELETE) батьківського запису:

- RESTRICT – видавати помилку при зміні / видаленні батьківського запису;
- CASCADE – оновлювати зовнішній ключ при зміні батьківського запису, видаляти дочірній запис при видаленні батька;
- SET NULL – встановлювати значення зовнішнього ключа NULL при зміні / видаленні батька (неприйнятно для полів, у яких встановлено прапор NOT NULL!);
- NO ACTION – не робити нічого, однак за фактом ефект аналогічний RESTRICT.

Для прикладу, встановимо зв'язок між таблицями і Katalog та Formuljar (рис. 4. 28). Таблиця Formuljar є дочірньою по відношенню до Katalog, оскільки містить поле idK, яке не є первинним ключем, в той час як поле idK в таблиці Katalog є первинним ключем і, відповідно, таблиця Katalog є батьківською.

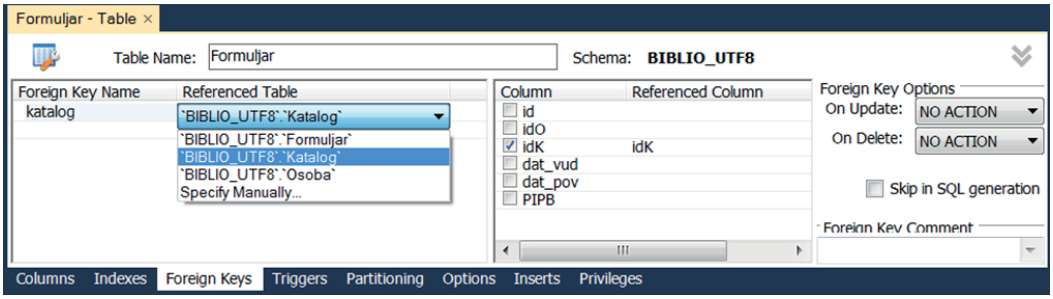


Рис. 4.28. Створення зв'язків між таблицями у конструкторі MySQL Workbench

При створенні зовнішніх ключів в дочірній таблиці автоматично створюються відповідні індекси.

4.3.3.5. Створення ER-діаграми

Модель сутність-зв’язок (ER-модель) (англ. Entity Relationship model, ERM) – модель даних, що дозволяє описувати концептуальні схеми предметної області.

ER-модель використовується при високорівневому (концептуальному) проектуванні баз даних. З її допомогою можна виділити ключові сутності (таблиці) і позначити зв’язки, які можуть встановлюватися між цими таблицями.

Зв’язки між сутностями відповідають логічним відношенням між сутностями, які встановлюють суттєвий зв’язок у даній предметній області. Наприклад, читач замовляє книгу, водій керує тролейбусом, виріб виготовлений з матеріалу.

Зв’язки мають такі характеристики:

- ступінь зв’язку (деколи називають потужність зв’язку);
- клас належності (обов’язковий або необов’язковий);
- тип зв’язку (ідентифікуючий або неідентифікуючий).

Ступінь зв’язку (потужність зв’язку) може бути один до одного (1:1), один до багатьох (1:M), багато до одного (M:1) чи багато до багатьох (M:N).

Наприклад, студент читає книжку – такий зв’язок має ступінь 1:1. У студента може бути декілька книжок, але кожна з цих книг є тільки в користуванні цього студента. Такий зв’язок має ступінь (1: M). Можлива інша ситуація, коли декілька студентів мають різні книжки, причому кожен з цих студентів може користуватися книжками друзів. Такий зв’язок матиме ступінь (M:N).

Клас належності визначає, обов’язковий є зв’язок або необов’язковий. Обов’язковий клас належності відповідає такому зв’язку, при якому кожен екземпляр сутності обов’язково повинен брати участь у зв’язку, необов’язковий клас належності – деякі екземпляри сутності можуть не брати участі в зв’язку. Наприклад, обов’язковий клас належності: кожен студент має прізвище. І приклад необов’язкового класу належності – студент має iPhone, але не кожен студент має iPhone. Студент може мати інший вид телефону.

Тип зв’язку між залежними сутностями може бути ідентифікуючим (identifying relationship) або неідентифікуючим (non-identifying relationship).

