Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

Підлягає поверненню на кафедру

Ю. В. Танасюк, Х. С. Одайська

ОРГАНІЗАЦІЯ БАЗ ДАНИХ

Навчальний посібник

Чернівці
Чернівецький національний університет
ім. Ю. Федьковича
2023

УДК 004.65 (075.8)
Т 18

Друкується за рішенням вченої ради
Чернівецького національного університету імені Юрія Федьковича
(протокол № 1 від 30 січня.2023 р.)

Рецензенти: Березький О. М., доктор технічних наук, професор кафедри комп'ютерної інженерії Західноукраїнського національного університету

Цвіркун Л. І., кандидат технічних наук, професор кафедри інформаційних технологій та комп'ютерної інженерії Національного технічного університету «Дніпровська політехніка»

Танасюк Ю. В., Одайська Х. С.

Т18 Організація баз даних : навчальний посібник : Чернівці : Чернівецький національний університет імені Юрія Федьковича, 2023. 208 с.

Розроблене методичне видання має на меті ознайомити студентів з основними поняттями та принципами побудови оптимальних моделей баз даних, вивчення можливостей сучасних систем управління базами даних (СУБД), оволодіння технологіями розроблення баз даних і застосунками для автоматизації роботи з ними, набуття практичних навичок логічного проектування бази даних у рамках реляційного підходу, відпрацювання умінь зі створення БД, забезпечення принципів цілісності інформації та виконання основних операцій оброблення і модифікації даних засобами мови SQL.

Для студентів вищих навчальних закладів, що здобувають освіту за ОПП програмами «Комп'ютерна інженерія» та «Програмування мобільних і вбудованих комп'ютерних систем».

УДК 004.65 (075.8)

ЗМІСТ

ВСТУП

Дисципліна «Організація баз даних» має на меті отримання студентами знань з області проектування та розробки баз даних. Оволодіння такими знаннями дозволить реалізовувати задачі автоматизації оброблення інформації, керування об'єктами, проектування комп'ютерних і обчислювальних програмних систем.

Мета навчальної дисципліни: засвоєння основних понять і принципів побудови оптимальних моделей баз даних, вивчення можливостей сучасних систем управління базами даних (СУБД), опанування класичними і сучасними моделями даних, вивчення технологій розроблення баз даних і застосунків для автоматизації роботи з ними, засвоєння теоретичних і практичних основ логічного проектування БД у рамках реляційного підходу, відпрацювання умінь і навичок створення баз даних, забезпечення принципів цілісності інформації та виконання основних операцій оброблення і модифікації даних у середовищі конкретної СУБД.

Завдання дисципліни: на основі отриманих теоретичних відомостей виробити у студентів уміння працювати в середовищах візуального проектування, як-от ERwin Data Modeler, DB Designer, та системах управління базами даних (MS SQL Server 20x, MySQL та ін.), розвинути навички з розроблення та оптимізації схеми бази даних, її реалізації в обраній СУБД, виконання базових перетворень даних та реалізації клієнт-серверної взаємодії з ядром бази даних за допомогою користувацького інтерфейсу.

У результаті вивчення навчальної дисципліни студенти повинні:

знати: призначення, склад, структуру та функції систем управління базами даних, основні моделі баз даних, принципи та методи проектування реляційних баз даних, основні операції реляційної алгебри, алгоритми маніпулювання даними та способи їх програмної реалізації, базові принципи теорії нормалізації бази даних, мови створення запитів для отримання необхідної інформації, основи мови SQL, засоби створення реляційних баз даних і прикладних програм;

вміти: працювати в середовищах об'єктно-орієнтованого проектування та системах управління базами даних, розробляти структуру бази даних, застосовуючи сучасні методи проектування та керування БД, виконувати розроблення реляційної БД, застосовувати сучасні мови запитів типу SQL, забезпечувати оптимальний доступ до даних, їх прискорений пошук і захищеність, створювати зручний користувацький інтерфейс, застосовуючи засоби сучасних середовищ візуального програмування.

Цей навчальний посібник охоплює низку тем, передбачених силабусом навчальної дисципліни «Організація баз даних». Кожна тема складається з теоретичного матеріалу, лабораторної роботи, практичних рекомендацій щодо виконання, прикладів реалізації з детальним описом, запитань для самоперевірки.

Тема 1. «Сучасні інформаційні системи» розглядає особливості традиційних файлових систем, баз даних та СУБД, а також архітектуру логічної побудови таких систем. Лабораторна робота № 1 допоможе студентам набути навичок зі створення баз даних та виконання у ній основних операцій обробки даних.

Тема 2. «Шаблони подання даних» ознайомлює з найбільш поширеними моделями подання даних та особливостями баз даних SQL та NoSQL. Лабораторна робота № 2 допоможе розпочати проектування схеми бази даних, тут наведено рекомендації щодо розробки ER-діаграм та її реалізації в онлайн-середовищі DB Designer для конкретного прикладу.

Тема 3. «Реляційна модель і нормалізація» присвячена одній з найпоширеніших моделей сучасних баз даних, її аспектам, основним термінам та властивостям. Розглянуто питання підтримки цілісності бази даних та її оптимізації, спираючись на підходи теорії нормалізації. Лабораторна робота № 3 стосується вдосконалення схеми бази даних, створеної на попередньому кроці, моделювання взаємодії у системі, що проектується, та зведення її до більш оптимального вигляду.

Тема 4. «Основи мови SQL. Створення таблиць бази даних» знайомить з історією становлення мови структурованих запитів, типами даних, особливостями оператора створення таблиць, а також типами даних та обмеженнями, які

застосовуються при визначенні властивостей полів таблиці. Лабораторна робота № 4 описує детальну процедуру генерування SQL-коду зі створення таблиць бази даних на основі розробленої діаграми «сутність–зв'язок», а також фізичну реалізацію цих операторів у СУБД MS SQL Server.

Тема 5. «Базові оператори мови SQL» містить детальний огляд операцій реляційної алгебри та приклади їх реалізації мовою SQL. Значну увагу приділено різноманітним варіантам операторів вибірки, з'єднання та групування даних таблиць. Лабораторна робота № 5 надасть можливість реалізувати розглянуті оператори на практиці та проаналізувати одержані результати.

Тема 6. «Розширені можливості мови SQL» надає інформацію про особливості створення вкладених запитів, збережених процедур та тригерів. Ці та інші об'єкти бази даних студенти матимуть змогу використати у лабораторній роботі № 6 для реалізації запитів за варіантом індивідуального завдання.

На лабораторних роботах кожен студент працює над індивідуальним варіантом завдання, яке розподіляє викладач.

Дисципліна «Організація баз даних» складається з двох модулів. Оцінювання навчальної діяльності студентів ґрунтується на результатах виконання та захисту лабораторних робіт, тестування по темах, а також написання проміжних і підсумкових контрольних робіт.

ТЕМА 1. Сучасні інформаційні системи

За всю історію обчислювальної техніки можна простежити дві основні сфери її застосування. Перша сфера – це застосування обчислювальної техніки для виконання математичних розрахунків, які надто довго або взагалі неможливо провести вручну. Розвиток саме цієї галузі сприяв інтенсифікації методів числового розв'язку складних математичних задач, розвитку класів мов програмування, орієнтованих на зручний запис числових алгоритмів, становлення зворотного зв'язку з розробниками нових архітектур ЕОМ.

Друга галузь, яка безпосередньо належить до теми наших занять, – це використання засобів обчислювальної техніки в автоматичних або автоматизованих інформаційних системах. У найширшому розумінні, **інформаційна система** *являє собою програмно-апаратний комплекс, функції якого полягають у надійному збереженні інформації у пам'яті комп'ютера, виконання специфічних для цього додатка перетворень інформації та/або обчислень, наданні користувачеві зручного та зрозумілого інтерфейсу. Зазвичай такі системи мають справу з великими обсягами інформації, яка має досить складну структуру.*

Класичними прикладами інформаційних систем є банківські системи, системи резервування авіаційних квитків, місць у готелі тощо.

Традиційні файлові системи

Традиційні файлові системи були першою спробою перевести у цифровий формат дані, що зберігалися на паперових носіях, і комп'ютеризувати їх обробку. Така документація могла містити всю зовнішню і внутрішню інформацію, пов'язану з деяким продуктом, завданнями, клієнтами або співробітниками організації, фінансовими транзакціями, операціями обліку та діяльністю підприємства загалом. Зазвичай такі документи сортуються і впорядковуються по папках або полицях у шафі. Паперові носії дають змогу успішно справитися з завданнями пошуку, якщо кількість об′єктів, що зберігаються, невелике. Вони цілком придатні для роботи з великою кількістю об'єктів,

які потрібно лише зберігати або вилучати. Проте встановлення перехресних запитів або виконання комплексної обробки відомостей у документації вимагає значних витрат часу та зусиль.

Традиційні файлові системи (ТФС) були розроблені для забезпечення більш ефективного доступу до даних, перетворених на електронний формат, оскільки працівники окремого підрозділу або відділу працювали з визначеними даними за посадовими функціями. Проте при перенесенні інформації на цифрові носії, замість організації централізованого сховища, дані у вигляді файлів були розподілені по окремих підрозділах за напрямами діяльності, де їх опрацьовували за допомогою спеціально створених програмістами прикладних програм [1].

Файл являє собою простий набір **записів**, які містять логічно пов'язані дані, структуровані у вигляді таблиці. Кожен запис складається з одного або декількох полів, кожне з яких подає деяку характеристику модельованого об'єкта. Набір прикладних програм кожного відділу дозволяв вводити дані, працювати з файлами і генерувати деякий фіксований набір спеціалізованих звітів. Найважливіше те, що фізична структура і методи зберігання записів файлів з даними жорстко визначалися у коді прикладної програми.

Інакше кажучи, **файлові системи** *звичайно забезпечують збереження слабо структурованої інформації, залишаючи подальшу структуризацію та обробку прикладним програмам.*

Завдяки різноманітності завдань у процесі обробки даних у межах кожного підрозділу сформовано необхідні бібліотеки програм над файловими системами, подібно тому, як це робиться у компіляторах. Оскільки в інформаційних системах необхідно підтримувати складні структури даних, ці індивідуальні засоби керування даними складали суттєву частину інформаційних систем, практично повторюючись від однієї системи до іншої.

Як обов'язковий компонент ТФС прикладні програми виконували такі функції:

- формування структури файлів;
- визначення методів запису у файл;
- реалізації запитів;
- формування звітів;
- створення користувацького інтерфейсу.

Фізична структура файлів із даними, способи їх зберігання та виконувані операції визначалися виключно в коді прикладних програм. У зв'язку з цим традиційним файловим системам вдалося на належному рівні забезпечити структурування даних та автоматизувати їх оброблення. Проте в ході використання цих систем було виявлено певні недоліки, зокрема:

- відокремлення та ізоляція даних;
- неконтрольоване дублювання даних;
- залежність від даних;
- несумісність форматів файлів;
- фіксований набір виконуваних запитів;
- швидке збільшення кількості застосунків.

Системи з базами даних

Усі наведені вище обмеження традиційних файлових систем є наслідками двох факторів:

1. Структура і вміст файлів визначалися всередині прикладних програм, а не зберігалися окремо і незалежно.
2. Окрім прикладних програм, не було передбачено жодних інших інструментів доступу до даних та їх обробки.

Для підвищення ефективності роботи з даними було впроваджено інший, централізований, підхід, заснований на базі даних та уніфікованому програмному забезпеченні для взаємодії з нею.

База даних – *це сумісно використовуваний набір логічно зв'язаних даних (та опис цих даних), передбачений для задоволення інформаційних потреб організації.* Це єдине велике сховище даних, яке одноразово визначається, а потім використовується одночасно багатьма користувачами. Замість відокремлених файлів із дублюванням інформації, тут всі дані зібрані разом із мінімальною часткою надлишковості. База даних уже не належить якомусь окремому підрозділу або відділу, а виступає корпоративним ресурсом. Причому база даних не лише містить дані цієї організації, але й їх опис. Тому базу даних деколи називають набором інтегрованих записів із самоописом. Загалому, опис даних називають системним каталогом або словником даних, а самі елементи опису – **метаданими**, тобто

даними про дані. Саме наявність самоопису даних у базі даних забезпечує незалежність між програмою та даними [1].

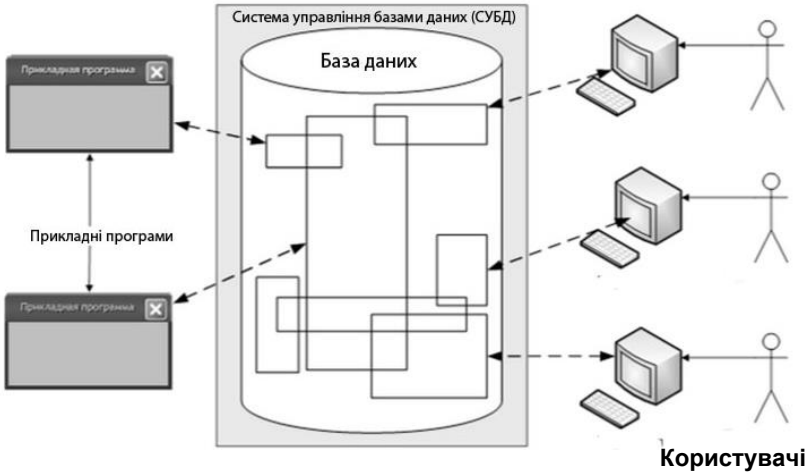


Рис. 1.1. Схема взаємодії користувачів із базою даних за посередництвом СУБД

Система управління базами даних (СУБД) – це універсальне програмне забезпечення, за допомогою якого користувачі можуть створювати, визначати структуру та підтримувати базу даних, здійснювати до неї контрольований доступ і виконувати обробку інформації.

СУБД – це програмне забезпечення, яке взаємодіє з прикладними програмами користувачів та базою даних і має такі можливості:

♦ Створення бази даних – зазвичай за допомогою мови визначення даних (*Data Definition Language, DDL*). Мова DDL надає користувачеві засоби для формування структури та складу таблиць, зазначення типів даних та створення обмежень для інформації, яка зберігається у базі.

♦ Додавання, оновлення та видалення інформаціъ з БД, що зазвичай здійснюється за допомогою мови маніпулювання даними (*Data Manipulation Language, DML*). Наявність централізованого сховища усіх даних та їх опис дозволяє використовувати єдину мову як загальний інструмент організації

запитів. Найбільш поширена мова як DDL так і DML – це мова структурованих запитів (*Structured Query Language*, *SQL*, вимовляється «ес-к'ю-ель»). Мова SQL наразі фактично обов'язкова для реляційних СУБД.

♦ Контрольований доступ до бази даних за допомогою засобів захисту, підтримки цілісності, паралельної обробки, багатокористувацького доступу, резервного копіювання тощо.

Трирівнева архітектура ANSI SPARC

З метою уніфікувати процес розробки, впровадження та використання систем управління базами даних комітет зі стандартів планування та вимог SPARC (англ. *Standards Planning and Requirements Committee*, американського національний інститут стандартів ANSI у 1975 р. розробив архітектуру бази даних, що описує логічну організацію системи з точки зору подання даних користувачам, загальної структури даних та їх фізичного зберігання. Архітектура ANSI SPARC складається з трьох основних рівнів розробки/існування бази даних (рис. 1.2):

- зовнішній (користувацько-логічний)
- концептуальний (логічний)
- внутрішній (фізичний).

Зовнішній рівень пов'язаний з *індивідуальними* поданням даних для окремих користувачів через інтерфейс прикладної програми або СУБД. Користувачі можуть належати до різних груп, як-от прикладні програмісти, працівники різних відділів, клієнти, адміністратори тощо. У кожного типу користувачів є своя мова взаємодії з СУБД. Зокрема, для простих клієнтів нею може бути мова запитів або спеціальна мова, заснований на формах і меню. Для прикладних програмістів – це одна з мов високого рівня, яку вони застосовують для створення інтерфейсу програми. Всі ці мови містять засоби підтримки операторів базових об'єктів і операцій БД.

Простих співробітників організації або клієнтів, що користуються послугами інформаційної системи, зазвичай цікавить лише деяка частина всієї БД. Окрім того, вигляд, у якому дані подаються користувачам, може суттєво відрізнятися від того, як вони зберігаються.

Концептуальний рівень пов'язаний з *узагальненим* поданням усіх даних, з якими мають справу користувачі. Інакше кажучи, різні користувачі можуть по-різному бачити дані, та працюють лише з обмеженою частиною бази. Тому завдання концептуального рівня полягає у поданні єдиної логічної моделі всієї бази даних.

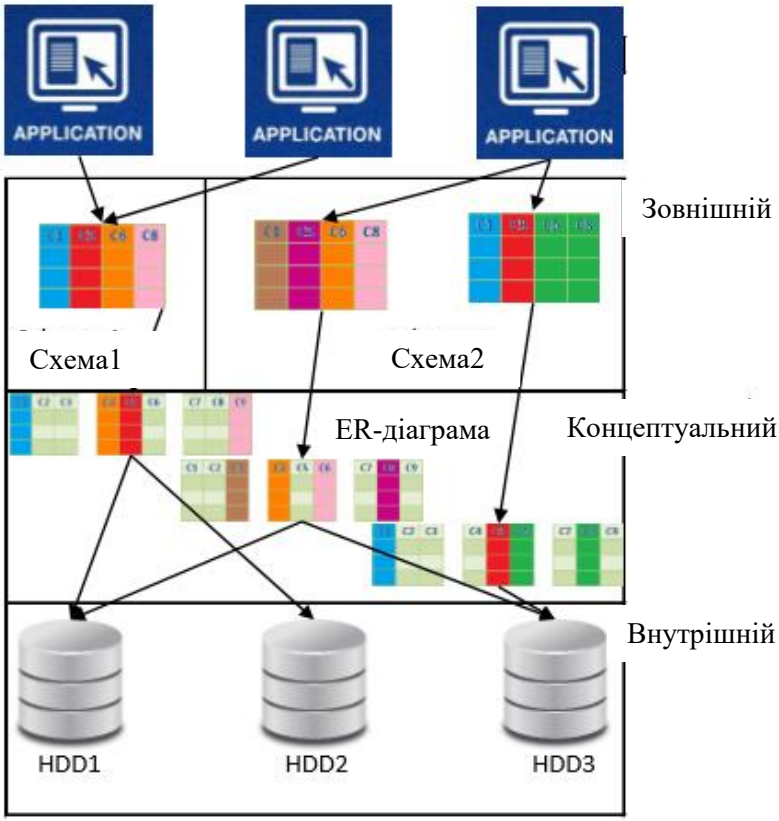


Рис. 1.2. Схематичне зображення трирівневої архітектури бази даних ANSI SPARC

Графічно концептуальна модель БД подається у вигляді ER-діаграми або діаграми «сутність–зв'язок». Така діаграма повинна розроблятися перед фізичною реалізацією бази даних і враховувати потреби усіх користувачів системи. Концептуальна

модель не залежить від того, у якій СУБД буде створено базу даних, і приховує складність її фізичного зберігання.

Внутрішній рівень найбільш наближений до фізичного сховища інформації, тобто пов'язаний зі способами зберігання даних на фізичних носіях. На цьому рівні описано, як дані з бази розміщуються на диску, якого вигляду набувають записи, в який спосіб вони індексуються для полегшення пошуку, тощо. Формат даних, іменування та розширення файлів, способи ущільнення та, за потреби, шифрування інформації визначаються суто в СУБД.

В архітектурі ANSI SPARC особливу увагу приділено ізоляції програмних застосунків від особливостей узагальненого подання даних у базі на низькому рівні, при збереженні зв'язку між ними за допомогою концептуального рівня. Нижче наведено причини такого поділу:

- Одні й ті самі дані можуть подаватися різним користувачам у різний спосіб.
- Кожен користувач повинен мати можливість змінити своє подання даних без впливу на інших користувачів.
- Взаємодія користувача з БД не повинна залежати від способів зберігання даних, типу їх індексування або хешування.
- Адміністратор БД повинен мати змогу змінювати структуру фізичного зберігання даних і не впливати на те, як бачать дані окремі користувачі.

Слід зазначити, що нижче внутрішнього рівня перебуває фізичний рівень зберігання даних на рівні файлів, що використовує операційну систему та контролюється СУБД.

Загальний вигляд або опис бази даних називають *схемою бази даних*. Відповідно до кожного рівня ANSI SPARC існує окрема схема БД. Концептуальна і внутрішня схеми для кожної бази даних визначаються одноразово. Такий взаємозв'язок дає змогу визначити місце зберігання елементів даних (записів у файлах). Через поєднання зовнішньої схеми з концептуальною вдається пов'язати подання даних користувача з його логічним виглядом на концептуальному рівні.

Архітектура СУБД

Більшість підходів до організації інформаційних систем на основі баз даних ґрунтуються на архітектурі «клієнт–сервер», яка існує у двох модифікаціях, в залежності від функцій, що покладаються на кожну зі сторін [2].

Однією з ранніх версій сучасних СУБД була **файл-серверна система**, яка передбачала наявність у мережі сервера для централізованого зберігання файлів БД, доступ до яких здійснювався через автономне ПЗ з боку клієнтів (рис. 1.3). Відповідно до запитів файли з файл–сервера надходили на робочі станції користувачів, де здійснювалася основна частина обробки даних.