Якщо книга не може існувати без автора, а вона не може, тоді тип зв’язку "книга – автор" буде ідентифікуючим. І якщо книга може існувати без власника, а вона може, тоді тип зв’язку "книга – власник" буде неідентифікуючим.

Для представлення схеми даних, сутностей і їх зв’язків в графічному вигляді в MySQL Workbench існує редактор ER-діаграм.

Зв’язки між сутностями зображують лініями. Ідентифікуючий зв’язок показують суцільною, неідентифікуючий – пунктирною лінією.

Створюється ER-діаграма за допомогою команди головного меню Model – Create Diagram From Catalog Object. При цьому відкриється нова вкладка із зображеними таблицями і зв’язками між ними (рис. 4.29).

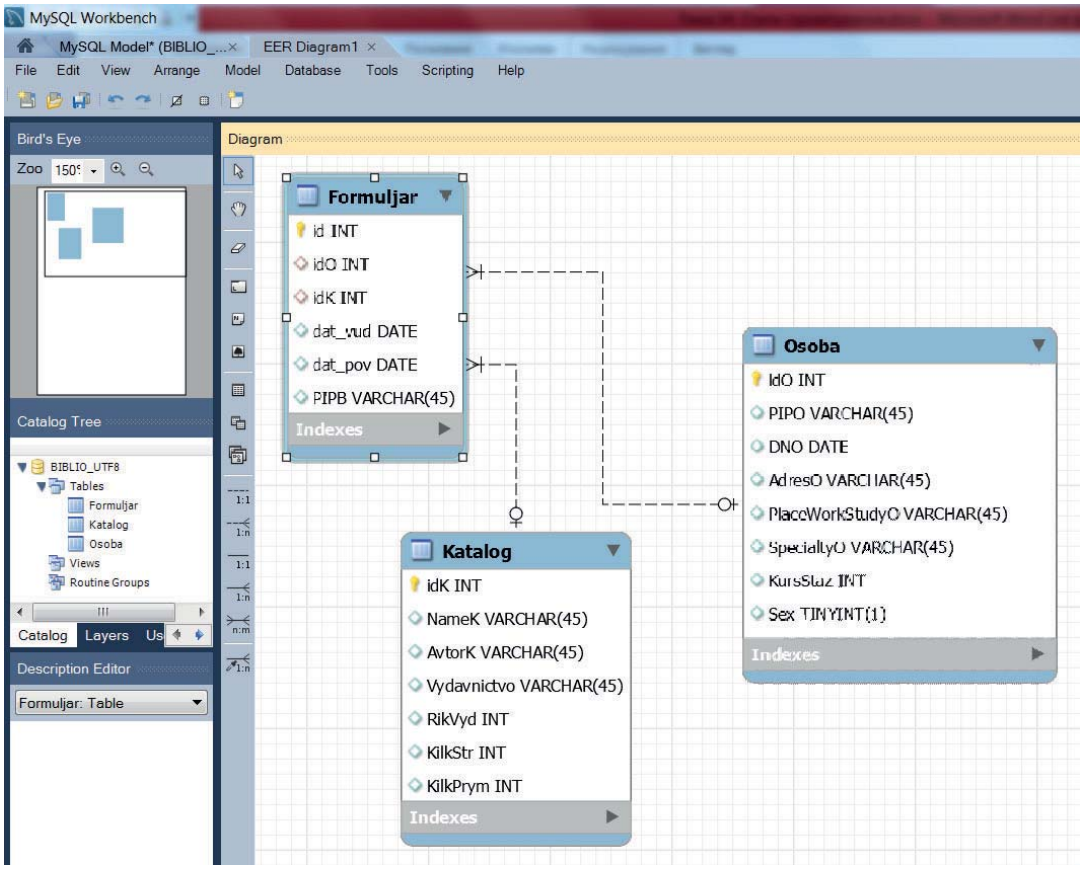


Рис. 4.29. Створення ER-діаграми у Workbench

Якщо зв’язки між таблицями встановлено на етапі створення таблиць у конструкторі (вкладка Foreign Keys, рис. 4.28), тоді MySQL Workbench при створенні ER-діаграми для цих зв’язків автоматично визначить ступінь зв’язку,

клас належності та тип і, відповідно, відобразить на діаграмі, як показано на рис. 4.29.