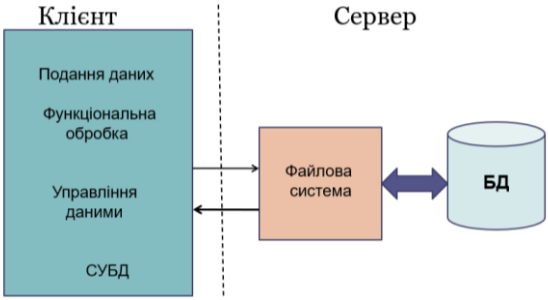


Рис. 1.3. Архітектура «файл–сервер»

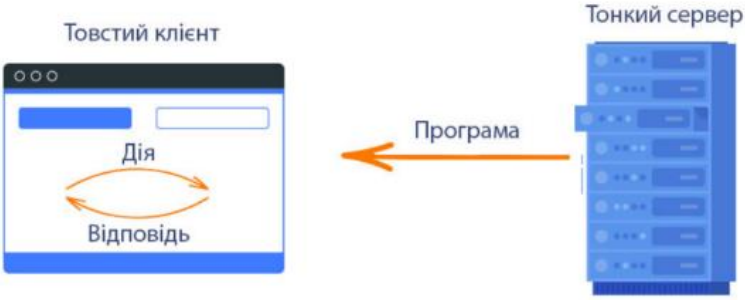


Рис. 1.4. «Товстий» клієнт та «тонкий» сервер у файл-серверній архітектурі [2]

Оскільки у такій системі сервер переважно виконував роль сховища файлів, не беручи участі в обробці самих даних, від нього вимагався лише достатній обсяг дискової пам'яті. Водночас на кожному клієнтському комп'ютері встановлена прикладна програма реалізовувала прикладний інтерфейс з визначеним набором операцій та засобами керування БД. Після завершення роботи користувачі надсилали назад на сервер копії власних файлів з обробленими даними, звідки вони були доступні іншим користувачам. Як бачимо, файл-серверна архітектура реалізована як «товстий» клієнт і дуже «тонкий» сервер (рис. 1.4).

Переваги:

- простота реалізації
- зручні засоби розробки інтерфейсу під кожного користувача
- робота в оффлайн-режимі
- організація багатокористувацького доступу.

Недоліки:

- генерування в мережі великих обсягів трафіку
- суттєве навантаження на клієнтських машинах
- складність обслуговування клієнтського ПЗ
- сповільнена обробка та оновлення даних на сервері
- низький рівень захищеності даних
- неконтрольована робота клієнтів.

У **клієнт-серверній архітектурі**, крім зберігання файлів БД, сервер повинен виконувати основну частину обробки даних завдяки розміщеній на ньому СУБД (рис. 1.5). Звернення користувачів щодо виконання деяких дій над даними відбувається з клієнтською частиною СУБД через ПЗ, яке автоматично генерує запити мовою SQL, що відправляються на сервер. Результати запитів можуть накопичуватися на сервері та передаватися клієнтові після повного їх виконання. На практиці поширена ситуація, коли для роботи клієнтам потрібна лише невелика частина всієї БД, і тому в програмі на боці клієнта може підтримуватися локальний кеш.

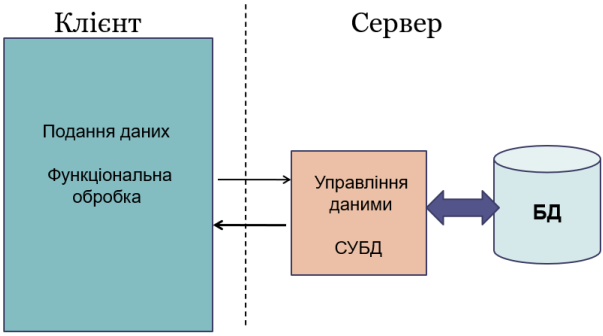


Рис. 1.5. Архітектура «клієнт–сервер»

Оскільки вся робота з БД (вибірка, додавання, видалення, тощо) відбувається з боку сервера, в клієнт-серверній організації клієнти є «тонкими», а сервер повинен бути «товстим», тобто настільки потужними, щоб задовольняти потреби всіх клієнтів (рис. 1.6).

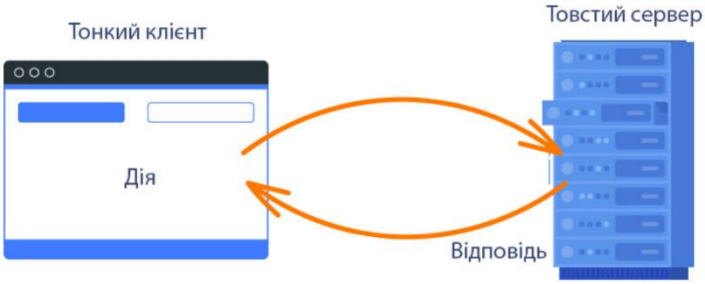


Рис. 1.6. «Тонкий» клієнт та «товстий» сервер у клієнт-серверній архітектурі [2]

Переваги:

- можливість обслуговування запитів від декількох клієнтів
- зменшення навантаження на мережу та машини з обох боків

- захист і контроль даних здійснюється засобами СУБД, що дозволяє блокувати небажані дії
- підвищена швидкість обробки даних з боку сервера
- масштабованість і здатність до розширення.

Недоліки:

- потреба створення окремого прикладного застосунку для забезпечення інтерфейсу (дублювання коду та запитів)
- правила функціональної обробки подекуди можуть бути суперечливими
- підвищені вимоги до апаратних потужностей та програмного забезпечення сервера, а також до засобів керування БД.

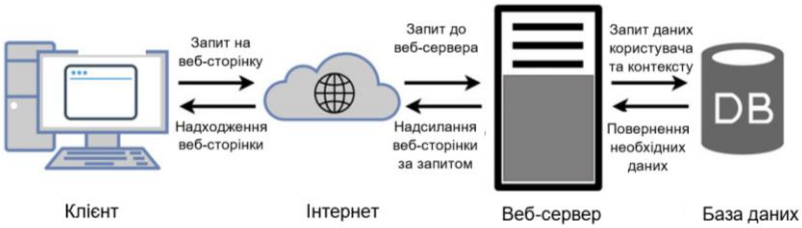


Рис. 1.7. Архітектура «клієнт–сервер» у веб-застосунках

Завдяки поширенню інтернет-технологій та веб-програмування модель «тонкого» клієнта набула можливість уніфікації користувацьких інтерфейсів і програм. По-перше, за допомогою мови гіпертекстової розмітки документів HTML відносно легко розробляти зручну у використанні інформаційну структуру, яку можна розмістити для доступу та обслуговування на одному з готових Web-серверів. По-друге, наявність готових до використання клієнтських частин – браузерів – усуває потребу в розробці власних користувацьких інтерфейсів, надаючи зручні досконалі механізми доступу до інформації з будь-якого місця та пристрою (рис. 1.7).

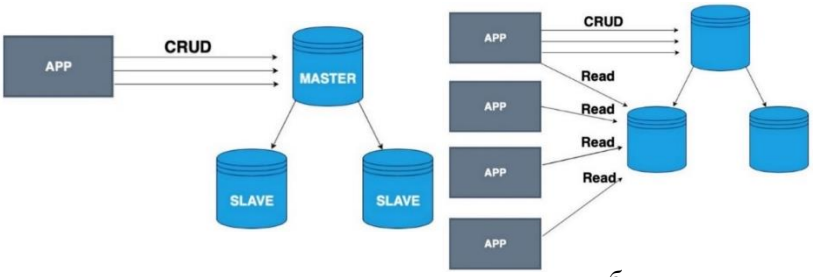
Клієнтська частина «тонкий клієнт», що взаємодіє з користувачем, є HTML-сторінкою у веб-браузері або Windows-застосунком, взаємодіючим з веб-сервісами. Вся програмна

логіка винесена на веб-сервер, який забезпечує формування запитів, що передаються на виконання серверу баз даних.

Масштабування бази даних

Яким би потужним не був сервер бази даних, рано чи пізно він може зіткнутися з проблемою перевантаження та неспроможністю належно обробляти користувацькі запити. До того ж зберігати базу даних в одному екземплярі небезпечно, адже в разі її відмови чи зламування можна втратити важливі дані. Для запобігання цим ризикам слід застосовувати масштабування БД, яке передбачає поділ даних на групи та розміщення їх на окремих резервних серверах. Такий підхід широко застосовується для баз даних SQL та NoSQL. Розглянемо дві техніки масштабування – реплікацію та шардинг.

Реплікація – це повне копіювання бази даних на один або декілька додаткових серверів. Розглянемо роботу цього методу з огляду на ефективність виконання базових операцій з базою даних – *CRUD* (***C****reate,* ***R****ead,* ***U****pdate,* ***D****elete* – створення, читання, оновлення, видалення). При застосуванні реплікації виділяють два типи серверів: *Master* та *Slave* (рис. 1.8). Master використовується для запису або зміни інформації, Slave – для копіювання інформації з Master та її читання. Найчастіше використовується один Master і кілька Slave, оскільки зазвичай запитів на читання більше, ніж запитів на зміну.



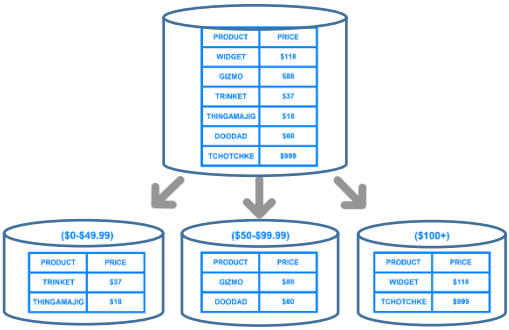
а *б*

Рис. 1.8. Застосування реплікації за принципом «Master–Slave» (*а*) та розподілення виконання базових операції оброблення даних (*б*)

Головна перевага реплікації – велика кількість копій даних. Завдяки цьому при виході з ладу головного сервера будь-який інший зможе його замінити. Такий підхід добре підходить для обслуговування користувачів у різних локаціях. Сервери можна розподілити так, щоб задовольнити потреби якомога більшої кількості користувачів та максимально швидко реагувати на їх запити. На кожному з цих серверів будуть зберігатися копії одних і тих самих даних, а клієнтські запити оброблятимуться швидше.

Однак як механізм масштабування реплікація не надто зручна. Причина цього – розсинхронізація та затримки під час передавання даних між серверами. Найчастіше реплікація використовується як засіб для забезпечення стійкості до відмови разом з іншими методами масштабування.

Шардинг (*sharding*), або фрагментування – це принцип проектування бази даних, при якому частини таблиці зберігаються окремо на різних серверах. Шардинг є найбільш прийнятним рішенням хмарних обчислень та обробки великих обсягів даних, особливо якщо його використовувати в парі з реплікацією, але це досить складно організувати, оскільки необхідно враховувати міжсерверну взаємодію.



.

Рис. 1.9. Шардинг таблиць бази даних

Вертикальний шардинг – це розміщення таблиці або логічно пов’язаних даних на окремому сервері (рис. 1.10, *а*). Цей принцип передбачає вертикальне масштабування – нарощування або зменшення обчислювальної потужності або ресурсів для зберігання БД у міру необхідності.

Горизонтальний шардинг – це розподіл частин однієї таблиці БД між декількоми менш потужними серверами (рис. 1.10, *б*). Зазвичай використовується для розміщення великих таблиць, які не вміщаються на одному сервері. Горизонтальне масштабування замість нарощування потужностей однієї машини передбачає залучення додаткових баз даних або розподіл великих таблиць по різних вузлах.

а *б*

Рис. 1.10. Масштабування: *а* – вертикальне; *б* – горизонтальне

Розподілені системи мають численні переваги:

- ✓ Більша надійність та доступність. Завдяки розміщенню декількох екземплярів даних на різних вузлах, у разі відмови локального сховища, дані можна отримати з іншого доступного сервера.
- ✓ Підвищена продуктивність, особливо при обробленні великих обсягів даних. Потужність і пропускну здатність системи можна нарощувати шляхом додавання до кластера нових серверів.
- ✓ Безперервність роботи без надмірної залежності від центральної точки.

Найбільші виклики розподіленої архітектури – це узгодженість та сумісне оброблення фрагментів даних, які зберігаються на декількох вузлах, між якими потрібно синхронізувати зміну даних в одній точці.

ЛАБОРАТОРНА РОБОТА № 1
Бази даних: створення та керування

Мета

Навчитися працювати в СУБД **MS SQL Server** та набути практичних навичок зі створення бази даних та застосування до неї основних операторів мови SQL.

Завдання

Вивчити можливості середовища **Microsoft SQL Server Management Studio**, створити базу даних, що складається з однієї таблиці; визначити структуру таблиці, заповнити її тестовими даними та навчитися виконувати операції вибірки, пошуку, оновлення, додавання та видалення даних у таблиці.

Теоретичні відомості

У сучасних реляційних середовищах база даних складається з набору взаємопов'язаних таблиць, створених відповідно до логічної схеми БД для тривалого зберігання даних та їх обробки. Схема БД розробляється у процесі проектування і досить рідко модифікується. Водночас вміст БД може змінюватися, доповнюватися або коригуватися. Завдяки гнучкості мови SQL різні типи користувачів можуть створювати довільні запити, з метою отримання інформації з різних таблиць бази даних.

До найбільш поширених операторів мови SQL при роботі з таблицями у БД належать:

- **`Insert`** – внесення нового запису до наявної таблиці;
- **`Select`** – вибірка даних зі створених таблиць;
- **`Update`** – оновлення даних у існуючих таблицях;
- **`Delete`** – видалення записів зі створених таблиць.

Використання цих чотирьох операторів розглянемо на прикладі бази даних що складається з однієї таблиці під назвою **Movies** (Кінофільми), структура та вміст якої подані у табл. 1.1.

Таблиця 1.1

Таблиця **MOVIES**

MID	Title	Studio	Runtime	Release_year	Rating	Budget
1	Недоторканні	Sony Pictures	112	2011	9,2	10
2	Маленькі жінки	Sony Pictures	135	2019	7,8	40
3	Кримінальне чтиво	Miramax Films	154	1994	8,4	8
4	Втеча з Шоушенка	Warner Brothers	142	1994	9,2	25
5	Список Шиндлера	Universal Studios	195	1993	9,4	22
6	Форест Гамп	Paramount Pictures	182	1994	9,2	55
7	Залізна людина	Paramount Pictures	126	2008	8,2	140
8	Амелі	Miramax Films	112	2000	8,2	10
9	До зустрічі з тобою	Warner Brothers	110	2016	8,6	20
10	Інтерстеллар	Warner Brothers	169	2014	8,0	165
11	Посіпаки	Universal Studios	91	2015	6,2	74
12	Щасливого Різдва	Universal Studios	103	2019	6,7	95

Розглянемо детальніше склад таблиці:

- Рядки таблиці називають записами, а стовпці – полями.
- Назви полів та самої таблиці **не залежать від регістру**.
- Поле MID – це унікальний ідентифікатор таблиці Movies, що для кожного рядка повинен набувати унікального значення. Таке поле називають **первинним ключем**.
- Title і Studio відповідають назві кінофільму та кіностудії, на якій він був виготовлений. Вважаємо, що у цих полях зберігатимуться набори символів, тобто рядкові значення. При визначенні умови в операторах Select, Update, Delete або при додаванні нового запису через Insert, значення символьних типів **беруться в одинарні лапки**.
- Runtime (тривалість) та Release_year (рік випуску) – поля цілого типу.
- Rating – рейтинг перегляду, поле дійсного типу.
- Budget (бюджет фільму) – містить суму в млн USD.

Розглянемо приклад виконання **операції вибірки**, що повертає інформацію про назву, номер та кіностудію для фільмів, випущених у 1994 році:

```
SELECT Title, MID, Studio
FROM Movies
WHERE Release_year=1994
```

Після зарезервованого слова **`Select`** через кому зазначаються назви полів, з яких потрібно одержати дані. Порядок полів можна змінювати. Цільову таблицю вказано після слова **`From`**. Використання після **`Select`** символу «*» дає змогу вивести інформацію з усіх полів таблиці. Результат вибірки можна обмежити за допомогою однієї або декількох умов (>, <, <=, >=, !=), позначених директивою **`Where`**.

Результат виконання вибірки завжди подається у вигляді таблиці:

Title	**MID**	**Studio**
Кримінальне чтиво	3	Miramax Films
Втеча з Шоушенка	4	Warner Brothers
Форест Гамп	6	Paramount Pictures

ЗАУВАЖЕННЯ: при створенні запитів команди мови SQL, так само як назви полів і таблиць, не залежать від регістру.

Нижче наведені приклади операцій додавання нового запису (**Insert**), оновлення (**Update**) та видалення (**Delete**) для бази даних кінофільмів.

Додавання нового запису до таблиці:

```
INSERT
INTO Movies (Mid, Title, Studio, Runtime, Release_year, Rating, Budget)
VALUES (23, 'Люди Ікс', '20th Century Fox', 104, 2000, 7.8, 75)
```

Зверніть увагу, що символьні назви взято в одинарні лапки, послідовність значень узгоджується з порядком розташування полів у списку, а в дійсному значенні Rating для відокремлення цілої та дробової частин використовується кома.

Оновлення збережених даних:

```
UPDATE Movies
SET Rating=8,2
WHERE MID=10
```

Для фільму з номером 10 поточне значення рейтингу замінюється на 8,2. Якщо не вказати умову, зміни торкнуться всіх фільмів.

Видалення наявного запису:

```
DELETE
FROM Movies
WHERE MID=3
```

З таблиці Movies повністю видаляється запис під номером 3.

Результат виконання операторів **Update**, **Delete** та **Insert** можна переглянити при оновленні таблиці.

ПРАКТИЧНЕ ЗАВДАННЯ

1. Під'єднання до MS SQL Server

Відкрийте середовище **Microsoft SQL Server Management Studio** з групи програм **Microsoft SQL Server Tools** (рис. 1.11). Не змінюйте параметри, визначені за замовчуванням. Вигляд вікна з'єднання може дещо відрізнятися від наведеного на рис. 1.11. Натисніть кнопку **Connect**.

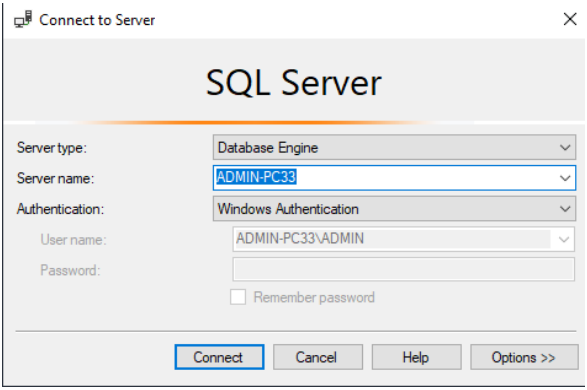


Рис. 1.11. Налаштоване з'єднання із сервером бази даних

2. Створення нової бази даних

2.1. У вікні **Object Explorer** (Провідник об'єктів) ліворуч натисніть правою кнопкою миші на папці **Databases** (Бази даних) і у контекстному меню оберіть опцію **New Database** (Нова база даних), як подано на рис. 1.12.

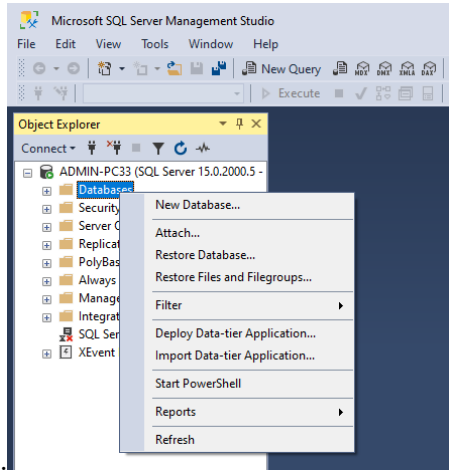


Рис. 1.12. Створення на сервері нової бази даних

2.2. Введіть параметри нової БД, зокрема її назву – **Movies** (рис. 1.13). У цьому ж вікні можна переглянути файли, з яких складає база даних, а також їх місцезнаходження на диску

(параметр **Path**). Натисніть **ОК.** У разі успішного створення нова база даних відображатиметься у Провіднику об’єктів.

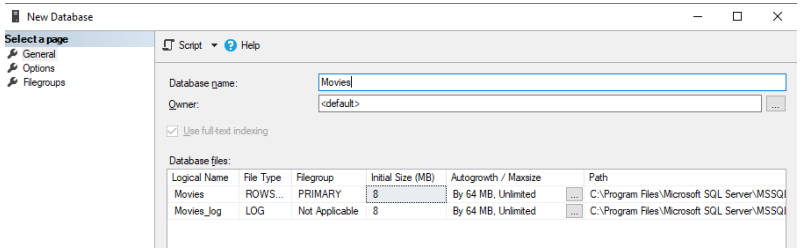


Рис. 1.13. Визначення параметрів нової бази даних

3. Створення таблиці в базі даних

3.1. Виділіть щойно створену БД і розгорніть її вміст, натиснувши на «+» ліворуч від її назви. З’являється цілий перелік елементів, що входять до складу БД, серед яких є таблиці (**Tables**).

3.2. Клацніть правою кнопкою миші на елементі **Tables** і у контекстному меню виберіть команду **New > Table**. Відкриється вікно для визначення полів нової таблиці та їх властивостей.

3.3. Визначте схему таблиці, тобто перелік полів, що відповідає таблиці **Movies**. Для цього в колонці **Colomn Name** (Назва стовпця) зазначте назву поля, згідно з табл. 1.1. У стовпчику **Data Type** (Тип даних) оберіть для кожного поля тип, відповідно до прикладу на рис. 1.14.

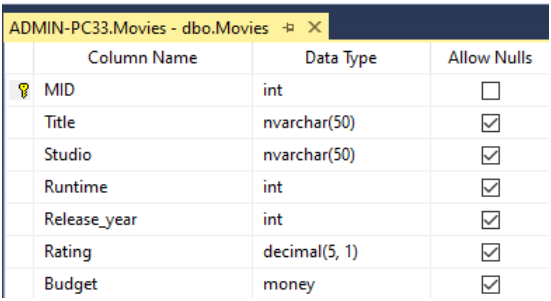


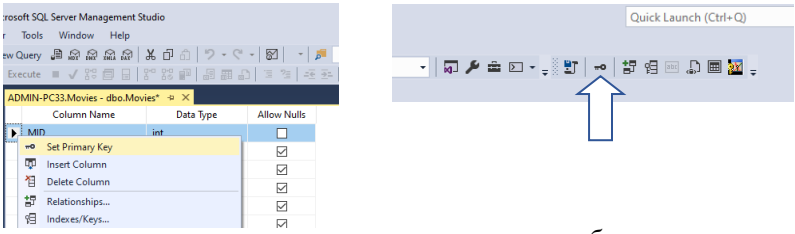
Рис. 1.14. Схема таблиці Movies

3.4. Для полів MID, Runtime, Release_year встановлено цілочисловий тип **`int`**, для Title та Studio – символьний тип

змінної довжини **nvarchar**, що підтримує введення до 50 символів тексту, зокрема кирилицею. Довжину поля зазначено параметром **Length** у вікні **Column Properties** (Властивості поля).

Поле Rating позначене дійсним типом фіксованої довжини **decimal(5,1)**, де 5 – це загальна кількість знаків у числі, а 1 – кількість знаків після коми. Ці значення можна змінити у **Column Properties** за допомогою відповідних параметрів **Precision** та **Scale**.

3.5. Серед полів таблиці обов'язково повинен бути **первинний ключ** – спеціальний ідентифікатор, що унікально позначає записи в таблиці та не може містити однакових значень. Зокрема, це поле MID. Клацніть правою кнопкою миші на цьому полі й у контекстному меню оберіть опцію **Set Primary Key** (Встановити первинний ключ) (рис. 1.7, *а*). Або натисніть на піктограму із зображенням ключа на панелі інструментів головного вікна (рис. 1.15, *б*). Після цього біля обраного поля з'явиться зображення ключа.



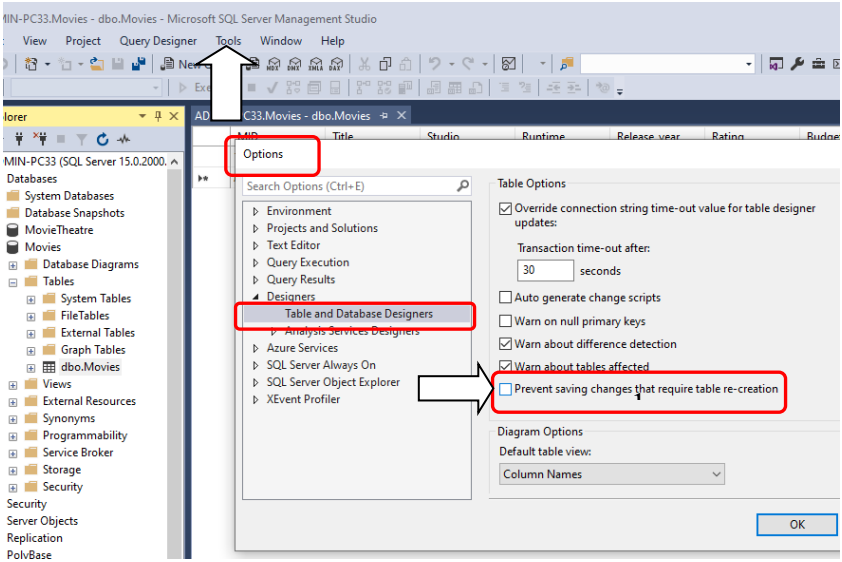
а *б*

Рис. 1.15. Визначення поля первинного ключа: *а* – у контекстному меню; *б* – на панелі інструментів головного вікна

3.6. Виконані налаштування можна зберегти при закриті вікна створення таблиці. Саме тоді буде запропоновано зберегти таблицю. Обирайте «Так», після чого у вікні, що з'явилося, введіть назву таблиці – **Movies**.

3.7. За замовчуванням немає дозволу на зміну параметрів полів таблиці після її створення. Аби усунути це обмеження для обраної таблиці, перейдіть до пункту меню **Tools** (Інструменти) > **Options** (Параметри) > **Designers** (Конструктори) > **Table**

and Database Designers (Конструктори таблиць і бази даних) і зніміть прапорець з опції **Prevent saving changes that require table re-creation** (Заборонити збереження змін, що вимагають повторного створення таблиці), як зображено на рис. 1.16.



1.16. Зняття обмежень на зміни структури таблиці

3.8. Для перегляду та редагування сформованої структури таблиці, клацніть на ній у Провіднику об'єктів правою кнопкою миші та оберіть опцію **Design**.

4. Заповнення таблиці

4.1. Аби внести дані до таблиці, клацніть на ній правою кнопкою миші та у контекстному меню оберіть пункт **Edit Top 200 Rows** (Редагувати перші 200 рядків). З'являється порожня таблиця з переліком полів, визначеним на попередньому кроці.

4.2. Заповніть таблицю даними з табл. 1.1. При закритті вікна дані зберігатимуться автоматично.

5. Створення запитів

5.1. Після внесення даних увійдіть до **Object Explorer** таблицю **dbo.Movies**, натисніть на ній правою кнопкою миші та у контекстному меню оберіть пункт **Select Top 1000 Rows**

(Вибрати перші 1000 рядків). При цьому разом із заповненою таблицею відкриється вікно для створення запитів із заданим шаблоном оператора **Select**. Наведену за замовчуванням команду можна змінювати і додавати нові оператори. Для виконання запитів у цій лабораторній роботі назви полів не обов'язково вказувати у квадратних дужках, ім'я таблиці не потрібно супроводжувати назвою бази даних. Форматування власне запиту (відступи, пробіли, переходи на новий рядок) також не впливатиме на результат його виконання, головне – дотримуватися правильної послідовності команд в операторі.

5.2. Розгляньте наведені нижче запити, створіть і запустіть їх на виконання. Для перегляду результату виконання запиту натисніть на піктограму 

Execute на панелі інструментів або натисніть **F5**.

Поясність одержані результати. Збережіть їх екранні форми.

а)
```
Select Title, Studio
From Movies
Where MID=7;
```

б)
```
Select Title, Budget
From Movies
Where Rating > 9,0;
```

в)
```
Select MID, Title, Runtime
From Movies
Where Release_year<2001;
```

г)
```
Select Title, Rating, Release_year
From Movies
Where Studio='Warner Brothers'
AND Rating>8,5;
```

5.3. Створіть запити на додавання, оновлення та видалення, наведені нижче. Збережіть результати їх виконання.

Переглянути результат виконання операторів **Update**, **Delete** та **Insert** можна, оновивши таблицю.

а)
```
Insert
into Movies (Mid, Title, Studio, Runtime, Release_year, Rating, Budget)
Values (14, 'Зелена миля', 'Warner Brothers', 188, 1999, 9.3, 60);
```

б)
```
Delete
From Movies
Where Budget<10;
```

в)
```
Update Movies
Set Release_year = 2001
Where Title='Амелі';
```

г) **Update** Movies
Set Runtime=Runtime+10
Where MID=8;

5.4. Створіть такі три запити, порівняйте результати їх виконання, з'ясуйте призначення функції **Count()**, як використання директиви **Distinct** впливає на результат запиту та для чого потрібно групувати записи за допомогою **Group By**:

д1) **Select Count**(Studio)
From Movies;

д2) **Select Count**(**Distinct** Studio)
From Movies;

д3) **Select** Studio, **Count**(Title)
From Movies
GROUP BY Studio;

5.5. Створіть та запустіть на виконання SQL-оператори для виконання таких операцій у базі даних кінофільмів. **Для оформлення звіту збережіть як SQL-скрипт власне запитів, так і скрін-шоти результатів їх виконання**:

а) Виберіть номер, назву та тривалість для всіх фільмів кіностудії Paramount Pictures.

б) Виберіть назву, рік випуску та бюджет кінофільмів з рейтингом 9,0 та вище.

в) Виберіть повну інформацію про фільми тривалістю більше 2 годин, що вийшли після 2000 року.

г) Виберіть номер, назву та кінокомпанію для фільмів, що вийшли у прокат з 1999 по 2015 рр.

д) Збільшіть на 15 млн бюджет фільму під номером 9.

е) Видаліть дані про фільми з рейтингом, нижчим за 7,0.

ж) Для усіх фільмів кінокомпанії Warner Brothers змініть назву студії на Warner Bros.

з) Додайте під номером 25 дані про кінофільм «За двома зайцями» 1961 року кіностудії ім. О. Довженка тривалістю 72 хв і рейтингом 9,4. Інформації про бюджет фільму немає.

і) У чому відмінність між функціями **Count()** і **Sum()**? Застосуйте їх для визначення загальної кількості кінофільмів та їх сумарного бюджету.

к) Як визначити, скільки фільмів виходило у прокат кожного року?

л) Виведіть на екран назви кінокомпаній без повторів.

м) Визначте сумарний бюджет для кожної кінокомпанії.

Звіт до лабораторної роботи повинен містити:

1. Завдання щодо створення кожного SQL-запиту з п. 5.5.
2. Скрипт SQL-запиту та скрін-шот результату його виконання.
3. Відповіді на запитання для самоперевірки.

Для захисту лабораторної роботи потрібно дати відповіді на контрольні запитання та вміти створювати типові запити.

Запитання для самоперевірки

1. Опишіть основні характеристики традиційних файлових систем і функції, які вони забезпечували.
2. У чому полягали головні причини обмежень традиційних файлових систем та як їх вдалося усунути?
3. Якими характеристиками володіє база даних як корпоративний ресурс?
4. Що таке СУБД? Чим забезпечується наявність нових можливостей СУБД у порівнянні з традиційними файловими системами?
5. У яких формах існує база даних згідно з трирівневою моделлю ANSI SPARC? Що вони позначають?
6. Що спільного та відмінного між зовнішнім та логічним рівнями існування БД?
7. Чим викликана популярність архітекрури «клієнт – сервер» у сучасних СУБД? Які її різновиди?
8. Який формат запису оператора **`Select`** мови SQL та в якому вигляді подається результат його виконання?
9. У який спосіб можна створити запит за участі кількох умов?
10. Що потрібно пам’ятати при використанні у запитах символьних значень?

Практичні завдання для самоперевірки

Як у базі даних реалізувати такі операції засобами мови SQL:

1. Виберіть назви та рік виробництва всіх фільмів у таблиці Movies.
2. Знайдіть номер, назву та тривалість кожного фільму.
3. Виберіть всі кіностудії без повторів.
4. Визначте рейтинг фільмів кіностудії Miramax Films.
5. Знайдіть назви фільмів тривалістю більше 1,5 години.
6. Виберіть номери та назви фільмів, що вийшли у прокат у 2019 р.
7. Виведіть дані про кіностудію, що випустила кінострічку «Форест Гамп». Чи відтвориться результат запиту, якщо у назві фільму заголовною буде лише перша літера?
8. Виберіть назву та рік виготовлення для фільмів, бюджет яких не перевищував 50 млн.
9. Виведіть назву, рік випуску і тривалість фільмів, що не належать кіностудіям Universal Studio або Paramount Pictures.
10. Знайдіть номери кінострічок, випущених після 2000 року з рейтингом, більшим за 8,5.
11. Виведіть дані про кінофільми студії Sony Pictures, виготовлені у 2011 р.
12. Виведіть назви фільмів, випущених з 2005 по 2019 рр.
13. Змініть назву фільму під номером 7 на «Iron Man».
14. Збільшіть на 10 млн бюджет кінострічок студії Miramax Films.
15. Для фільмів з номерами від 3 до 7 зменшіть заявлений бюджет на 10%.
16. Для фільмів з номерами 6 та 12 змініть назву кіностудії на «20th Century Fox».
17. Видаліть усі фільми, рейтинг яких нижчий за 5,0.
18. Видаліть фільми кінокомпанії Universal Studios тривалістю від 1,5 до 2,5 годин.
19. Видаліть фільми кінокомпанії Warner Bros., виготовлені після 2010 року тривалістю менше 2 годин.
20. Додайте новий запис з інформацією про будь-який фільм.

ТЕМА 2. Шаблони подання даних

Модель – це зображення реальності, що окреслює лише її обрані характеристики. Елементи реальності, важливі для однієї моделі, можуть виявитися несуттєвими для іншої. Таку модель реальності втілює база даних, адже саме ця структура передбачена для постійного зберігання відомостей та показників, що генеруються навколишнім світом у процесі виконання операцій. Шляхом маніпулювання даними у різний спосіб користувачі можуть отримати з них інформацію, необхідну для прийняття рішень або успішного виконання функціональних обов'язків.

З позиції моделювання джерело інформації називають предметною областю (ПрО), а систему позначень, що при цьому використовується, – моделлю даних (МД). Опис предметної області у термінах обраної моделі називають концептуальною схемою ПрО.

Шаблони подання даних

Одні і ті самі дані можна зберігати у різних форматах. Якщо говорити про інформацію в електронному вигляді, то форма її організації залежить від змісту власне даних та від способу їх застосування.

Типи баз даних, які ще називають моделями БД або сімействами БД, визначають шаблони і структури, які використовуються для організації даних в системі управління базами даних (СУБД). Вибір типу БД впливає на спосіб подання даних та на операції, які над ними може виконувати прикладна програма.

Текстові файли

Бази даних пласких текстових файлів (*Flat-file databases*) – це перший і найпростіший спосіб подання даних, що найбільш наближений до традиційного підходу зберігання інформації на паперових носіях. Цей метод досі успішно застосовується для роботи з невеликими обсягами інформації, зокрема у середовищі Excel. Однотипна інформація, що зберігається, розбита на окремі поля визначеного типу.

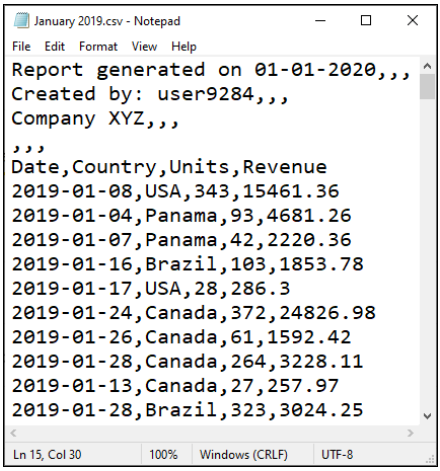


Рис. 2.1. Приклад текстового csv-файлу

Для відокремлення даних різних полів використовують спеціальні символи, як-от кома або крапка з комою в csv-файлах (*CSV – Comma-Separated Values*), двокрапка або пробіл у Unix-подібних системах, взагалі без розділювачів із використанням полів фіксованої довжини (рис. 2.1).

Особливості використання:

- Обмежений тип і рівень складності інформації, що зберігається.
- Важко встановити зв’язки між елементами даних.
- Відсутність функцій паралелізму.
- Підходять тільки для систем, що не висувають значних вимог до читання та записування даних.
- Використовуються для зберігання конфігураційних даних.
- Не потребують залучення стороннього програмного забезпечення.

Приклади:

- `/etc/passwd` та `/etc/fstab` у Linux та Unix-подібних системах
- CSV-файли.

Ієрархічні бази даних

Ієрархічна модель даних перше була задіяна в системі керування інформацією IMS (*Information Management System*) від компанії IBM у рамках проекту висадки на Місяць. У першій ієрархічній системі були повністю реалізовані функції СУБД, зокрема, мови визначення та маніпулювання даними, опис даних, засоби підтримки цілісності, паралелізм, відновлення, а також механізми ефективного оброблення запитів. Згідно з цією моделлю можна створювати бази даних з деревоподібною структурою записів, у якій кожен вузол містить дані свого типу.

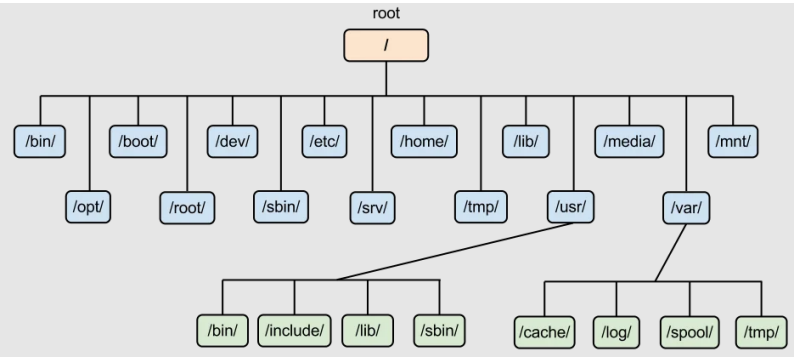


Рис. 2.2. Файлова система – типовий представник ієрархічної БД

Як зображено на рис. 2.2, на верхньому рівні дерева міститься єдиний вузол – «корінь» (root), на наступному рівні розташовуються вузли, пов'язані з цим коренем, потім вузли, пов'язані з вузлами попереднього рівня і т.д. Кожен вузол може мати лише одного попередника або «батька», а такі бази підтримують відношення типу «один-до-багатьох». З огляду на спосіб зберігання даних така структура складається зі схеми елементів або сегментів даних (описова інформація) та їхніх екземплярів (значень). Зокрема, структура ієрархічної БД на рис. 2.3 складається з таких сегментів, як Кафедра, Співробітники, Приміщення тощо. Кожен сегмент може містити свої внутрішні елементи, а саме поіменовану сукупність полів. Наприклад, для сегмента Співробітник полями можуть бути повне ім'я, посада, рік народження та ін.

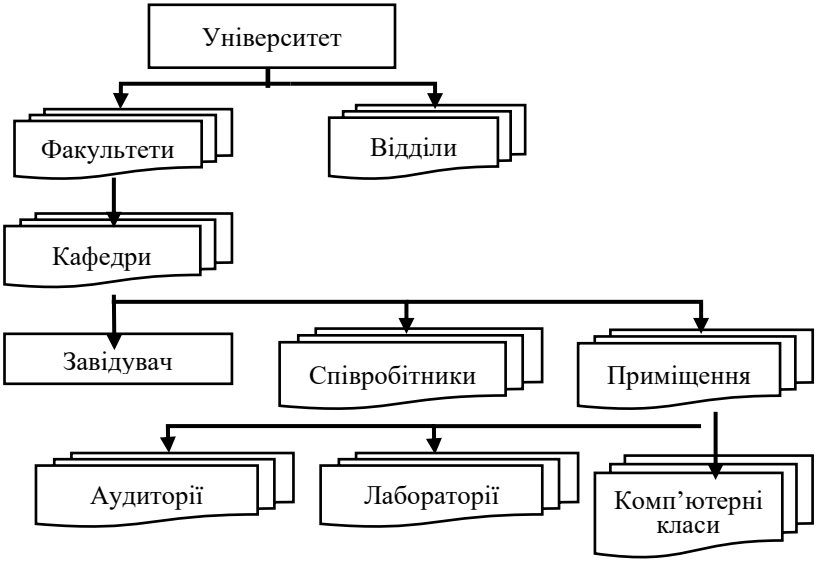


Рис. 2.3. Приклад ієрархічної БД «Університет»

Пошук даних в ієрархічній системі завжди починається з кореня. Потім відбувається сходження з одного рівня на інший, поки не буде досягнуто шуканий запис. Переміщення від одного запису до іншого здійснюється за допомогою посилань.

Особливості використання:

- Простота опису структури реального світу, у якій одні елементи підпорядковуються іншим.
- Швидке виконання запитів, які часто містять надлишкові дані.
- Не завжди зручно починати пошук з кореня.
- Відсутність альтернативного способу переміщення по базі записів.
- Неможливо організувати зв'язки типу «багато-до-багатьох».

Приклади:

- файлові системи
- система доменних імен DNS (*Domain Name System*)
- LDAP (*Lightweight Directory Access Protocol*).

Мережні бази даних

Мережний підхід до організації даних розширює функціональність ієрархічних баз даних. На відміну від останньої, у мережній системі батьківських записів може бути як завгодно багато, що забезпечує моделювання більш складних відношень між сутностями (рис. 2.4).

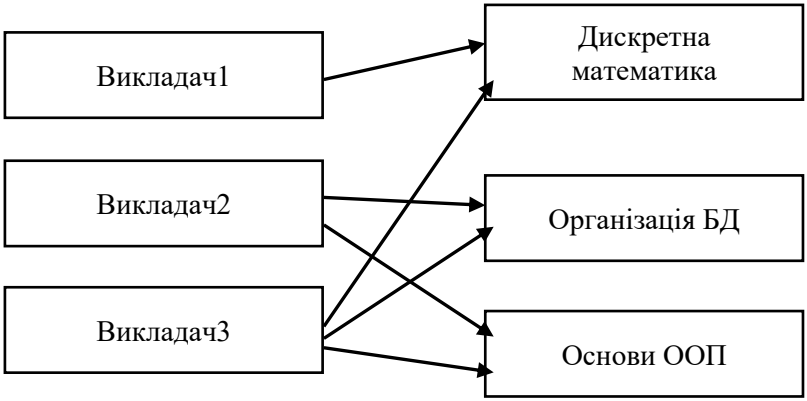


Рис. 2.4. Схема мережної БД, що позначає проведення занять викладачами з різних дисциплін

Особливості використання:

- Замість деревоподібної структури дані подаються у вигляді графу.
- Такі системи досить складні та потребують потужного програмного забезпечення.
- Як і в ієрархічних системах, перехід від запису до запису відбувається за встановленими посиланнями.
- Складність виконання запитів із залученням багатьох вузлів.