Щоб добавити на діаграму новостворену таблицю (в область Diagram), необхідно просто перетягнути її з вікна Catalog Tree.

Для експорту схеми даних в графічний файл необхідно вибрати "File → Export", а потім один з варіантів (PNG, SVG, PDF, PostScript File).

4.3.2. Підключення MySQL Workbench до MySQL-сервера

MySQL Workbench – інструмент для візуального проектування баз даних, що інтегрує проектування, моделювання, створення та експлуатацію БД у єдине безшовне оточення для системи баз даних MySQL.

Власне, для фізичного створення БД на MySQL-сервері, або для перенесення схеми БД з MySQL Workbench на MySQL-сервер, першим кроком є встановлення зв'язку з цим сервером, будь від локальний, віддалений чи у хмарі.

Основними параметрами для з'єднання з MySQL-сервером є:

- Ім'я з'єднання;
- Метод з'єднання;
- IP сервера;
- Порт;
- Ім'я користувача;
- Ім'я бази даних;
- Пароль користувача.

Залежно від вибору методу (протоколу) з'єднання – SSH чи SSL, кількість параметрів з'єднання може бути ще більшою.

Протоколи SSH та SSL – це криптографічні протоколи безпечного передавання даних мережею, що використовують більшість одних і тих же криптографічних примітивів, але різняться аутентифікацією за ключами.

У більшості випадків роботи з клієнт-серверними БД цілком достатньо використання стандартного протоколу передачі даних TPC/IP.

Задаються параметри з’єднання у вікні, яке викликається командою New Connection (рис. 4.30) або кнопкою ⊕ панелі MySQL Connection залежно від версії Workbench.

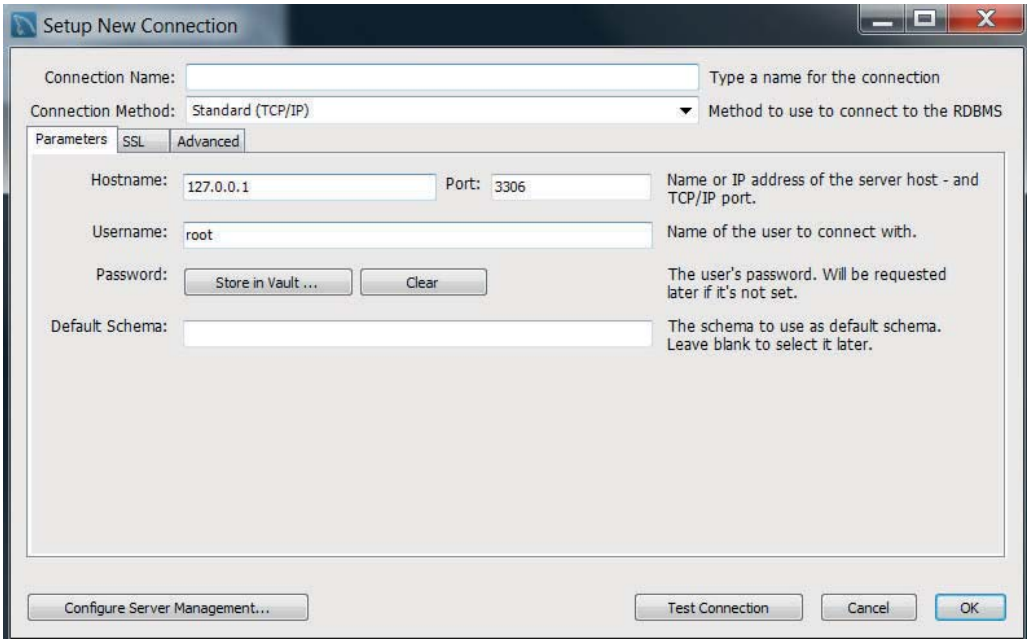


Рис. 4.30. Задавання параметрів з’єднання з сервером з MySQL Workbench

Connection Name є обов’язковим параметром та дозволяє розрізняти з’єднання при роботі з кількома серверами і має нести змістове навантаження про тип з’єднання, користувача або до якої БД здійснюється підключення.

Connection Method визначає способи передачі даних мережею. За замовчуванням пропонується стандартний протокол Standard (TPC/IP).

Hostname визначає IP адресу сервера. За замовчуванням пропонується підключення до локального сервера. Якщо ж підключення здійснюється до віддаленого MySQL сервера, тоді необхідно знати його IP адресу.

Port для MySQL сервера (локального або віддаленого) – 3306. Для хмарної СКБД MySQL порт буде іншим.

Username та Password вказує логін та пароль зареєстрованого користувача на сервері.

Default Schema є необов’язковим параметром, задає використання за замовчуванням однієї з БД при відкритому з’єднанні.

Для роботи MySQL Workbench на стороні сервера необхідно відкрити підключення до SQL-сервера командою Open Connection to Start Querying (рис. 3.19) та у полі Stored Connection вибирати необхідне під'єднання, наприклад UTF8. Для ідентифікації користувача також необхідно ввести пароль користувача на SQL-сервері.

MySQL Workbench відкриє вікно провідника (Object Browser) та вікно SQL-редактора (SQL Editor) (рис. 4.31).

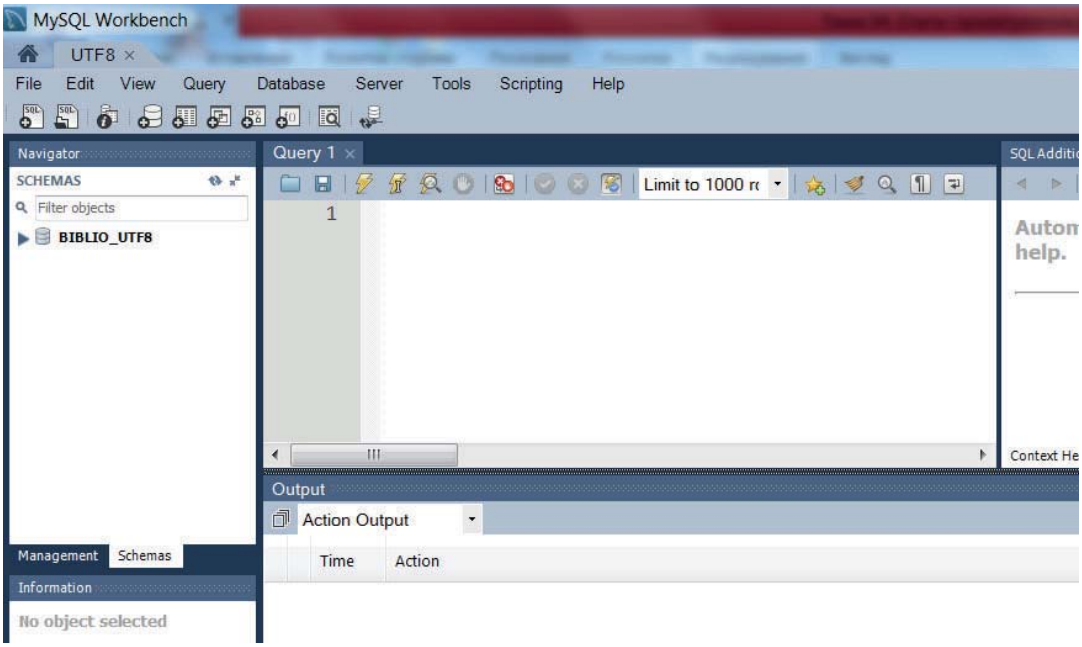


Рис. 4.31. Відкрите з'єднання MySQL Workbench з MySQL-сервером

Тепер у відкритому з'єднанні можна безпосередньо працювати з сервером:

- додавати, змінювати та видаляти бази даних, таблиці, перегляди, збережені процедури (в тому числі транзакції), тригери, тощо;
- додавати, редагувати, видаляти дані та здійснювати вибірки даних з таблиць;
- додавати нових користувачів, змінювати привілеї та паролі наявних користувачів, видаляти користувачів, здійснювати копіювання та відновлення БД, тощо.

Всі ці дії можна реалізовувати як за допомогою меню та відповідних піктограм, так і за допомогою SQL команд у вікні Query.

Основи мови SQL розглянемо в наступних темах, а зараз – етапи створення нових таблиць та зв'язків між ними засобами моделювання Workbench.