Приклади:

- IDMS (*Information Database Management System*).

Концептуальна модель даних

Концептуальна модель бази даних – це проект бази даних, що загалом описує узагальнені інформаційні потреби користувачів. Вона розробляється на однойменному рівні архітектури ANSI SPARC (рис. 1.2) і подає логічну схему всієї бази даних (рис. 2.5).

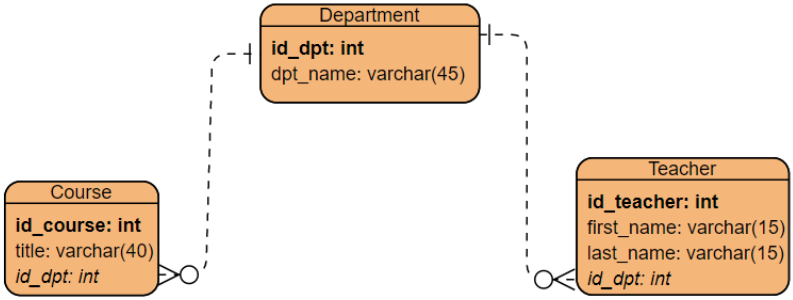


Рис. 2.5. Фрагмент концептуальної моделі БД «Навчальний процес»

Особливості використання:

- Концептуальна модель не дає змоги вносити та обробляти дані.
- Враховує потреби всіх користувачів.
- Не залежить від СУБД або фізичної реалізації БД.
- Приховує складність фізичного зберігання даних.
- Зображає об'єкти та процеси ПрО, про які необхідно зберігати інформацію у БД, їхні характеристики, обмеження та залежності.
- Розробляється перед фізичною реалізацією БД у СУБД.
- Графічно подається у вигляді діаграми «сутність–зв'язок» (*Entity Relationship Diagram,* ***ERD***);
- Основні компоненти ER-діаграми – це сутність, атрибут та зв'язок.

Більше інформації про концептуальну модель та принципи її створення подано у лабораторних роботах № 2 та 3.

Реляційні бази даних

Реляційні (від англ. *relational),* або SQL-бази даних – це найстаріший тип досі широко використовуваних БД загального призначення. Дані об'єктів та явищ, а також зв'язки між ними, організовані згідно з попередньо створеною концептуальною моделлю, фізично втілюються у вигляді таблиць з унікальним набором полів.

Department

id_dpt	dpt_name
248	Cybersecurity
325	Marketing
123	Computer Engineering

Course

id_course	title	id_dpt
1	Information Security	248
2	Market Analytics	325
3	Cryptography	248
4	Computer Circuitry	123
5	Enterprise Marketing	325

Teacher

id_teacher	first_name	last_name	it_dpt
1	Anna	Catmire	123
2	John	Elliot	248
3	Ashley	Hart	123
4	Patrik	Amberson	325
5	Maika	Fernandez	248

Рис. 2.6. Приклад відношень реляційної БД «Навчальний процес»

На рис. 2.6 зображено таблиці, або як їх називають у реляційній моделі, відношення, з внесеними даними та умовними позначеннями зв'язків. Склад цієї реляційної БД та структура

таблиць узгоджуються з концептуальною моделлю, поданою на рис. 2.5. Кожен стовпець таблиці подає деяку важливу характеристику сутності. Дані про конкретний об'єкт або процес зберігаються у формі записів у таблиці. Взаємодію між таблицями реалізовано за допомогою числових кодів – ключів. Унікальне значення ключа позначає у різних таблицях характеристики, притаманні одному екземпляру об'єкта або явища. Завдяки зв'язкам записи однієї таблиці можна доповнювати інформацією з інших, пов'язаних таблиць, а також уникати дублювання та внесення некоректних даних.

Особливості використання:

- Поле зовнішнього ключа, наприклад **id_dpt** таблиці Teacher на рис. 2.6, може містити посилання на записи в інших таблицях, зокрема Department, завдяки чому утворюються зв'язки.
- Високоорганізована структура та гнучкість робить реляційні БД потужними й адаптованими до різних типів даних.
- Для доступу до даних використовується мова SQL.
- Надійний вибір для багатьох застосунків загального призначення.

Властивості баз даних SQL:

Аби гарантувати успішне виконання транзакцій та високий рівень надійності зберігання даних, реляційні БД повинні володіти чотирма властивостями, які позначають абревіатурою **ACID**:

A*tomicity* (Атомарність): гарантує, що кожна транзакція виконується успішно або не спрацьовує взагалі. Часткове виконання операції неприпустиме навіть у випадку повної відмови у роботі системи.

C*onsistency* (Узгодженість)**:** база даних повинна дотримуватися правил, які на кожному кроці виконують перевірку результатів виконання операції та запобігають пошкодженню даних.

I*solation* (Ізоляція)**:** транзакції, що виконуються паралельно, не впливають одна на одну.

***D**urability* (Надійність)**:** Transactions are final, and even system failure cannot “roll back” a complete transaction.

Приклади:

- MySQL
- MS SQL Server
- PostgreSQL
- Oracle Database
- Azure SQL Database.

Докладніше реляційну модель даних розглянемо в темі 3.

Нереляційні бази даних

Бази даних NoSQL (від англ. *non* або *non relational SQL*) – це категорія баз даних, що при зберіганні та видобуванні інформації втілюють підходи, відмінні від стандартного реляційного шаблону [2–5]. Такі бази даних існували вже в другій половині 1960-х років, проте гучно заявили про себе на початку XXI століття після сплеску використання Web 2.0 у продуктах таких компаній, як Facebook, Google та Amazon.com. Бази даних NoSQL дедалі більше використовуються при опрацюванні великих обсягів даних та у веб-застосунках реального часу. NoSQL-системи ще називають «Not only SQL» – не тільки SQL, аби наголосити, що вони також підтримують SQL-подібну структуру даних та мову запитів.

Передумови такого підходу до впорядкування даних:

- відсутність попереднього визначеної схеми даних;
- підтримка горизонтального масштабування;
- розподілення копій даних між кількома вузлами;
- пришвидшення обробки великих обсягів даних;
- зберігання слабо структурованої інформації;
- гнучкість подання різнотипної інформації;
- інтуїтивно зрозуміла схема подання даних;
- багатовимірність даних.

Наразі вирізняють чотири основні категорії баз даних NoSQL:

- «ключ–значення»;
- документо-орієнтовані;
- стовпчикові;
- графові.

Бази даних «ключ–значення»

Сховища типу «*key–value*» – це одна з найпростіших баз даних NoSQL, у якій довільний фрагмент інформації, як-от JSON-об'єкт, BLOB (*Binary Large Objects* – великі бінарні об'єкти), зображення або текст, супроводжується визначеним ключем, за яким потім можна здійснювати пошук.

Рис. 2.7. Подання даних за принципом «ключ-значення» [2]

Користувач повинен надавати унікальні та змістовні назви ключів, завдяки чому можна ідентифікувати значення, а також з'ясувати їх тип і формат (рис. 2.7).

Сховища «ключ–значення» користуються найбільшою популярністю у типових розробках для зберігання простих значень, які підлягатимуть подальшому опрацюванню за межами БД. Як приклад можна розглянути інтернет-магазин, який торгує книжками, музичними дисками та фільмами. Кожна група товарів має дещо відмінні характеристики, які необхідно надавати клієнтам. Для таких цілей підійде база даних «ключ–значення», приблизний вигляд якої наведено на рис. 2.8.

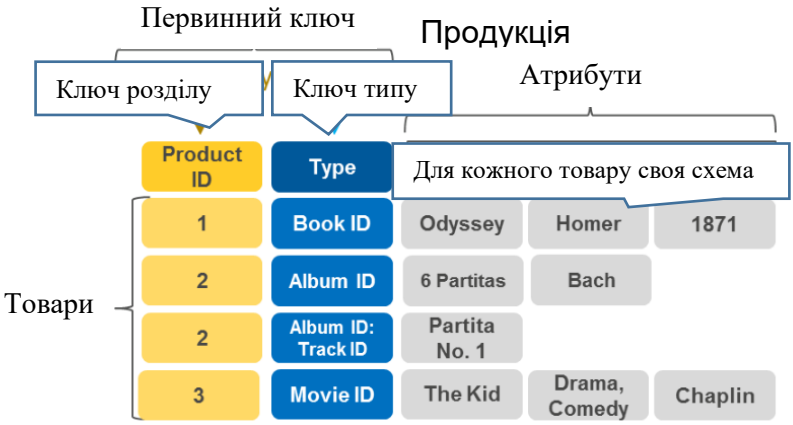


Рис. 2.8. Використання ключів для ідентифікації різнотипних атрибутів

Кожен продукт позначається номером розділу (Partition Key, де 1 – позначає книгу, 2 – музичний альбом, 3 – фільм), а також унікальний ключ елемента в межах розділу (Sort Key, узагальнено позначений як Book, Album або Movie ID). За такою схемою для кожного типу товару можна зберігати свій набір важливих характеристик – атрибутів.

Особливості використання:

- Сховища забезпечують швидкий доступ із незначним навантаженням на ресурси.
- Часто зберігають дані налаштувань, інформацію про стан системи, параметри застосунку або прапорці для веб-сайтів.
- Підійдуть для зберігання даних за принципом словника або хеш-таблиці.

Приклади:

- Berkley DB
- Dynamo
- Redis
- Memcached
- ZooKeeper
- etcd.

Документо-орієнтовані бази даних

Бази даних або сховища документів (*document-oriented*) забезпечують доступ і пошук за принципом «ключ–значення», оскільки також використовують ключ для унікальної ідентифікації окремих частин інформації. Дані структуровані у вигляді документів та колекцій. Головна відмінність від попереднього представника баз даних NoSQL полягає у тому, що замість BLOB-об'єктів з необробленими даними, документо-орієнтовані бази можуть зберігати дані в структурованих документах формату PDF, Microsoft Word DOC, JSON або XML.

На противагу традиційним таблицям баз даних, що складаються зі стовпців визначеного типу, документи містять пари ключів та відповідних їм значень і не повинні мати однакову структуру (рис. 2.9). Завдяки цьому, для внесення додаткових даних можна просто долучити більше документів без змін у структурі всієї бази даних.

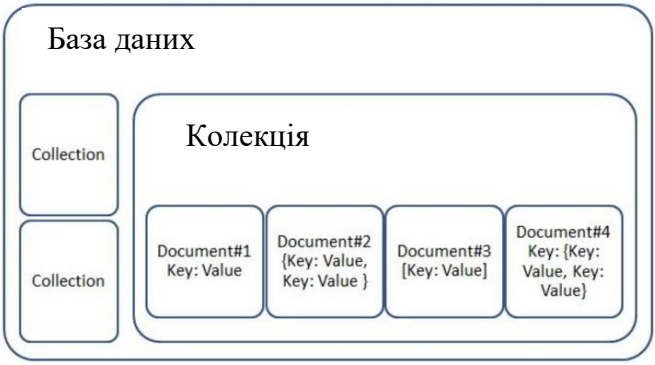


Рис. 2.9. Структура документо-орієнтованої бази даних

Документи зібрано в колекції, за призначенням подібні до реляційних таблиць. У сховищі документів можна виконувати запити для пошуку колекцій із визначеними атрибутами. До переваг цих баз даних належать гнучкість моделювання даних, адже немає потреби вкладати дані у рамки реляційної моделі. Вони можуть обробляти структуровану, неструктуровану та слабо структуровану інформацію та забезпечувати високу

продуктивність запису даних при жорсткій узгодженості (вагома перевага при використанні гнучкої методології Agile).

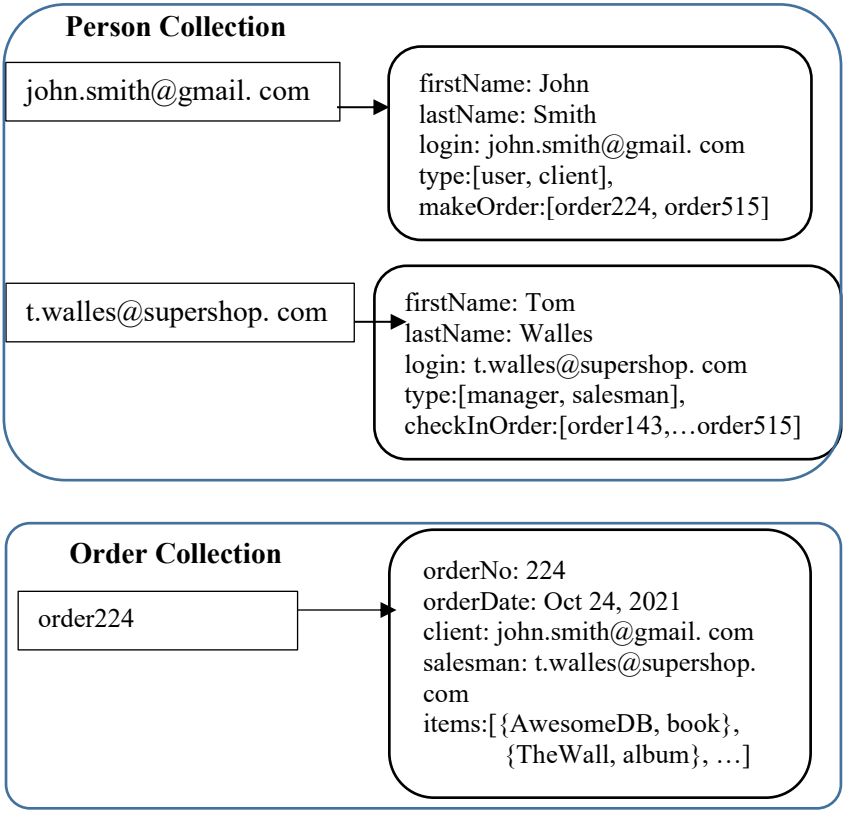


Рис. 2.10. Вміст колекцій і документів у документо-орієнтованій базі даних інтернет-магазину

Документо-орієнтовані бази даних також забезпечують відокремлення колекцій за сутностями (профілі користувачів, товарів, замовлень тощо), як зображено на рис. 2.10. Проте ці бази даних не підтримують розширених запитів та не дають змоги використовувати операції з'єднання (joins).

Особливості використання:

- Попри те, що дані в документах організовані відповідно до деякої структури, бази даних документів не передбачають жодного конкретного формату або схеми.
- Кожен документ може відрізнятися за внутрішньою структурою, яку інтерпретує база даних.
- Вміст файлів, що зберігаються у базі даних документів, можна аналізувати за допомогою запитів.
- Бази даних документів добре підходять для швидкої розробки застосунків.
- Кожен документ у базі даних самостійний із власною структурою подання даних.
- У будь-який момент можна змінювати властивості даних, не змінюючи структуру документа або самі дані.

Приклади:

- MongoDB
- RethinkDB
- MarlLogic
- CouchDB
- BaseX
- IBM Domino
- Elasticsearch.

Стовпчикові бази даних

Стовпчикові бази даних (інші назви: нереляційні сховища на основі стовпців, бази даних із широкими стовпцями, англ. *wide column store*) належать до сімейства NoSQL БД, проте за організацією даних схожі на реляційні.

У реляційних БД усі рядки відповідають фіксованій схемі, яка визначає характеристики, які міститиме таблиця, їхні типи даних та інші критерії. У стовпчикових базах для подання даних замість таблиць використовуються «сімейства стовпців» (англ. *column family database*). Ці сімейства містять рядки, кожен з яких може мати власну схему атрибутів, яку можна легко змінювати без впливу на інші записи. Рядок складається з унікального ідентифікатора, використовуваного для пошуку, за яким розміщено набори стовпців.

	company				
ID	name	address		website	
1	DataX	city	San Francisco	protocol	https
		state	California	domain	datax.com
		street no.	135	subdomain	www
		street	Kearny St		
2	sPan	country	Spain	protocol	http
		zip code	28004	domain	span.es
		city	Madrid	subdomain	www
		street	Plaza del Rei	email	info@span.es
		building	1		

Стовпець

Сімейство стовпців

Рис. 2.11. Приклад NoSQL бази даних із широкими стовпцями

Wide column store можна уявити як двовимірну базу даних «ключ–значення», де стовпчик може містити довільну кількість додаткових атрибутів (рис. 2.11).

Особливості використання:

- Бази даних на основі сімейства стовпців добре працюють із застосунками, для яких важлива висока продуктивність і масштабованість при опрацюванні рядків.
- Оскільки всі дані та метадані запису доступні через єдиний ідентифікатор рядка, для пошуку та виведення необхідної інформації не потрібно використовувати обчислювально складні операції з'єднання типу join.
- Стовпчикові бази даних орієнтовані строго на операції над рядками.
- Зведені запити, як-от визначення суми, середнього значення, групування, та інші процеси, що передбачають підбиття підсумків, узагальнення та створення звітів, виконати складно, якщо взагалі можливо.

- БД цього типу можна використовувати для рекомендаційних механізмів, каталогів, виявлення шахрайства тощо.

Приклади:

- Google BigTable
- HBase
- Scylla
- Vertica
- Apache Cassandra.

Графові бази даних

Графові бази даних (*graph DB*) передбачені для зберігання взаємозв'язків між об'єктами та навігації між ними.

Вузлами графової бази даних можуть бути екземпляри сутностей, що генерують дані або подають деяку важливу характеристику. Проте найбільшу цінність становлять ребра, які позначають зв'язки між об'єктами. Кожне ребро має початковий і кінцевий вузли, а також свій тип та напрямок. Ребра можуть описувати взаємодію об'єктів за принципом «батько–нащадок», позначати вплив, права власності тощо. Обмежень на кількість та тип зв'язків, що їх може утворювати вузол, немає.

Обхід у графовій базі даних можна здійснювати або по ребрах визначених типів, або по всьому графу. Причому навігація між вузлами виконується дуже швидко, оскільки з'єднання не прокладаються під час виконання запиту, а зберігаються у базі даних. Графовим базам даних властива низка переваг при застосуванні у соціальних мережах, сервісах поширення реклами, рекомендацій і системах виявлення шахрайства, коли потрібно утворювати взаємозв'язки між даними та швидко їх запитувати. У виконанні таких завдань вони суттєво випереджають реляційні БД за продуктивністю, простотою, способом внесенням змін і наочністю подання інформації.

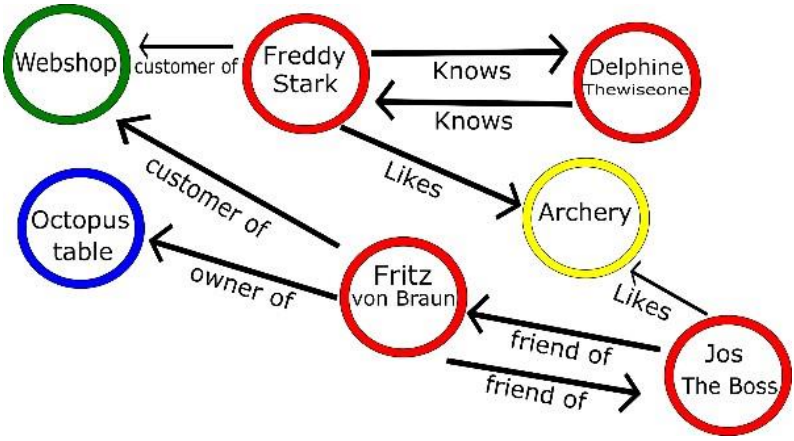


Рис. 2.12. Приклад сховища даних типу граф

На рис. 2.12 наведено приклад графу, який доцільно використати для поширення таргетованої (спрямованої) реклами. Тут можна відстежувати дружні стосунки клієнтів порталу електронної комерції (Webshop) у соцмережах (Freddy Stark знайомий з Delphine Thewiseone) та на основі їх інтересів (Archery – Стрільба з луку) пропонувати рекламу відповідних ресурсів.

Особливості використання:

- Найкраще підходять для реалізації проектів, які передбачають природну графову структуру даних.
- За виглядом схожі на мережні бази даних.
- Зосереджені на зв'язках між елементами.
- Явно відображають зв'язки між типами даних.
- Не вимагають покрокового переміщення між елементами.

Приклади:

- ArangoDB
- FlockDB
- Giraph
- HyperGraphDB
- Neo4j
- OrientDB.

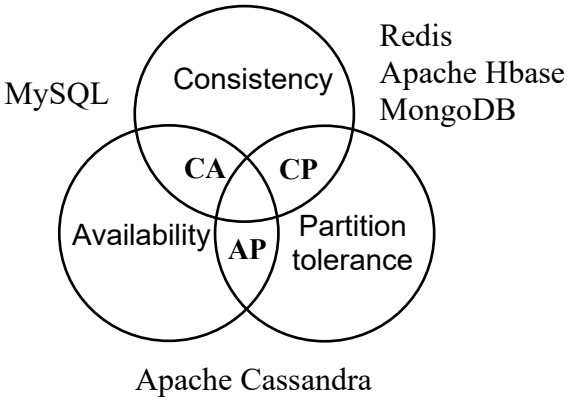


Рис. 2.13. Схематичне зображення теореми CAP

Властивості баз даних NoSQL:

Порівняно з реляційними SQL базами даних, яким властиві принципи ACID, сховища NoSQL здебільшого дотримуються теорії **CAP** (рис. 2.13). Однойменна теорема стверджує, що у будь-якій комп'ютерній системі розподілення даних неможливо одночасно забезпечити більше двох із наведених нижче трьох принципів.

C*onsistency* (Узгодженість): усі вузли бачать дані однаково, а результатом кожного запиту є найбільш актуальні дані або повідомлення про помилку.

A*vailability* (Доступність): результат кожного запиту коректний і не містить помилкових даних.

P*artition tolerance* (Стійкість до розподілення**)**: попри розбиття даних на окремі ізольовані частини та втрату зв'язку між деякими вузлами, система не втрачає стабільності та здатності коректно обробляти запити.

Оскільки Partition tolerance – обов'язкова властивість розподілених баз даних NoSQL, такі системи можуть водночас бути узгодженими або стійкими до розподілення (CP) або доступними та сійкими до розподілення (AP). Завдяки цим властивостями можна класифікувати нереляційні бази.

Наприклад, першочерговим важелем архітектури MongoDB є узгодженість. Водночас Apache Cassandra робить наголос на

доступності. Це не означає, що при використанні MongoDB виникають проблеми з доступом, а Cassandra містить неузгоджені дані. Вони підкреслюють яка властивість обов'язкова, а яка може налаштовуватися.

Рекомендації щодо застосування SQL і NoSQL баз даних

SQL є гарним варіантом для роботи з взаємопов'язаними даними. Реляційні бази даних ефективні, гнучкі та доступні для більшості застосунків. Перевага реляційних баз даних у тому, що коли один користувач оновлює конкретний запис, кожен екземпляр бази даних автоматично оновлюється, а інформація надається у реальному часі [5].

SQL і реляційні бази даних спрощують оброблення великих обсягів структурованої інформації, здатні до розширення та надають гнучкий механізм створення запитів. При виконанні операцій доступ здійснюється лише до потрібних даних, наприклад, операція оновлення даних виконується одноразово, замість того, щоб вносити зміни до кількох файлів. Завдяки цьому поліпшується цілісність даних. Оскільки кожен фрагмент інформації зберігається в єдиному екземлярі, не виникає плутанини з різними версіями.

До найбільших IT-компаній, які користуються послугами SQL, належать Uber, Netflix і Spotify. Навіть такі великі корпорації, як Google, Facebook і Amazon, що підтримують власні системи баз даних, усе ще користуються SQL для створення запитів та аналізування даних.

Якщо цінність SQL полягає в забезпеченні достовірності даних, бази даних NoSQL виявилися ефективними у роботі з великими даними, так званими Big Data. Такі бази даних також підійдуть для компаній, яким потрібно швидко масштабуватися через зміну вимог. Простий у використанні та гнучкий підхід NoSQL забезпечує високу продуктивність.

NoSQL також є вдалим рішенням, коли потрібно мати справу з великими обсягами постійно змінюваних даних, при роботі з гнучкими моделями даних, які неможливо звести до фіксованого набору реляційних відношень. Для роботи з великою кількістю неструктурованих даних підійдуть документо-орієнтовані бази, на кшталт CouchDB, MongoDB і Amazon DocumentDB. Швидкий

доступ до сховищ типу «ключ–значення», хоча і без належної підтримки цілісності інформації, надає Redis. Коли потрібно виконати складний та гнучкий пошук серед великої кількості даних, хорошим варіантом може бути Elasticsearch.

Важливою перевагою баз даних NoSQL є їхня масштабованість. На відміну від SQL-аналогів, їхній вбудований шардинг та вимоги щодо високої доступності підтримують горизонтальне масштабування. До того ж, бази даних NoSQL, як-от Cassandra, розроблена компанією Facebook, обробляють величезні обсяги даних, розподілених по багатьох серверах, що усуває небезпеку єдиної точки відмови та сприяє максимальній доступності.

Реляційні бази даних доцільно використовувати, коли:

- необхідна повна узгодженість даних
- дані структуровані
- потрібно виконувати операції за участі багатьох документів або складних з'єднань (joins).

Бази даних NoSQL варто обрати, якщо:

- потрібно обробляти великі обсяги даних
- система повинна бути високопродуктивною
- дані неструктуровані з гнучким набором характеристик
- необхідно забезпечити доступність і масштабованість системи.

ЛАБОРАТОРНА РОБОТА № 2
Розроблення концептуальної моделі бази даних

Мета

Провести високорівневий аналіз предметної області, для якої розробляється база даних, та на його основі створити концептуальну модель бази даних.

Завдання

Здійснити опис об'єктів та процесів, про які необхідно зберігати інформацію в базі даних; визначити основні сутності предметної області; охарактеризувати їх за допомогою атрибутів та зобразити виокремлені елементи на ER-діаграмі.

Теоретичні відомості

Об'єктно-орієнтоване проектування – це процес створення схеми бази даних на основі аналізу та моделювання об'єктів предметної області [6, 7].

Технологічно процес об'єктно-орієнтованого проектування передбачає:

- Опис основних об'єктів предметної області, коли спеціаліст отримує інформацію про те:
  - як об'єкт функціонує;
  - які операції виконуються над об'єктом;
  - які операції виконує сам об'єкт.
- Опис характеристик об'єктів, при якому визначаються основні атрибути (властивості, ознаки, риси, ідентифікатори), тобто важливі характеристики об'єкта, які потрібно зберігати;
- Утворення зв'язків між об'єктами, узагальнення кількох об'єктів за спільними ознаками, що дозволяє оптимізувати модель, скоротивши кількість таблиць у майбутній базі даних.

Робота виконується з використанням CASE-засобів (англ. *Computer-Aided Software Engineering*) проектування баз даних, до яких належать: DB Designer, ERwin Data Modeler, SQL Power Architect, Visual Paradigm, Vertabelo, Skipper та ін.

Увесь процес розробки концептуальної, або об'єктно-орієнтованої, моделі бази даних складається з таких етапів:

На першому етапі здійснюється опис об'єктів предметної області. Він полягає у виокремленні та описі існуючих об'єктів і процесів.

- ❖ Сукупність однотипних об'єктів, про які необхідно зберігати інформацію в БД, узагальнено позначають **сутністю**.

В описі предметної області сутності – це іменники, що відповідають на запитання Хто? Що? *Наприклад*, у предметній області *Навчальний процес* можна виокремити такі об'єкти, а отже, й сутності: Викладач, Студент, Дисципліна, Кафедра, Факультет, Розклад тощо. На ER-діаграмі сутність позначається прямокутником з унікальною змістовною назвою.

Сутність – це аналог таблиці бази даних.

Розрізняють конкретні та абстрактні сутності.

- **Конкретні** (об'єкти, предмети) – самостійні, батьківські, що містять вихідну інформацію і можуть слугувати основою для інших сутностей: Студент, Робітник, Товар, Відділ, Група.
- **Абстрактні** (процеси, явища) – залежні, похідні, дочірні, пов'язані з іншими сутностями, використовують їх властивості: Екзамен, Розклад роботи, Замовлення, Договір, Постачання.

- ❖ **Екземпляр сутності** – це реальні представники того чи іншого об'єкта, про які потрібно зберігати дані в базі.

Наприклад, Організація баз даних, Дискретна математика, Комп'ютерна арифметика, Теорія електричних кіл – це екземпляри сутності Дисципліна. Записи в таблиці містять дані про кожен екземпляр сутності (конкретний об'єкт або процес).

Другий етап у розробці схеми БД полягає в описі **атрибутів** – характеристик об'єктів предметної області. Атрибути повинні нести важливу інформацію для конкретної системи, бути спільними для всіх екземплярів сутності, використовуватися у деяких операціях предметної області.

- ❖ **Атрибути – це поля (стовпці) майбутніх таблиць**. Кожен атрибут характеризується назвою, типом даних, довжиною та обмеженнями.

Наприклад, для сутності Кафедра у предметній області *Навчальний процес* важливими будуть такі атрибути, як: Назва, Спеціальність, Факультет, Адреса тощо.

- Серед атрибутів будь-якої сутності повинен бути хоча б один **атрибут первинного ключа**, за допомогою якого можна унікально ідентифікувати записи в таблиці.

Зазвичай, це атрибут «Порядковий номер», у назві якого часто використовують скорочення “id” («ай-ді», ідентифікатор).

Третій етап розробки концептуальної моделі передбачає виокремлення **сутностей-класифікаторів**. Вони утворюються на основі атрибутів, якщо:

- у сутності є атрибут, який має власні характеристики та може існувати у предметній області у вигляді самостійного об’єкта, як-от атрибут Розклад;
- якщо можливі значення атрибута ніколи або дуже рідко змінюються, наприклад, Жанр книги або Спеціальність.

У цих випадках атрибут виокремлюється у самостійну сутність, встановивши з вихідною сутністю відповідний зв’язок.

Наприклад, у сутності Кафедра можна виокремити як сутність атрибут Спеціальність, для якого можна вказати номери і назви всіх спеціальностей, наявних в університеті. Цей перелік фіксований і зазнає нечастих змін у разі створення або скорочення деякої спеціальності.

Четвертий етап полягає у створенні зв’язків між усіма сутностями. На ER-діаграмі визначено три типи зв’язків: «один-до-одного» (1:1), «один-до-багатьох» (1:M) або «багато-до-багатьох» (N:M). Останній тип зв’язку в подальшому потребує вдосконалення через оптимізацію моделі бази даних.

- **Зв’язок, який володіє власними характеристиками, слід оформити як самостійну сутність.**

Наприклад, якщо між сутностями Студент і Викладач існує зв’язок *Екзамен*, то він володіє власними характеристиками, як-от: номер відомості, дата проведення, оцінка тощо. Отже, зв’язок Екзамен виокремлюється у додаткову сутність і пов’язується з вихідними сутностями.

- **Для того, щоб між сутностями був зв’язок, у них повинен бути принаймні один спільний атрибут**.

Наприклад, сутності Студент і Група пов'язані через спільний атрибут – номер групи.

ПРИКЛАД ВИКОНАННЯ ЗАВДАННЯ

Предметна область – *Мережа кінотеатрів.*

Постановка задачі:

Розробити схему бази даних для інформаційної системи мережі кінотеатрів, що діє по всій Україні. Кожен кінотеатр має визначену назву, адресу та може містити кілька залів для кінопоказу. Фільми, що виходять у прокат, мають низку характеристик, серед яких – назва, рік випуску, жанр, тривалість тощо. Показ фільмів відбувається у різних кінотеатрах протягом визначеного періоду. Глядачам повинен надаватися розклад сеансів із зазначенням часу початку сеансу та залу, в якому демонструється фільм.

Опис виокремлених об'єктів, наведений у табл. 2.1, розпочато з найбільш важливих сутностей, на яких заснована наша система – це Фільми та Кінотеатри. Сутність Зали не самостійна, а пов'язана з конкретним кінотеатром. Показ фільмів та Сеанси також є похідними сутностями, залежними від трьох попередньо описаних об'єктів. Операції, зазначені для об'єктів, допомагають розібратися у тому, як вони взаємодіють або впливають одне на одного.

Таблиця 2.1

Приклад опису об'єктів предметної області «Мережа кінотеатрів»

Об'єкт	Опис	Операції
Фільми	Конкретна сутність. Узагальнене позначення фільмів. Характеризується атрибутами назва, рік випуску, тривалість і т. д.	• виходить у прокат • демонструється у кінотеатрах • існують сеанси перегляду
Кінотеатри	Конкретна сутність. Об'єкт у мережі для перегляду фільмів, має	• демонструє фільми протягом певного

Об’єкт	Опис	Операції
	такі характеристики, як місто, адреса, кількість залів.	періоду згідно з графіком • у різних містах • за розкладом сеансів
Зали	Конкретна сутність, що узагальнено позначає зали у різних кінотеатрах. Слід зазначати назву залу (малий, великий, блакитний, дитячий), кінотеатр, якому належить, та кількість глядацьких місць.	• у кінотеатрі може бути один або більше залів • позначається назвою • містить фіксовану кількість місць • місце показу різних фільмів
Показ фільмів	Абстрактна сутність. Фіксує графік виходу фільмів у прокат. Зазначає фільм та період його показу.	• терміни демонстрації фільму • визначено для кожного кінотеатру
Сеанси	Абстрактна сутність, що описує розклад показу фільмів у кінотеатрі. Потребує для кожного показу зазначення: кінотеатру, залу та часу початку.	• сеанси тривають, поки фільм у прокаті • в одному кінотеатрі може відбуватися кілька сеансів протягом дня • один фільм може демонструватися у різних залах • глядачі купують квитки саме на певний сеанс

Для виокремлених об’єктів (сутностей) необхідно визначити найбільш важливі характеристики (атрибути).

Рекомендації щодо визначення атрибутів сутностей:

- Для унікальної ідентифікації кожного екземпляра сутності серед її характеристик обов'язково повинен бути **первинний ключ**. Він не несе у собі жодного змістового навантаження і виконує роль порядкового номера. Первинний ключ повинен бути **цілочисловим**.
- Атрибут повинен бути простим, не містити різнотипної інформації та відображати деяку характеристику об'єкта, суттєву для конкретної предметної області.
- Якщо атрибут містить внутрішні компоненти, до яких при роботі з БД потрібно буде окремо звертатися, їх слід подавати у вигляді самостійних атрибутів (полів таблиці*). Наприклад, для зберігання в інтернет-магазині даних про доставку замовлення можна виділити окремі атрибути, як-от номер декларації, служба доставки, місто відправлення, номер відділення, вартість доставки тощо.*
- Атрибут повинен бути атомарним, тобто зберігати для кожного екземпляра сутності рівно одне значення. *Отже, атрибут не може складатися з діапазону, переліку або набору кількох значень.*
- Діапазон (мінімальне і максимальне значення) або період краще замінити двома атрибутами, що позначають початок і кінець (ліву і праву межі діапазону). *Наприклад, період демонстрації фільму необхідно подати за допомогою двох атрибутів: дати початку і дати завершення показу.*
- Якщо атрибут набуває переліку значень, найімовірніше, його потрібно виокремити в самостійну сутність. *Наприклад, у фільмі може грати декілька акторів або замовлення складається з різних товарів.* Такі випадки буде детальніше розглянуто у наступних лабораторних роботах.
- Не варто створювати атрибут, значення якого періодично змінюється. *Наприклад, замість віку краще використовувати рік народження, замість терміну дії – дату завершення терміну придатності, замість кількості проданих квитків – фіксувати купівлю кожного квитка окремо, а підсумкову інформацію визначити за допомогою запиту та агрегатних функцій, як-от Count() або Sum().*

- Перелік можливих значень, яких може набувати деякий атрибут, доцільно оформити у вигляді окремої сутності. До них належать дні тижня, назви місяців, способи оплати, типи номерів у готелі, перелік послуг, що надаються, тощо.

Характеристики, виокремлені для кожної сутності мережі кінотеатрів, наведено у табл. 2.2. Надалі при створенні схеми БД будемо використовувати назви латиницею.

Таблиця 2.2

Атрибути виокремлених об'єктів предметної області «Мережа кінотеатрів»

Сутність	**Атрибут**	**Опис**
Фільм **(Film)**	Номер фільму **(film_id)**	Унікальний ідентифікатор цілого типу. Умовний порядковий номер кожного фільму, що виходить у прокат. Первинний ключ
	Назва фільму **(film_title)**	Простий атрибут символьного типу
	Рік випуску (**release_year**)	Простий атрибут цілого типу
	ЖАНР (**Genre**)	Атрибут, що позначає жанр фільму (комедія, драма, детектив, мультфільм, фентезі, бойовик, трилер, тощо). Може набувати фіксованих значень з визначеного переліку, що повторюватимуться для різних фільмів. **Потребує подальшого уточнення**
	Тривалість (**duration**)	Простий атрибут цілого типу, що визначає тривалість фільму в хвилинах

Сутність	Атрибут	Опис
Кінотеатр **(Cinema)**	Номер кінотеатру **(cinema_id)**	Унікальний ідентифікатор цілого типу. Умовний порядковий номер кожного кінотеатру в мережі. Первинний ключ
	Назва кінотеатру **(cinema_name)**	Простий атрибут символьного типу, що зберігає назву кінотеатру
	Місто **(city)**	Простий атрибут символьного типу. Містить назву міста, у якому працює кінотеатр
	Адреса **(address)**	Атрибут символьного типу, що містить адресу кінотеатру
	Кількість залів (**number_halls**)	Простий атрибут цілого типу. Позначає кількість залів у кінотеатрі
Зал **(Hall)**	Номер залу **(hall_id)**	Унікальний ідентифікатор цілого типу. Кожен зал у мережі кінотеатрів має свій порядковий номер
	Назва залу **(hall_name)**	Назва, символьне позначення залу. Простий атрибут
	КІНОТЕАТР ***(*Cinema*)***	Атрибут, що умовно позначає кінотеатр, до якого належить зал. Повинен вказувати на відповідний запис про кінотеатр у сутності Cinema. **Потребує подальшого уточнення**
	Кількість місць (**number_seats**)	Простий цілочисловий атрибут для зберігання загальної кількості місць у залі

Сутність	Атрибут	Опис
Показ фільмів (**Film_ Release**)	Номер показу (**release_id**)	Унікальний ідентифікатор цлого типу. Вихід на екрани кожного нового кінофільму позначається окремим порядковим номером
	ФІЛЬМ *(***Film***)*	Атрибут, що умовно позначає фільм у прокаті. Повинен вказувати на відповідний запис про кінофільми у сутності Film. **Потребує подальшого уточнення.**
	Дата початку показу (**start_date**)	Атрибути, що позначають період, діапазон, інтервал, розбивають на два прості атрибути: дату початку і дату кінця типу datetime
	Дата завершення показу (**end_date**)	Аналогічно до попереднього атрибуту дати початку (див. вище)
Сеанс (**Film_ Show**)	Номер сеансу (**show_id**)	Атрибут цілого типу, що умовно та унікально позначає показ фільму в будь-якому кінотеатрі
	ПОКАЗ ФІЛЬМУ (**Film release**)	Атрибут, що умовно позначає прокат визначеного фільму. Повинен вказувати на відповідний запис у сутності Film_Release. **Потребує подальшого уточнення**

Сутність	Атрибут	Опис
	ЗАЛ *(**Hall**)*	Атрибут, що вказує на зал у відповідному кінотеатрі. У сутності Зали таку відповідність уже встановлено. **Потребує подальшого уточнення**
	Час початку сеансу (**show_time**)	Простий атрибут темпорального типу, що зберігатиме час початку сеансу

Оскільки у сутності Фільми атрибут Жанр може набувати фіксованих значень, які повторюватимуться для різних екземплярів, виокремимо його як однойменну сутність, опис якої наведено у табл. 2.3. У такий спосіб із ним буде простіше працювати з точки зору доповнення або оновлення, а також зберігання характеристик фільмів.

Таблиця 2.3

Атрибути об'єкта Жанр (Genre)

Атрибут	Опис
Номер жанру (**genre_id**)	Первинний ключ, атрибут, що унікально ідентифікує кожен жанр, у якому знімаються кінофільми. Цілочислове поле
Назва жанру **(genre_name)**	Назва жанру, позначена відповідним первинним ключем. Символьне поле. Простий атрибут

На прикладі сутності Жанр продемонструємо фрагмент таблиці, на яку ця сутність перетвориться у базі даних (табл. 2.4). Звісно, що у базі буде створено порожню таблицю, а дані внесені частково, аби продемонструвати її можливості. Таблиця, що складається з двох полів (атрибутів), під унікальним номером зберігає назву кожного жанру. У разі потреби до таблиці можна легко додати новий рядок з інформацією про додатковий жанр.

Таблиця 2.4

Приклад таблиці Genre (Жанр) у базі даних

genre_id	genre_name
1	Детектив
2	Мелодрама
3	Пригодницький
4	Комедія
5	Трилер
6	Фантастика

Після попереднього аналізу предметної області виокремлені сутності визначеного складу зображено у вигляді діаграми «сутність–зв'язок » (ER-діаграми) на рис. 2.14. Наведена схема БД поки що не містить зв'язків та потребує вдосконалення. Проте вже зараз на ній можна побачити ключові для конкретної інформаційної системи об'єкти та процеси, їх характеристики, важливі для зберігання й обробки, а також те, як вони взаємопов'язані.

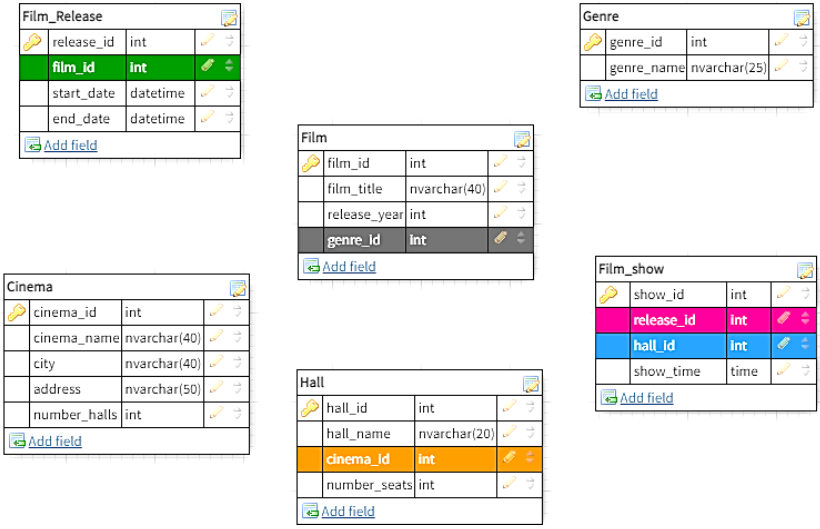


Рис. 2.14. Попередній набір сутностей БД «Мережа кінотеатрів»

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. *Ознайомлення з варіантом індивідуального завдання (с. 183).*

2. *Визначення сутностей майбутньої БД.*

2.1. Насамперед слід виокремити базові найважливіші об'єкти, без яких ваша система не змогла б існувати. Об'єкти, явища або процеси, про які необхідно зберігати інформацію. Зазвичай, це іменними, які відповідають на запитання Хто? Що?