4.3.3. Імпорт схеми даних з MySQL WorkBench на MySQL-сервер

Створити БД на віддаленому SQL-сервері на основі розробленої схеми БД в MySQL Workbench можна декількома способами:

- синхронізувати створену модель з базою даних на сервері;
- використати майстер побудови бази даних;
- почергово скопіювати код створення таблиць у буфер обміну та запустити його на виконання у вікні SQL Editor, тощо.

4.3.3.1. Синхронізація моделі з базою даних на сервері

При здійсненні синхронізації MySQL Workbench надає можливість не тільки створення на сервері нових таблиць, переглядів та процедур, але і можливість модифікувати БД на сервері шляхом зміни графічної схеми БД.

Додаючи нові або змінюючи наявні об'єкти (таблиці, перегляди, процедури, а також окремі стовпці таблиць і т.д.) у графічному редакторі, дуже просто змінити так само БД на сервері шляхом синхронізації.

Синхронізація моделі та БД на сервері реалізується командою Database – Synchronize Model… з вікна MySQL Model або EER Diagram (рис. 4.32) та складається з декількох кроків:

- встановлення параметрів з'єднання з SQL сервером шляхом вибору наявного збереженого з'єднання в полі Stored Connection;
- задавання опцій генерації SQL команд зміни БД згідно змін в моделі;
- перевірка коректності підключення до SQL-сервера;
- вибір БД на сервері для синхронізації з моделлю;
- узгодження схеми даних і структури БД на сервері;
- визначення відмінностей між графічною схемою даних і структурою БД на сервері;
- генерація SQL коду приведення БД на сервері до графічної схеми даних;
- виконання згенерованого коду (Execute).

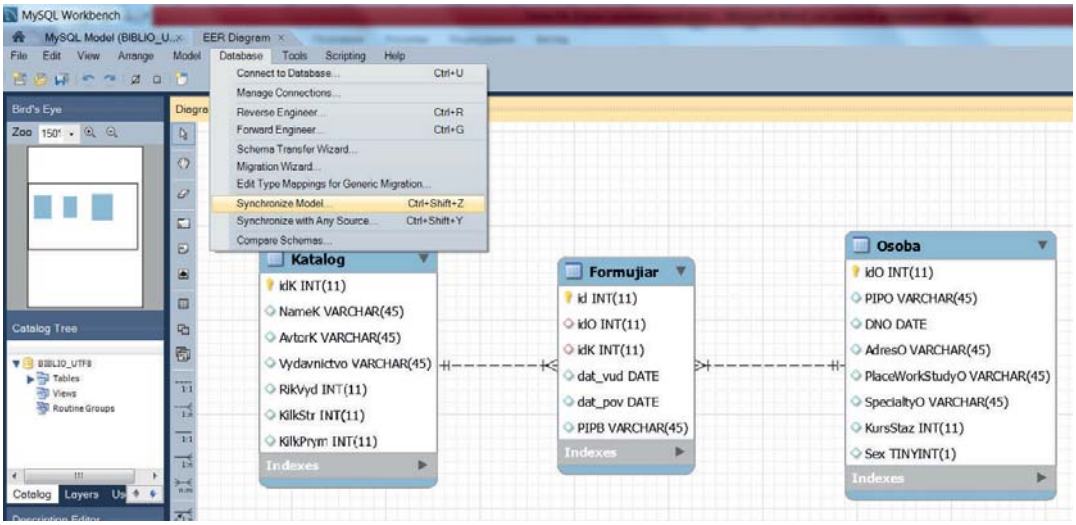


Рис. 4.32. Синхронізація моделі даних та БД на MySQL сервері

Згенерований код за цим способом можна також редагувати і потім виконувати, можна зберегти у файл (Save to File) або скопіювати в буфер обміну (Copy to Clipboard).

Синхронізацію проводиться кожен раз після будь-яких змін у моделі даних, якщо ці зміни необхідно відобразити на сервері.

4.3.3.2. Використання майстра побудови бази даних

Для фізичної реалізації БД на стороні сервера також можна викликати майстер побудови бази даних командою Database – Forward Engineer (рис. 4.33) з вікна MySQL Model або EER Diagram.

Робота майстра здійснюється декілька кроків:

- встановлення параметрів з'єднання з SQL сервером (аналогічно першому кроку синхронізації моделі та БД);
- задавання опцій створення БД. При повторному створенні бази даних для видалення старих таблиць необхідно включити прапорці DROP Objects Before Each CREATE Object та Generate DROP SCHEMA;
- вибір об'єктів (таблиць, переглядів, процедур, тригерів та користувачів) з моделі для створення на їх сервері;
- генерація SQL команд з попереднім переглядом. Як і при синхронізації, так і тут можна редагувати згенеровані SQL команди ще до їх виконання.

Згенеровані SQL команди можна також зберегти у файл (Save to File) або скопіювати в буфер обміну (Copy to Clipboard);

- з'єднання з сервером та виконання SQL команд.

Якщо помилки відсутні, тоді отримаємо повідомлення про успішне завершення роботи майстра побудови бази даних (Forward Engineer Finished Successfully). В іншому випадку, щоб переглянути протокол помилок, необхідно скористатися командою Show Logs.

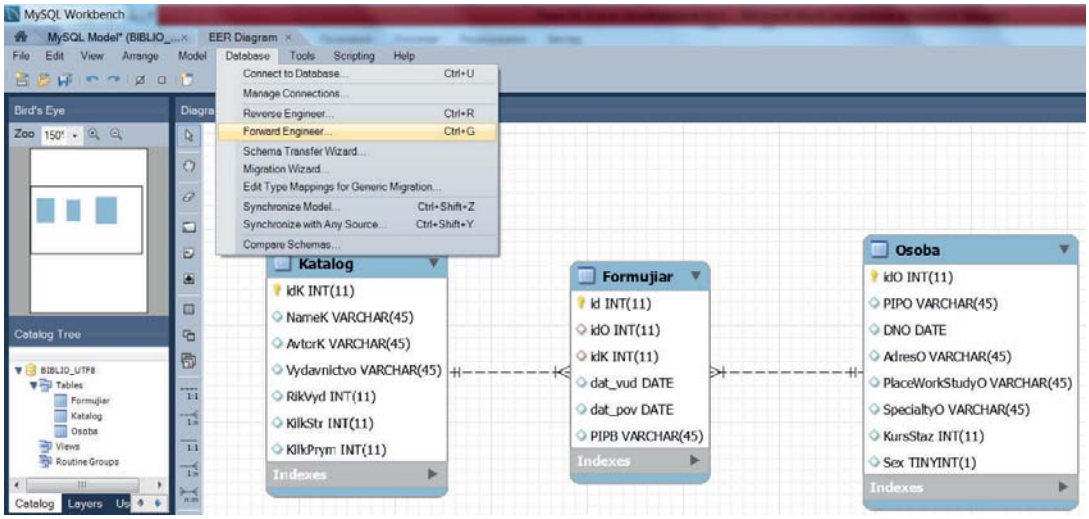


Рис. 4.33. Виклик майстра побудови бази даних

4.3.3.3. Виконання на стороні сервера SQL команд (з моделі даних)

Ще один з варіантів дозволяє створювати таблиці не всі відразу, а почергово. Для даного алгоритму необхідно на кожній з таблиць у вікні EER Diagram за допомогою ПКМ вибрати команду Copy SQL to Clipboard (копіювати в буфер обміну, рис. 4.34), перейти у вікно SQL редактора (Query) Workbench, вставити їх з буфера обміну та запустити на виконання.

Послідовність створення таблиць має значення, тому спочатку створюються батьківські таблиці, а потім – дочірні.

Якщо ж під час створення таблиці у конструкторі для кожного текстового поля не задано collation кодової таблиці, яка підтримує кирилицю, тоді необхідно явно дописати у кінці SQL команди назву кодової таблиці, яка підтримує кирилицю, наприклад, utf8 або cp1251 (рис. 4.35).