2.2. У процесі аналізу просувайтесь від самостійних об'єктів до залежних або похідних.

2.3. Надавайте сутностям змістовні назви, що узагальнено позначають усю сукупність однотипних об'єктів/процесів.

2.4. Не слід плутати сутності (Фільм) з екземплярами сутності («Пірати карибського моря», «Форест Гамп», «Маленькі жінки»).

3. *Визначення атрибутів об'єктів БД.*

3.1. Для кожної сутності визначте суттєві для вашої бази характеристики (атрибути), дані про які потрібно буде зберігати.

3.2. Обов'язково забезпечте наявність унікального ідентифікатора (первинного ключа). Перелік сутностей та їхні атрибути зобразіть у вигляді табл. 2.5.

Таблиця 2.5

Об'єкти та атрибути предметної області «НАЗВА»

Об'єкт	**Атрибут**	**Опис**

За приклад можна взяти заповнення табл. 2.2.

УСІМ ОБ'ЄКТАМ ТА АТРИБУТАМ ПОТРІБНО ДАВАТИ ЗМІСТОВНІ НАЗВИ ЛАТИНСЬКИМИ ЛІТЕРАМИ (АНГЛІЙСЬКОЮ АБО ТРАНСЛІТОМ).

3.4. Перевірте чи немає серед атрибутів таких, що володіють власними характеристиками і які варто виокремити в самостійні сутності.

3.5. Атрибутам, пов'язаним з іншими сутностями, надавайте узагальнені назви (як-от у прикладі в табл. 2.2, Фільм, Кінотеатр, Зал тощо), тип зазначайте умовно.

4. *Створення схеми БД.*

В онлайн-середовищі **DB Designer**, доступному за посиланням https://app.dbdesigner.net/, для створення нової схеми БД у меню **Schema** (Схема) оберіть пункт **New** (Нова) (рис. 2.15):

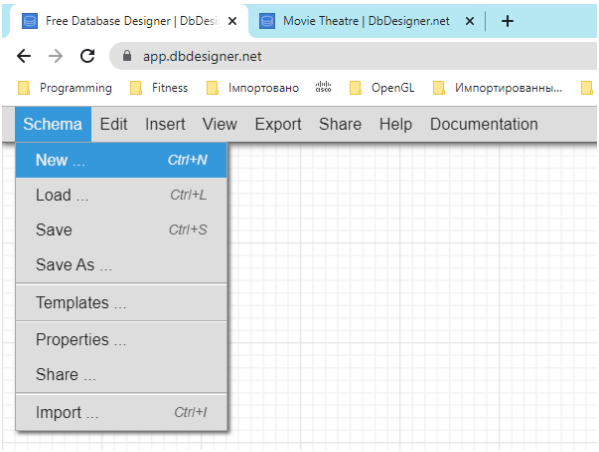


Рис. 2.15. Створення нової діаграми БД у DB Designer

Дайте діаграмі змістовну назву (**Title**) та оберіть середовище (**Database**), у якому буде реалізовано базу. Зокрема, у прикладі на рис. 2.16 обрано MS SQL Server. Після зазначення усіх необхідних параметрів натисніть кнопку **Create New Schema** (Створити нову схему).

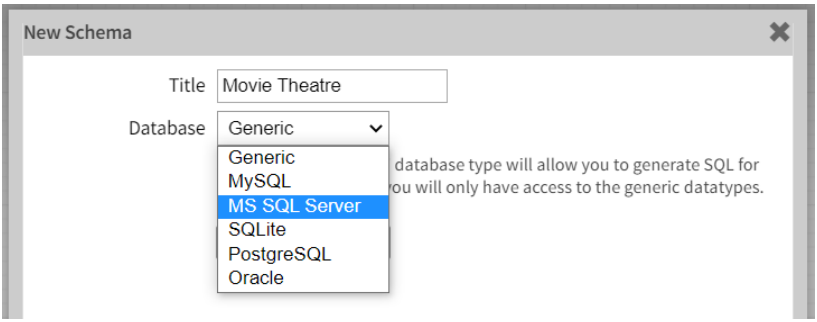


Рис. 2.16. Визначення характеристик нової схеми БД

5. *Створення нових сутностей на схемі БД.*

Щоб додати на схему нову сутність, тобто майбутню таблицю БД, клацніть правою кнопкою миші на білому полі та у контекстному меню, що з'явилося, оберіть пункт **Table** (Таблиця), як зображено на рис. 2.17.

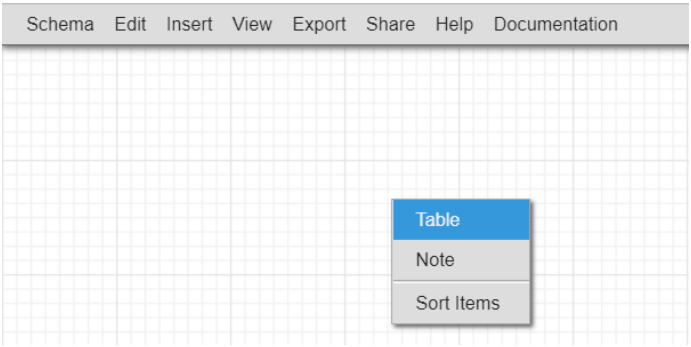


Рис. 2.17. Додавання нової таблиці (сутності) до схеми БД

Після цього з'являється умовне позначення майбутньої таблиці (рис. 2.18). Щоб змінити параметри сутності, зокрема, вказати її назву, клацніть по олівцю на сірому полі у правому верхньому куті позначення нової таблиці.

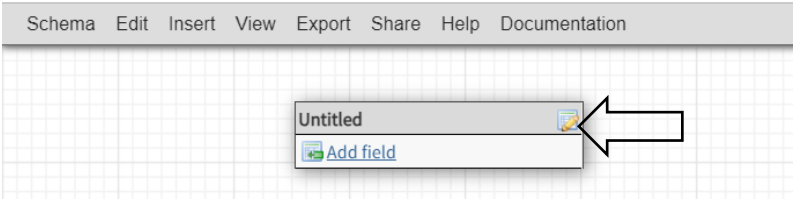


Рис. 2.18. Прототип нової таблиці

Назва сутності відповідає назві таблиці у БД. Після зміни натисніть кнопку **Save** (Зберегти). Доступні також опції **Cancel** (Скасувати) та **Delete Table** (Видалити таблицю) (рис. 2.19).

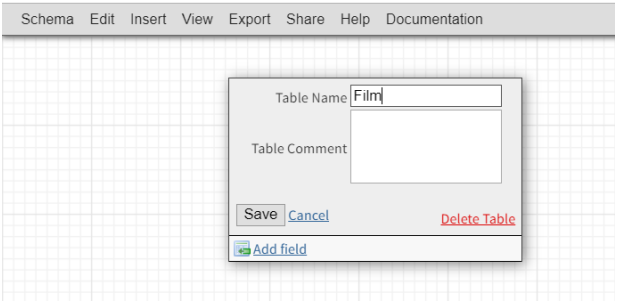


Рис. 2.19. Визначення назви нової таблиці БД

6. *Визначення атрибутів.*

Після створення сутності необхідно визначити атрибути, тобто поля майбутньої таблиці. Для цього потрібно натиснути на посилання **Add field** (Додати поле) (рис. 2.18). Вікно визначення параметрів атрибута зображено на рис. 2.20.

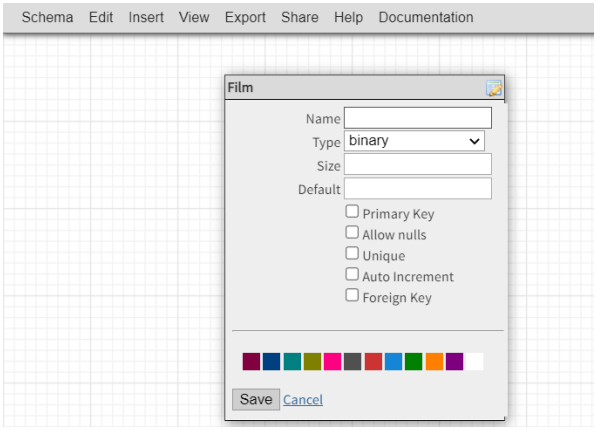


Рис. 2.20. Визначення атрибутів таблиці БД

Для кожного поля потрібно зазначити назву (**Name**) і тип (**Type**). Перелік характеристик, які можна обирати для полів, відповідає середовищу керування базами даних, обраному при створенні нової схеми.

Для атрибуту первинного ключа потрібно обрати опцію **Primary Key.** Якщо це поле – простий порядковий номер, що не

потребує внесення змістовних значень вручну, для нього можна обрати опцію **Auto increment** (Автозаповнення).

Щоб дозволити пропуски значень *для неключових полів* оберіть властивість **Allow nulls**. Решту опцій ми розглянемо у наступних лабораторних.

Для збереження параметрів поля потрібно натиснути кнопку **Save**. Перевіряйте та, за потреби, змінюйте характеристики полів, які зазначаються за замовчуванням.

7. Збереження розробленої схеми БД: меню **Schema**>**Save**.

Зауважте, що середовище DB Designer безкоштовне, проте вимагає попередньої реєстрації. За замовчуванням, можуть накладатися обмеження на кількість сутностей, що подаються на схемі, а також на кількість нових діаграм. Для розширення можливостей із метою навчання зі свого профілю можна надіслати запит, вказавши посилання на сайт навчального закладу, й отримати необмежений доступ на рік.

Звіт з лабораторної роботи повинен містити:

1. Опис варіанта завдання.
2. Таблицю 2.5 з описом сутностей та їх атрибутів *(за бажанням)*.
3. Екранну форму створеної у середовищі моделювання схеми БД із визначеним набором сутностей та їх атрибутів.
4. Відповіді на запитання для самоперевірки.

Для захисту лабораторної роботи необхідно подати звіт і бути готовими відповісти запитання з переліку нижче.

Запитання та завдання для самоперевірки

1. Яке призначення шаблонів подання даних у контексті оброблення інформації? Наведіть приклади шаблонів баз даних та переваги, які вони надають.
2. Яке призначення шаблонів подання даних у контексті оброблення інформації? Наведіть приклади шаблонів баз даних та переваги, які вони надають

3. Порівняйте властивості SQL (реляційних) та NoSQL баз даних. Які чинники слід брати до уваги при обранні того чи іншого типу системи.
4. Що таке ER-діаграма? Для чого вона створюється? З яких графічних елементів складається?
5. Які назви ER-діаграми вам відомі? Чому її потрібно розробляти, перш ніж БД буде реалізовано у СУБД?
6. У чому полягає об'єктно-орієнтоване проектування і з яких етапів складається процес розробки схеми БД?
7. Що таке сутність? Яку роль вона відіграє у моделі БД?
8. Які сутності називають конкретними, а які – абстрактними? Наведіть приклади сутностей кожного типу з вашої моделі БД.
9. Чим екземпляри сутності відрізняються від її атрибутів? Наведіть приклади екземплярів та атрибутів для двох довільних сутностей на вашій діаграмі.
10. У якій спосіб на ER-діаграмі подаються характеристики сутності? Як їх з'ясувати для об'єктів предметної області? Чи доцільно подати на схемі всі можливі характеристики об'єкта?
11. Використання яких атрибутів сутності слід уникати при розробленні концептуальної моделі БД?
12. Назвіть деякі сутності, які на вашу думку, були б типовими для таких ситуацій або систем:
    - Магазин із виготовлення меблів під замовлення
    - Онлайн-замовлення і доставка їжі
    - Прибирання і чергування у гуртожитку
    - Пошиття та ремонт одягу
    - Вирощування і постачання овочів
    - Будь-який інший варіант.

ТЕМА 3. Реляційна модель і нормалізація

Концептуальна модель зображає логічну структуру БД загалом. Вона дозволяє побачити виокремлені однотипні об'єкти предметної області та взаємодію між ними проте не дає можливості внести дані або виконати над ними певні операції. Можна сказати, що ER-діаграма – це проект, на основі якого буде фактично реалізовано базу даних. Якого ж вигляду набуватимуть дані при зберіганні їх у конкретній СУБД?

Реляційна модель даних

Найбільш поширеною моделлю сучасних баз даних вважається реляційна модель [6–8]. Розроблена Едгаром Коддом (Edgar Codd) у компанії IBM у 1970 році, **реляційна модель (РМ)** – це логічна модель даних, що описує структуру, в яку об'єднуються групи елементів, припустимі операції над даними та спеціальні правила, що забезпечують актуальний стан інформації. Назва моделі походить від англ. слова *relation* – відношення.

Реляційна модель базується **на трьох аспектах**:

Аспект структури моделі визначає, що єдиною структурою даних, що використовується у реляційних БД, є двовимірна таблиця, або, як її називають у РМ – нормалізоване n-арне відношення.

Аспект цілісності покликаний забезпечити унікальність кожного об'єкта або явища, про які зберігається інформація у базі, уникнути неконтрольованого дублювання та усунути суперечливості збережених даних. Ба більше, реляційна модель даних дозволяє описувати складні процеси за участі декількох класів об'єктів завдяки утворенню між ними зв'язків. До складу таких похідних відношень входять ключові атрибути об'єктів, що взаємодіють, а також атрибути, що характеризують цей зв'язок.

Аспект обробка реляційної моделі визначає операції, які можна виконувати над відношеннями, що ґрунтуються на реляційній алгебрі та реляційному численні, результатом яких також є відношення.

Фактично, **реляційна база даних** – це набір простих таблиць, між якими встановлено зв'язки за допомогою числових кодів.

У реляційній моделі БД були введені спеціальні терміни, порівняння яких з традиційними позначеннями та поняттями концептуальної моделі даних наведено у табл. 3.1 [1, 8, 11].

Таблиця 3.1

Порівняння термінів логічного проектування БД

Традиційне позначення	**Концептуальна модель**	**Реляційна модель**
Таблиця	Сутність	Відношення
Стовпець/поле	Атрибут	Атрибут
Рядок/запис	Екземпляр сутності	Кортеж
Множина логічно допустимих значень	Домен	Домен
Унікальний ідентифікатор	Первинний ключ	Первинний ключ/індекс
Кількість стовпців	-	Степінь/арність
Кількість рядків	-	Кардинальність
Заголовок таблиці	Набір атрибутів	Схема відношення

Відношення – це двовимірна таблиця, що складається зі стовпців і рядків. У реляційній моделі відношення використовуються для зберігання інформації про класи об’єктів (сутності), розміщені у БД. Кожен клас об'єктів має власну схему відношення, атрибути якої описують характеристики об’єкта.

Відношення РМ складається із заголовка і тіла (рис. 3.1).

Заголовок відношення – це множина пар *<Назва атрибута: Тип атрибута>*:

(<A1:T1>, <A2:T2>,… <An:Tn>),

де *Ai* – назва атрибута, *Ti* – тип атрибута, *n* – степінь (арність) відношення. У межах одного відношення кожен атрибут має унікальне ім’я.

Поряд із типом даних важливою характеристикою атрибута є ***домен***. Тип даних визначає формат фізичного зберігання даних, зокрема, розмір у байтах комірки, що відводиться під кожне значення, внутрішнє подання даних у пам'яті, діапазон усіх можливих значень поля. Водночас домен базується на типі даних, проте обмежує його до деякої множини логічно допустимих значень, властивих деякому атрибуту. За допомогою домена користувач може розкривати зміст атрибуту, а також джерело, звідки він одержує дані. Однотипні атрибути можуть бути визначені на різних доменах. Застосування доменів дозволяє уникнути введення невідповідних даних та виконання некоректних операцій. Наприклад, якщо для відношення, зображеного на рис. 3.1, ми замість назви книги введемо прізвище її автора, це не викликатиме помилки з боку типу даних, адже обидва поля символьного типу. Проте з точки зору логіки це некоректно, оскільки ці два атрибути визначені на різних доменах. Так само беззмістовно буде порівнювати рік випуску книги з кількістю сторінок у ній.

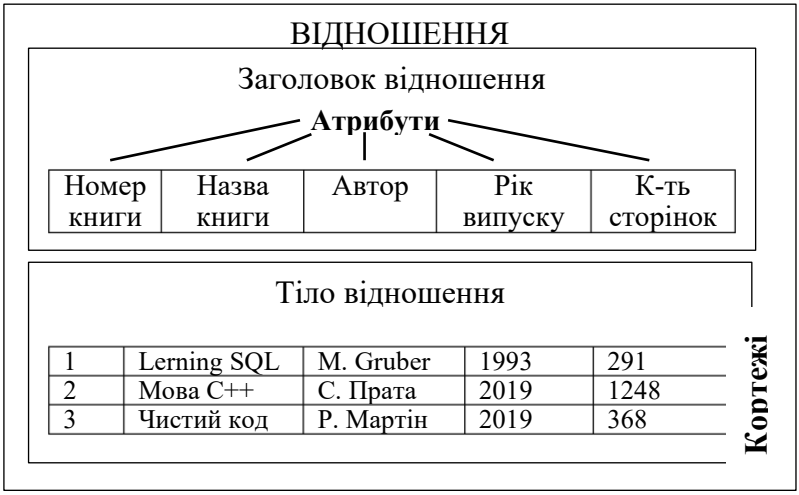


Рис. 3.1. Склад відношення реляційної моделі

Заголовок відношення називають ***схемою відношення***. При створенні таблиці визначається саме її схема, тобто склад об'єктів, про які необхідно зберігати інформацію у БД.

Тіло відношення складається з кортежів. Як елемент відношення, кортеж – це множина з *m* пар *<Назва атрибута: Значення атрибута>*:

(<A1:V1>, <A2:V2>,… <An:Vn>),

де *Vi* – значення, яке узгоджується за типом даних і доменом відповідного атрибута за схемою відношення, *m* – кардинальність.

Властивості відношень

Більша частина властивостей відношень БД походить від властивостей математичних множин.

1. У відношенні немає однакових атрибутів: схема відношення, як математична множина атрибутів, не може містити однакових елементів, а послідовність їх розташування у заголовку не відіграє ніякої ролі. Атрибути у схемі відношення відрізняються за назвою, а не за порядком розташування.
2. У множині не повинно бути повторюваних елементів. Аналогічно, тіло відношення, елементами якого є кортежі, не може містити рядків-дублікатів.
3. Кожен елемент кортежу повинен містити єдине й неподільне значення. Відношення, що містить атомарні значення атрибутів, називають нормалізованим.
4. Набір можливих значень для даної комірки відношення визначається множиною або доменом. У відношенні БД значення кожного атрибута повинні походити з того домену, на якому визначено атрибут.

Цілісність у реляційній моделі БД

Аспект цілісності реляційної моделі має на меті забезпечити однозначний і несуперечливий стан даних, що зберігаються у відношеннях. Для цього повинні виконуватися дві вимоги цілісності: цілісність сутностей та цілісність посилань.

Вимога цілісності сутностей: будь-який кортеж будь-якого відношення повинен відрізнятися від будь-якого іншого кортежу цього ж відношення.

Кортежі відношення, тобто рядки таблиці, можуть ідентифікуватися за тим, яких значень набувають їх атрибути. Оскільки кожен кортеж містить інформацію про окремий об'єкт або екземпляр сутності, він має відрізнятися від усіх інших елементів відношення. Проте може трапитися, що деякі рядки цілковито збігатимуться за значеннями своїх атрибутів, а отже ці об'єкти не вдасться розрізнити. Тому в будь-якому відношенні повинен бути окремий ключовий атрибут – **первинний ключ** (*Primary Key, PK*), значення якого однозначно та унікально ідентифікуватимуть кортежі відношень.

Складні сутності реального світу подаються у реляційній БД у вигляді кортежів кількох відношень.

Зовнішній ключ (*Foreign key, FK*) – це атрибут або множина атрибутів усередині відношення, що відповідають первинному ключу іншого відношення, і служать для утворення зв'язків типу «один-до-багатьох».

Зовнішній ключ не володіє властивістю унікальності та визначається на домені відповідного первинного ключа батьківського відношення.

Дочірнє відношення за значенням зовнішнього ключа посилається на батьківське відношення з метою одержання додаткової інформації про об'єкт, позначений цим ключем.

Вимога цілісності посилань, або **вимога зовнішнього ключа**: для кожного значення зовнішнього ключа дочірнього відношення, у батьківському відношенні, до якого веде посилання, завжди знайдеться кортеж з таким самим значенням первинного ключа.

Цілісність посилань забезпечує правильність введення даних, а завдяки зв'язку зміни ключа в батьківському відношенні можуть автоматично вноситися до дочірнього відношення.

Основи теорії нормалізації

У процесі проектування схему базу даних можна зобразити за допомогою різного набору відношень. Проте певний набір завжди буде володіти кращими властивостями при роботі з БД, ніж усі інші набори відношень, що позначають ті самі дані. Нормалізація бази даних – це добре відома техніка, що містить

низку рекомендацій для створення оптимальної моделі БД. Основна мета нормалізації полягає у зменшенні надлишковості та залежності даних [9, 11].

Шляхом декомпозиції нормалізація допомагає розбити великі таблиці на декілька менших і більш коретних, визначаючи логічний зв'язок між ними. Це послідовний процес зведення відношень до більш оптимального вигляду з використанням шести нормальних форм (НФ): **1 НФ, 2 НФ, 3НФ, НФ Бойса-Кодда, 4 НФ, 5НФ.**

Кожна наступна НФ зберігає властивості попередніх НФ та дещо їх поліпшує. Процедура нормалізації – зворотна: завжди можна повернутися до попередньої НФ, а отже, в процесі декомпозиції дані не втрачаються.

Умова зворотності перетворень вимагає збереження схем БД та попередніх залежностей між атрибутами. При виконанні операцій над нормалізованими відношеннями не повинні з’являтися раніше відсутні кортежі, тобто інформація про неіснуючі факти/об’єкти/ процеси.

Дотримання у базі даних принципів нормалізації має такі позитивні наслідки:

- скорочення витрат на зберігання дублікатів даних, а відповідно, зменшення розміру бази даних;
- зростанням обсягу даних у базі не позначається негативно на часі виконання запитів та SQL-операторів;
- відсутність аномалій модифікації, зокрема вставки, редагування та видалення даних.

Ненормалізоване відношення виглядає має вигляд багаторівневої таблиці, яка містить групи даних, що повторюються. Така ситуація виникає при спробі записати до однієї комірки таблиці більше одного значення. Ба більше, між атрибутами у схемі відношення можуть існувати додаткові залежності, що призводять до неконтрольованої надлишковості даних [9].

Необов’язково застосовувати на практиці всі 6 нормальних форм. За невеликої кількості атрибутів (стовпців) у базі даних достатньо забезпечити виконання умов перших трьох нормальних форм. Приклад ненормалізованого відношення наведено у табл. 3.2.

Таблиця 3.2

Навчання студентів

ID	Студент	Номер_залік.	Предмет	Викладач
1	Мороз В.	22341	Фізика Хімія Історія	Божко І.С. Власенко М.В. Ільків А.К.
2	Коваль І.	22678	Фізика Укр.мова	Божко І. С. Калужна Л.В.
3.	Ткач Р	23890	Хімія Право Біологія	Власенко М.В. Козьмик О.Г. Грінь Ф.І.

У табл. 3.2 можна вирізнити такі недоліки:

Використання неатомарних атрибутів: *Предмет* і *Викладач*. Для кожного екземпляра сутності ці атрибути набуватимуть цілої низки значень, редегування яких може викликати суттєві труднощі. Наприклад, як до такого переліку додати новий предмет, який відвідує студент, а також дані відповідного викладача – це аномалія редагування.

Як видалити деякий предмет із переліку – аномалія видалення.

Операції пошуку даних у таблиці: які предмети веде конкретний викладач – досить громіздкі, якщо взагалі можливі.

Наявність залежностей між атрибутами: *Студент* та *Номер залікової книжки*, достатньо обрати лише один з цих ідентифікаторів, аби не допустити введення некоректних даних. Адже неможливо перевірити, чи належить вказаний номер конкретному студентові, ба більше, коли ці дані повторюватимуться для зазначення інших предметів, які вивчатиме студент.

З'ясуємо, як усунути зазначені недоліки відношення та коротко розглянемо три найпоширеніші нормальні форми.

1НФ (перша нормальна форма)

Відношення нормалізоване, тобто перебуває в 1НФ, тоді, коли всі його атрибути прості (атомарні). У межах кортежу реляційного відношення значення будь-якого атрибута не може

бути складовим, тобто містити в одному записі кілька значень одного типу або визначених на різних доменах.

Таблиця 3.3

Нормалізоване відношення «Навчання студентів»

ID	Студент	Номер_залік.	Предмет	Викладач
1	Мороз В.	22341	Фізика	Божко І.С.
2	Мороз В.	22341	Хімія	Власенко М.В.
3	Мороз В.	22341	Історія	Ільків А.К.
4	Коваль І.	22678	Фізика	Божко І. С.
5	Коваль І.	22678	Укр.мова	Калужна Л.В.
6	Ткач Р	23890	Хімія	Власенко М.В.
7	Ткач Р	23890	Біологія	Грінь Ф.І
8	Ткач Р	23890	Право	Козьмик О.Г.

Наведене у табл. 3.2 відношення ненормалізоване. Для того, щоб звести його до 1НФ, потрібно доповнити дані про викладання предметів відомостями про студента, і кожен запис унікально ідентифікувати (табл. 3.3). Попри усунення неатомарних атрибутів, оновлене відношення потребує подальшого вдосконалення, адже в ньому наявна велика кількість повторюваних значень.

Іншим прикладом ненормалізованого відношення може бути таблиця, один або декілька атрибутів якої містять групи різнотипних значень. Розглянемо відношення Товари (табл. 3.4).

Таблиця 3.4

Товари

Код	Назва	Характеристики	Ціна
245	Варення	склянка, 0,3 л, скло	35,78
307	Кефір	пакет, 0,5 л, плівка	28,50
421	Цукор	коробка, 850 г, картон	34,69
166	Йогурт	пляшка, 0,9 л, пластик	42,30
583	Гречка	пакет, 1 кг, поліетилен	54,20
233	Молоко	пакет, 0,75 л, плівка	30,00

У таблиці Товари (табл. 3.4) складовий атрибут *Характеристики* поєднує в собі чотири властивості товару: тип

упаковки, об'єм/вага, одиниці вимірювання та матеріал упаковки. Для читання ця інформація цілком зрозуміла, проте з точки зору її зберігання у базі даних, оброблення та модифікація засобами мови SQL викликають суттєві складнощі. Спростити ситуацію допоможе розбиття складового атрибута на низку логічних компонентів та зберігання кожної характеристики окремо (табл. 3.5).

Таблиця 3.5

Нормалізоване відношення Товари

Код	Назва	Упаковка	Об'єм	Одиниці	Матеріал	Ціна
245	Варення	склянка	0,3	л	скло	35,78
307	Кефір	пакет	0,5	л	плівка	28,50
421	Цукор	коробка	850	г	пластик	34,69
166	Йогурт	пляшка	0,9	л	поліетил.	42,30
583	Гречка	пакет	1	кг	плівка	54,20
233	Молоко	пакет	0,75	л	плівка	30,00

2НФ (друга нормальна форма)

Її ще називають нормальною формою складового первинного ключа. Відношення перебуває у 2НФ, якщо воно відповідає вимогам 1НФ і кожен його неключовий атрибут функціонально повно залежить від усього первинного ключа і не залежить від будь-якого його компонента.

Функціональна залежність (ФЗ) – це зв'язок, що може виникнути між характеристиками, що зберігаються в базі даних. Якщо у відношенні R(*A, B, C*) деякий атрибут *A* функціонально визначає атрибут *B*, то таку залежність позначають у такий спосіб:

$$\boldsymbol{A \rightarrow B},$$

де *A* – детермінант, а *B* – залежна частина.

Зокрема, у відношенні Навчання студентів (табл. 3.3) існує функціональна залежність *Студента* від *Номера залікової книжки*, адже за номером залікової ми можемо з'ясувати дані про самих студентів. У нашому прикладі – це повне ім'я, проте можуть бути й інші відомості, як-от номер групи, спеціальність, рік вступу, рівень навчання та ін. Така функціональна залежність графічно позначається як:

Номер залікової →Студент

Функціональні залежності між окремими атрибутами зазвичай виникають через те, що в одній таблиці поєднано декілька об'єктів або процесів.

Якщо таблиця містить складений (композитний) ключ і має у своєму складі часткові залежності, можна застосувати такі способи нормалізації:

1) Ввести додаткове поле-лічильник, яке виконуватиме функцію первинного ключа, міститиме унікальні значення, які ідентифікуватимуть кожен рядок таблиці. Оскільки первинний ключ простий, в ньому немає часткових залежностей і з будь-яким іншим атрибутом таблиці встановлюється однозначна відповідність. Власне, табл. 3.3 демонструє застосування такого підходу. Проте додавання окремого ключа не запобігає появі аномалій.

2) Розбити вихідне відношення на декілька логічних таблиць так, щоб ключові атрибути (ключові поля) розійшлись по різних відношеннях. Цей спосіб також може бути використаний у вищих нормальних формах.

Таблиця 3.6

Облік співробітників

Номер_працівника	Посада	Оклад
152	Інженер 1-ї категорії	35000
248	Інженер 1-ї категорії	35000
678	Механік	33000
109	Інженер-програміст	46000
351	Інженер-програміст	46000
429	Механік	33000
152	Інженер-програміст	46000

Відношення, наведене у табл. 3.6, зображає ситуацію, коли працівник, що має свій унікальний номер, може обіймати декілька посад. Тому поєднання значень цих двох атрибутів буде унікальним і виконуватиме роль складового первинного ключа. За визначенням, неключове поле *Оклад* має визначатися за значенням ключа. Проте цього не відбувається, оскільки цей показник залежить лише від посади, і жодним чином не від

номера працівника. Тому в цьому відношенні наявні такі функціональні залежності:

(Номер_працівника, Посада) $\rightarrow$ ***Оклад***

Посада $\rightarrow$ ***Оклад***

Окрім того, що має місце повторюваність однакових значень, виникають аномалії. Зокрема, якщо з'являється нова посада, то дані про неї неможливо додати до таблиці, допоки хтось із робітників не почне її обіймати (а залишати поле порожнім не можна, оскільки воно входить до складу первинного ключа). Якщо для посади змінюється сума окладу, потрібно внести зміни до всіх рядків, де містяться дані про неї.

Такі аномалії стали наслідками поєднання в одному відношенні двох різних сутностей: Перебування на посаді та Посадовий оклад.

Для декомпозиції вихідного відношення слід дотримуватися таких рекомендацій:

Вихідне відношення: R*(K1, K2, A1, …, An, B1,…,Bn*)

- Ключ: (*K1, K2)* – складовий.

Функціональні залежності:

- *(K1, K2) ->(A1, …, An, B1,…,Bn*) – залежність усіх атрибутів від ключа відношення.
- *K1 ->(A1, …, An*) – залежність деяких атрибутів від частини складового ключа.

Декомпозиція відношення:

- R1*(K1, K2, B1,…,Bn*) – залишок від початкового відношення. Ключ *(K1, K2*). (К1 – Номер_працівника, К2 – Посада).
- R2*(K1, A1, …, An*) – залежні атрибути, виокремлені разом з частиною складового ключа. Ключ *K1*. (*К1 – Посада*, *А – Зарплата*).

У результаті нормалізації одержимо два пов'язаних між собою відношення, до яких додамо ще деякі атрибути, зокрема первинні ключі (рис. 3.2).

Посади

ID_посади	Посада	Оклад
1	Інженер 1-ї категорії	35000
2	Механік	33000
3	Інженер-програміст	46000

Перебування на посаді

ID_роботи	Номер_працівника	ID_посади	Ставка
1	152	1	0,5
2	248	1	1,25
3	678	2	0,75
4	109	3	1,0
5	351	3	0,75
6	429	2	1,5
7	152	3	0,5

Рис. 3.2. Декомпозиція відношення Облік співробітників

Унаслідок два відокремлені відношення володіють кращими властивостями з точки зору виконання операцій обробки, ніж вихідна табл. 3.5. Тепер зміни параметрів посади торкатимуться одного рядка однойменної таблиці та не впливатимуть на Перебування на посаді, завдяки ідентифікації за первинним ключем.

3НФ (третя нормальна форма)

Відношення перебуває у 3НФ, якщо воно задовольняє вимогам 2НФ і в ньому відсутні транзитивні залежності. Зокрема, кожен кортеж складається зі значення первинного ключа, який ідентифікує певний екземпляр сутності, і набору незалежних атрибутів, які описують цей екземпляр.

Якщо у відношенні R(*X, Y, Z*) між атрибутами існують такі функціональні залежності: $\boldsymbol{X \rightarrow Y}$ та $\boldsymbol{Y \rightarrow Z}$, а ФЗ у протилежному напрямку відсутні, тоді *Z* **транзитивно залежить** від *X*.

Наприклад, у таблиці 3.7 зберігаються поштові дані доставки користувачів.

Таблиця 3.7

Поштові дані клієнтів

КлієнтID	ZIPклієнта	Місто
12345	58000	Чернівці
67124	2105	Вінниця
43908	03150	Київ
55608	73000	Херсон
11207	69123	Запоріжжя

У наведеному вище випадку стовпець *Місто* залежить від поштового індексу (*ZIP*) клієнта, який зі свого боку залежить від *Номера_клієнта* як від первинного ключа:

КлієнтID→ZIPклієнта

ZIPклієнта→Місто

Такий сценарій може мати негативні наслідки. Наприклад, коли в поле індексу вносять значення, не виконується перевірка, чи відповідає воно вказаному місту, а при оновленні індексу, місто автоматично не змінюється, а отже, порушується несуперечливий стан даних.

Для усунення транзитивної залежності слід вдатися до декомпозиції із дотриманням таких рекомендацій:

Вихідне відношення: R*(K1, A1, …, An, B1,…,Bn*).

- Ключ: *K1*.

Функціональні залежності:

- *K1->(A1,…,An, B1,…,Bn*) – залежність усіх атрибутів від ключа.
- (*A1, …, An)->(B1,…,Bn*) – залежність одних неключових атрибутів від інших.

Декомпозиція відношень:

- R1*(K1, A1, …, An)* – залишок вихідного відношення. Ключ: *K1*, (*КлієнтID*, *ZIPклієнта*).
- R2*(A1, …, An, B1,…,Bn*) – пов’язані атрибути, виокремлені з вихідного відношення разом з детермінантом функціональної залежності. Ключ і детермінант (*A1,… An*), (*ZIPКлієнта*, *Місто*).

Два отримані відношення зображені на рис. 3.3.

КлієнтID	ZIPклієнта
12345	58000
67124	2105
43908	03150
55608	73000
11207	69123

ZIPклієнта	Місто
58000	Чернівці
2105	Вінниця
03150	Київ
73000	Херсон
69123	Запоріжжя

Рис. 3.3. Зведені до 3НФ відношення з даними для доставки

Завдяки декомпозиції відношення, зображеного в табл. 3.7, нам вдалося роз'єднати дві окремі сутності: поштові дані міст і дані клієнтів, пов'язавши їх через поштовий індекс. *Порівняйте, як у такій системі виконуватимуться операції, які викликали суперечливості на попередньому кроці.*

Узагальнено властивості всіх нормальних форм наведено у табл. 3.8.

Таблиця 3.8

Властивості нормальних форм реляційних відношень

НФ	Короткий опис
1НФ	Всі атрибути відношення атомарні та прості
2НФ	Наявність складового первинного ключа і залежність від нього всіх неключових атрибутів
3НФ	Неключові атрибути незалежні між собою (відсутні транзитивні залежності)
НФБК	Складовий первинний ключ не залежить від неключових атрибутів
4НФ	Усунення багатозначної залежності
5НФ	Відсутність залежності з'єднання та уникнення появи хибних рядків при зворотному з'єднанні відокремлених відношень через операцію з'єднання (моделювання зв'язків типу «багато-то-багатьох».

ЛАБОРАТОРНА РОБОТА № 3
Оптимізація концептуальної моделі бази даних

Мета роботи

Змоделювати взаємодію між об'єктами предметної області та оптимізувати концептуальну модель бази даних.

Завдання

У середовищі проектування бази даних створити зв'язки між сутностями, згідно з принципами теорії нормалізації усунути багатозначні залежності, завершити розробку ER-діаграми.

Теоретичний матеріал

Взаємодію об'єктів предметної області у концептуальній моделі позначають за допомогою зв'язків. Зв'язки утворюються між сутностями, у яких є принаймні один спільний атрибут.

Існують такі типи зв'язків між сутностями:

❖ **Один-до-одного (1:1)**: один екземпляр першої сутності пов'язаний точно з одним екземпляром другої сутності. Такий зв'язок найчастіше свідчить про те, що одну сутність надмірно розділили на дві.

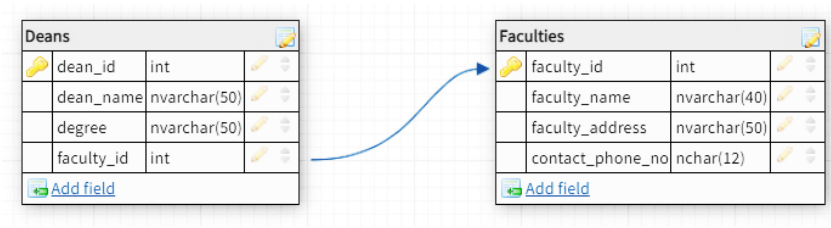


Рис. 3.4. Приклад утворення зв'язку типу «один-до-одного»

У прикладі на рис. 3.4 для збереження поточної інформації у межах університету між сутностями Декани (Deans) та Факультети (Faculties) можна встановити зв'язок цього типу за спільним полем **faculty_id**, оскільки на одному факультеті може бути лише один декан.

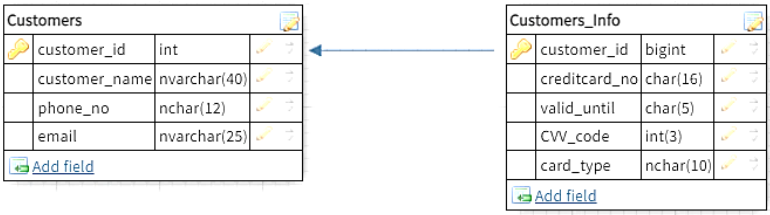


Рис. 3.5. Відокремлення двох різних типів інформації про один об’єкт за допомогою зв’язку «один-до-одного»

Подекуди зв’язок типу «один-до-одного» може використовуватися для розбиття на частини інформації про один і той самий об’єкт. Як бачимо на рис. 3.5, дані про клієнтів інтернет-магазину, розподілені між двома об’єктами: в сутності Customers міститься загальна контактна інформація про клієнтів, а сутність Customers_info призначена для зберігання даних про банківську картку, яку клієнт зазначив при оформленні замовлення. Такий підхід може бути доцільним для розмежування загальнодоступної та конфіденційної інформації про клієнтів, а також надання доступу до важливих даних персоналу з визначеними привілеями.

❖ **Один-до-багатьох (1:M):** при утворенні зв’язку цього типу кожен екземпляр однієї сутності, яку називають батьківською, пов’язаний із довільною кількістю (0 і більше) екземплярів іншої, похідної (дочірньої) сутності. Зі свого боку, кожен екземпляр похідної сутності пов’язаний точно з одним екземпляром батьківської сутності. Отже, екземпляр сутності-нащадка може існувати лише за умови наявності батьківської сутності. Це найбільш поширений тип зв’язку, який зазвичай утворюється за полем первинного ключа батьківської сутності, яке у дочірній сутності набуває статусу **зовнішнього ключа**.

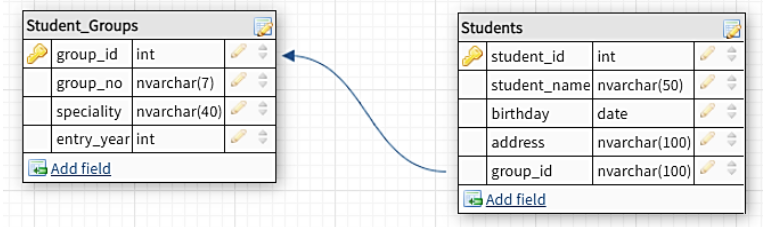


Рис. 3.6. Приклад утворення зв’язку типу «один-до-багатьох»

У прикладі, зображеному на рис. 3.6, між сутностями Студентські групи (Student_Groups) та Студенти (Students) встановлено зв'язок за спільним атрибутом «номер групи» (**group_id**). Сутність Студентські групи – батьківська, адже для неї це атрибут первинного ключа, що унікально позначає кожну групу, створену в університеті. Сутність Студенти – дочірня із зовнішнім ключем «номер групи», який для студентів, що навчаються в одній групі, набуватиме однакових значень. За цим номером завжди можна звернутися до батьківської сутності, аби дізнатися більше інформації про групу із заданим номером. До того ж завдяки цьому зв'язку неможливо буде для студента вказати номер групи, якої не існує у батьківській таблиці.

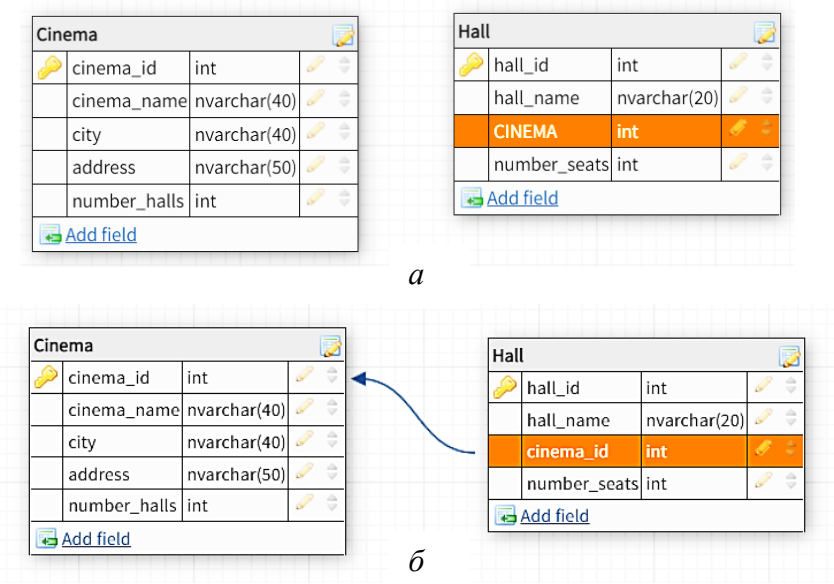


Рис. 3.7. Утворення зв'язку типу «один-до-багатьох» для предметної області Мережа кінотеатрів через спільний атрибут

На схемі БД, попередньо розробленій для інформаційної системи Мережі кінотеатрів (рис. 2.1), у сутності Зал (Hall) серед атрибутів зазначено узагальнений атрибут CINEMA, що позначає кінотеатр, у якому цей зал функціонує (рис. 3.7, *а*). Завдяки існуванню батьківської сутності Cinema, що міститиме дані про

всі кінотеатри у мережі, до похідної сутності та майбутньої таблиці БД Hall можна перенести не всі атрибути, властиві кінотеатрам, а лише відповідний порядковий номер (**cinema_id**), під яким усі ці дані ховатимуться. Для залів цей атрибут втратить обов'язкову властивість унікальності та стане зовнішнім ключем, через який утвориться зв'язок між двома сутностями (рис. 3.7, *б*).