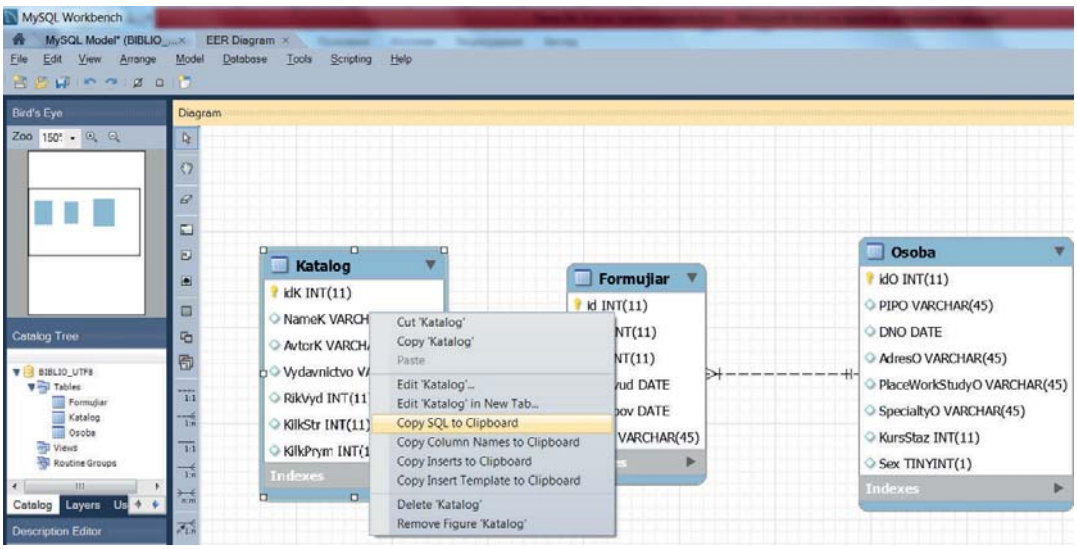


Рис. 4.34. Копіювання SQL коду створення таблиці з моделі даних у буфер обміну

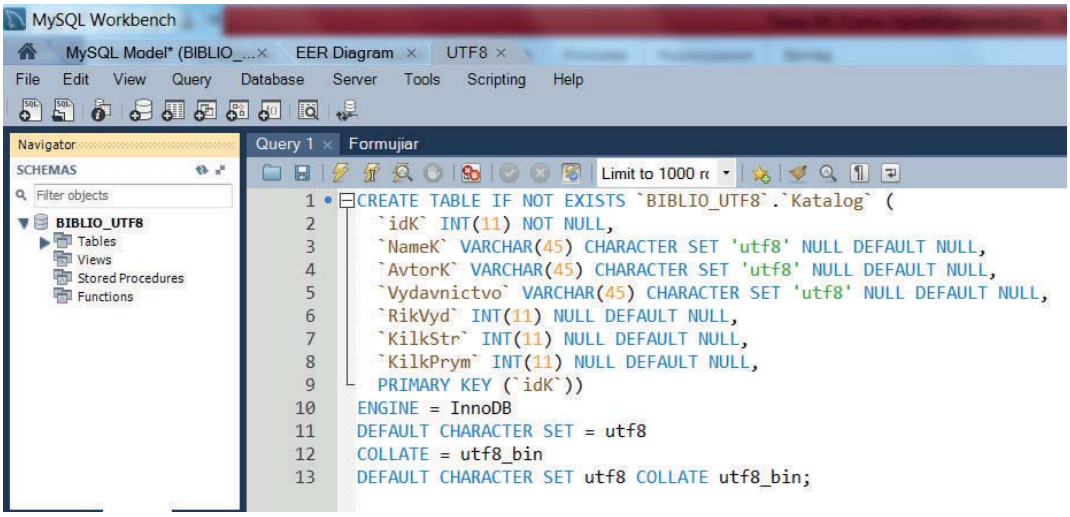


Рис. 4.35. Редагування згенерованого коду створення таблиці

Характерною помилкою при генерації коду створення дочірніх таблиць БД зазвичай є назви зв'язків між таблицями, яка виникає, якщо ключові поля у декількох (або у всіх) таблицях мають однакову назву, наприклад – id. Щоб не модифікувати структури таблиць, в багатьох випадках достатньо у скриптах створення таблиць відредагувати назви зв'язків, наприклад:

…

CONSTRAINT `id1`

…

CONSTRAINT `id2`

…

Але вирішальне значення має етап логічного проектування БД, на якому необхідно детально розробити та нормалізувати схему БД.