❖ **Багато-до-багатьох (M:N):** кожен екземпляр однієї сутності може бути пов'язаний із кількома екземплярами іншої сутності і навпаки. Цей тимчасовий тип зв'язку може використовуватися лише на початкових стадіях розробки моделі. **Надалі зв'язок «багато-до-багатьох» потрібно замінити двома зв'язками типу «один-до-багатьох» шляхом створення проміжної сутності**.

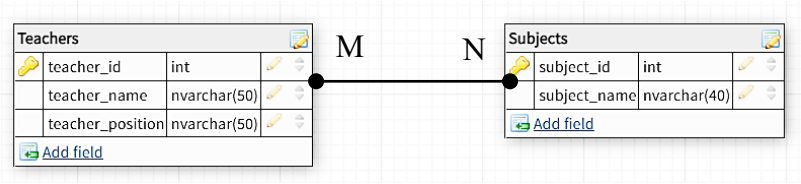


Рис. 3.8. Приклад взаємодії типу «багато-до-багатьох»

На рис. 3.8 зображено приклад утворення зв'язку типу «багато-до-багатьох» між сутностями Викладачі (Teachers) та Навчальні дисципліни (Subjects): один викладач може викладати кілька предметів, а один предмет може різними викладачами, які ведуть лекції, лабораторні, практичні заняття або викладають на різних спеціальностях. Причому кількість предметів (N) не дорівнює кількості викладачів (M). У цих сутностей немає спільного атрибутау, а отже поки що неможливо зберегти інформацію про те, хто яку дисципліну забезпечує.

Оптимізація моделі бази даних є комплексним логічним процесом, який потребує глибокого знання предметної області та методів аналізу взаємодії між її об'єктами.

До оптимізації слід вдаватися в нижченаведених випадках.

- У предметній області між сутностями наявні зв'язки типу «багато-до-багатьох», які зводяться до зв'язку «один-до-

багатьох» з одночасним створенням необхідних сутностей-зв'язок (рис. 3.9).

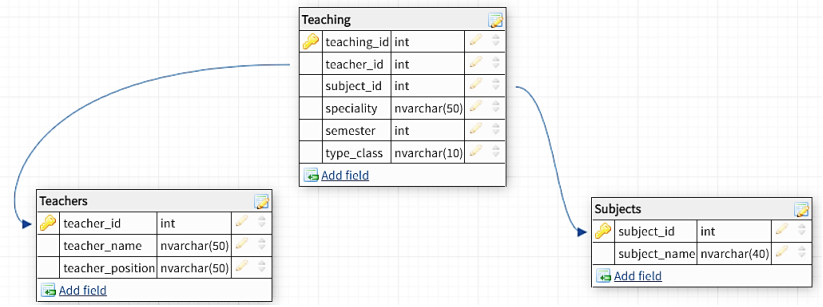


Рис. 3.9. Оптимізація зв'язку «багато-до-багатьох»

Впровадження проміжної сутності Teaching (рис. 3.9) дає змогу описати процес викладання різних предметів викладачами. Зокрема, у цій сутності зазначається номер дисципліни (**subject_id**), за яку відповідає певний викладач (**teacher_id**), спеціальність (**speciality**), для якої предмет викладається, та навчальний семестр (**semester**). До того ж окремо вказано тип занять (**type_class**), як-от лекційні, лабораторні або практичні, які проводить викладач у рамках конкретної дисципліни.

- Наявність складних, неатомарних атрибутів, які для деяких екземплярів сутності можуть набувати одночасно кілька значень. Суперечливий атрибут потрібно ввести до складу самостійної сутності та пов'язати з вихідним об'єктом.

Наприклад, на рис. 3.10, *а*, зображено сутність Навчальні дисципліни (Subjects), одним із атрибутів якої є теми (**topics**), передбачені програмою курсу. Зазвичай одна дисципліна охоплює низку окремих тем. До того ж тема є радше самостійним об'єктом зі своїми характеристиками, ніж просто властивістю навчальної дисципліни.

Для розв'язання проблеми складового атрибута створюємо окрему сутність Теми (Topics), яка призначена для зберігання тем по ВСІХ предметах. Кожна тема характеризується унікальним номером (**topic_id**), назвою та номером предмета (**subject_id**), до якого вона належить (рис. 3.10, *б*).

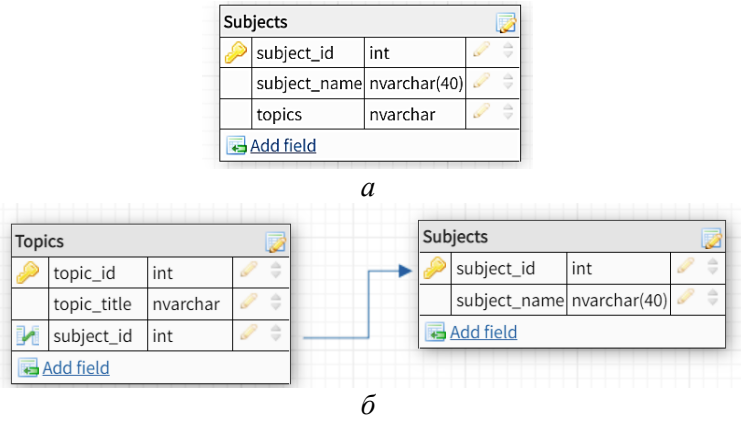


Рис. 3.10. Приклад усунення неатомарного атрибута **topics** у сутності Subjects

- Якщо атрибут – складовий, володіє власними характеристиками та для різних екземплярів сутності набуватиме одних і тих самих значень із визначеного переліку, його слід виокремити в батьківську сутність і поєднати через зв'язок «один-до-одного» з вихідною сутністю.

У прикладі на рис. 3.6 серед атрибутів студентської групи зазначена спеціальність. Цей атрибут може набувати фіксованих значень, передбачених переліком спеціальностей (комп'ютерна інженерія, інженерія програмного забезпечення, комп'ютерні науки, кібербезпека тощо). Для груп, що здобувають освіту за одним напрямом, назва спеціальності буде повторюватися. Ба більше, спеціальність характеризується своїм набором властивостей, як-от номер спеціальності та галузь знань, а отже, може подаватися на схемі як самостійний об'єкт.

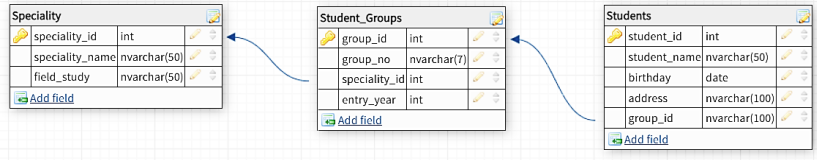


Рис. 3.11. Приклад перетворення складового атрибута на самостійну сутність

Як показано на рис. 3.11 складовий атрибут «спеціальність» (**speciality**) сутності Student_Groups (рис. 3.6) відокремлено для створення об'єкта Speciality з подальшим утворенням зв'язку між ними. До похідної сутності Student_Groups переноситься лише номер спеціальності (**speciality_id**), що стає у ній зовнішнім ключем. За цим номером завдяки утвореному зв'язку завжди можна довідатися детальну інформацію про конкретну спеціальність. Отже, це позбавляє нас необхідності переміщувати зайву інформацію до дочірньої сутності.

- Логічна оптимізація, коли необхідно скорегувати модель бази даних згідно з функціями і задачами, які необхідно розв'язувати у системі, що розробляється.

ПРАКТИЧНЕ ЗАВДАННЯ

Виконання оптимізації моделі бази даних, розробленої у попередній лабораторній, зводиться до такої послідовності кроків:

1. Визначення типів зв'язків між сутностями.
2. Створення зовнішніх ключів і зв'язків типу «один-до-багатьох».
3. Технічна оптимізація зв'язків типу «багато-до-багатьох».
4. Виокремлення складових неатомарних атрибутів у самостійні сутності.
5. Логічна оптимізація сутностей (у разі потреби).

1. Визначення зв'язків між сутностями

На основі попереднього набору сутностей, створеного у лабораторній роботі № 2 (рис. 2.14), можна виокремити типи взаємодії у вигляді таблиці, поданої на рис. 3.12.

Як бачимо, діаграма не містить зв'язків типу «багато-до-багатьох». Зокрема, за таким принципом безпосередньо взаємодіють сутності Фільми і Кінотеатри, адже один фільм може демонструватися в кількох кінотеатрах, а один кінотеатр може одночасно проводити покази різних фільмів. Цей тип зв'язку вдалося усунути завдяки впровадженню проміжних сутностей Показ фільму та Сеанс.

Фільм (Film)					1:1(M)	
Кінотеатр (Cinema)			1:M			
Жанр (Genre)	1:M					
Зал (Hall)						1:M
Показ фільмів (Film_Release)						1:M
Сеанс (Film_Show)						
	Фільм (Film)	Кінотеатр (Cinema)	Зал (Hall)	Жанр (Genre)	Показ фільмів (Film_Release)	Сеанс (Film_Show)

Рис. 3.12. Типи зв'язків на схемі бази даних «Мережа кінотеатрів»

У предметній області «Мережі кінотеатрів» зв'язки **«один-до-багатьох» (1:M)**, утворені між такими сутностями:

- **Фільм** – **Показ фільму**. Моделює розклад виходу в прокат кінофільмів. Фактично, це зв'язок «один-до-одного», проте він допомагає змоделювати ситуацію, коли один фільм може демонструватися кілька разів. Фільм – батьківська сутність, Показ фільму – дочірня.

- **Кінотеатр** – **Зал**. В одному кінотеатрі може бути кілька залів для кінопоказу, базова (батьківська) сутність – Кінотеатр, властивості якої буде використовувати похідна сутність Зал.

- **Жанр** – **Фільм**. Батьківська сутність Жанр при перетворенні на таблицю міститиме перелік усіх можливих жанрів кіно. Різні кінофільми можуть створюватися в одному жанрі, тому сутність Фільм буде похідною.

- Зал – **Сеанс**. В одному залі можуть відбуватися різні сеанси за визначеним розкладом. Проте неможливо проводити декілька сеансів в одному залі одночасно. Батьківська сутність – Зал.

- Показ Фільму – **Сеанс**. Один фільм може демонструватися декілька разів на різних сеансах, проте один сеанс не може призначатися для показу декількох кінострічок одночасно. Батьківська сутність – Показ Фільму.

2. Створення зовнішніх ключів і зв'язків типу «один-до-багатьох» на ER-діаграмі

Між створеними сутностями слід встановити зв'язки, як зазначено на рис. 3.12.

Розглянемо встановлення зв'язку типу «один-до-багатьох» на прикладі двох сутностей: Жанр (батьківська) і Фільм (дочірня). Для утворення зв'язку ці сутності повинні мати спільний атрибут. Зокрема, фільм позначатиметься жанром, до якого він належить. Оскільки всі можливі жанри ідентифіковані у батьківській сутності, до дочірньої ми можемо перенести номер жанру (**genre_id**). На попередньо створеній схемі у середовищі DB Designer для сутності Film потрібно змінити властивості узагальненого атрибута **GENRE**. Для цього клацаємо на зображенні олівця у відповідному полі (рис. 3.13).

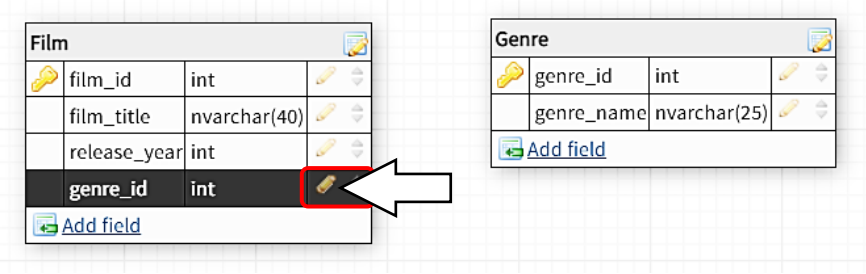


Рис. 3.13. Редагування властивостей атрибута на ER-діаграмі у середовищі DB Designer

Далі відкривається вікно редагування параметрів атрибута (рис. 3.14). Тут можна змінити назву зовнішнього ключа (**Name**). Рекомендовано, щоб ім'я починалося з назви батьківської таблиці, містило нижнє підкреслення та id. Тип (**Type**) цього

атрибута повинен відповідати параметрам спільного поля батьківської таблиці. Для утворення зв'язку зробимо атрибут зовнішнім ключем, позначивши опцію **Foreign Key**. На нижній панелі, що з'явилася, зі спадного списку потрібно вибрати назву батьківської таблиці (**Ref.Table**) – Genre та назву поля (**Ref. Field**) - **genre_id**, до якого веде зв'язок. Для застосування змін потрібно натиснути кнопку **Update**, для скасування змін – **Cancel.** Щоб видалити попередньо утворений зв'язок у дочірній таблиці для атрибута зовнішнього ключа слід прибрати позначку з опції **Foreign Key**.

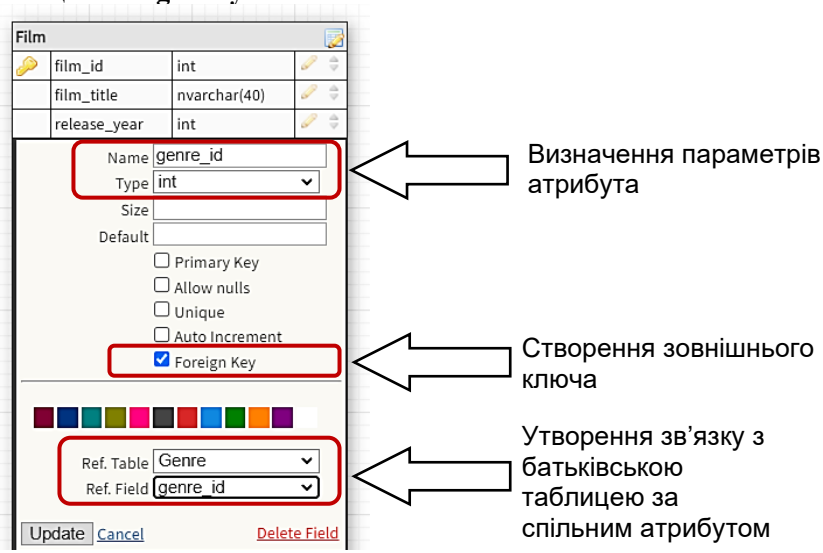


Рис. 3.14. Визначення параметрів атрибута зовнішнього ключа

У результаті виконаних налаштувань між двома сутностями утворюється зв'язок. Він позначається стрілкою, яка веде від поля зовнішнього ключа дочірньої таблиці Film і вістрям вказує на поле первинного ключа батьківської таблиці Genre (рис. 3.15).

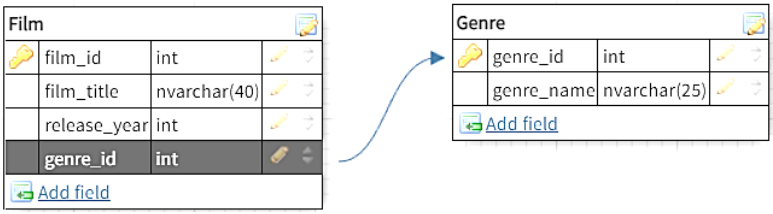


Рис. 3.15. Утворення зв'язку типу «один-до-багатьох»

Такі самі дії потрібно виконати для інших пар сутностей, пов'язаних зв'язком «один-до-багатьох». При належному проектуванні всі сутності на ER-діаграмі повинні бути з'єднані. В результаті, для Мережі кінотеатрів одержується схема БД, зображена на рис. 3.16.

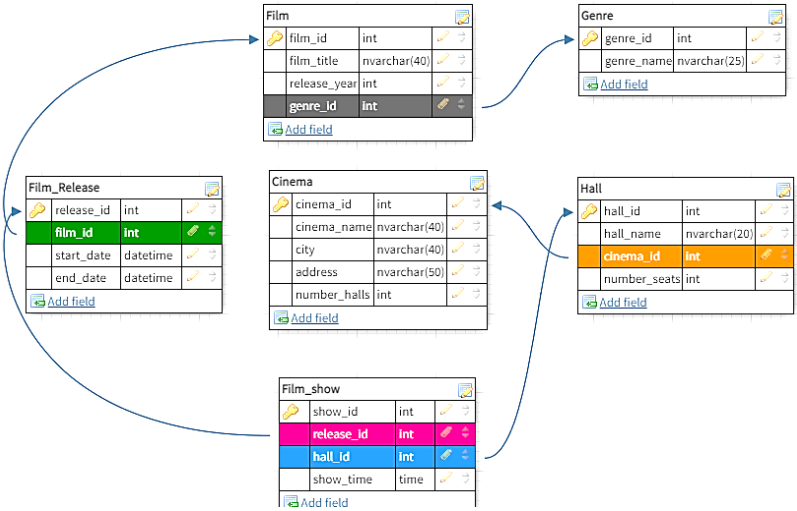


Рис. 3.16. Схема БД Мережі кінотеатрів з утвореними зв'язками

3. Оптимізація зв'язків типу «багато-до-багатьох»

Зв'язки типу «багато-до-багатьох» небезпечні, адже вони викликають суперечливості та неоднозначне тлумачення інформації, що зберігається у з'єднаних таблицях, не відображають реальний стан речей та можуть викликати появу

невідповідних даних при сумісному використанні майбутніх таблиць у запитах.

Для усунення зв'язків цього типу вводять ***проміжні сутності***.

Аби розглянути приклад оптимізації у нашій базі даних, припустимо, що для кожного фільму потрібно зберігати інформацію про акторів, що в ньому грають. Для цього введемо ще одну сутність Актори. Ця сутність взаємодіє з Фільмами за принципом «багато-до-багатьох» (рис. 3.17), адже один актор може грати у різних фільмах, а до одного фільму залучено кількох акторів.

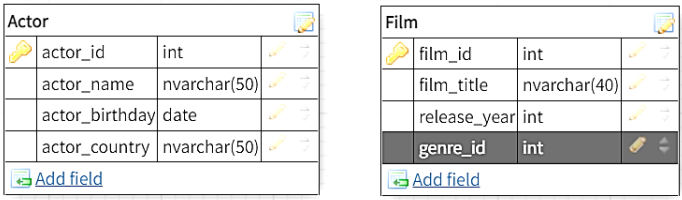


Рис. 3.17. Приклад сутностей, що взаємодіють за принципом «багато-до-багатьох»

4. Виокремлення складових атрибутів у самостійні сутності

Кожному представникові акторського складу можна надавати умовний порядковий номер, під яким зберігатиметься повне ім'я актора, дата та країна народження та будь-яка інша корисна інформація. Звісно, що найважливіша характеристика акторів – це фільми, у яких вони грали. Зазвичай це ціла низка кінострічок. З іншого боку, до атрибутів сутності Фільми можна було б додати акторів, що беруть у ньому участь. Тут так само маємо справу зі складовим неатомарним атрибутом. За логікою взаємодії акторів та фільми об'єднують ролі. Тому, аби усунути проблеми з комплексним атрибутом, оптимізувати зв'язок «багато-до-багатьох» та зіставити акторів з відповідними фільмами, створимо між сутностями Актор і Фільм проміжну сутність Роль (рис. 3.18).

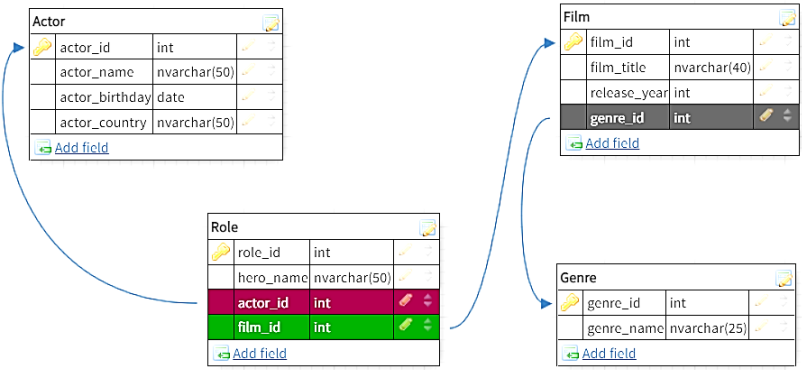


Рис. 3.18. Створення проміжної сутності для усунення зв'язку типу «багато-до-багатьох»

Нова сутність Роль під унікальним номером (**role_id**) зберігатиме інформацію про кожну роль в історії кінематографа, із зазначенням імені кіногероя (**hero_name**) та актора (**actor_id**), що його втілив у фільмі (**film_id**). При цьому актори, що грають в одному фільмі, позначатимуться однаковим номером фільму. А за номером актора можна буде легко визначити фільми, у яких вони грали.

Остаточна ER-діаграма для Мережі кінотеатрів зображена на рис. 3.19.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. *Для сутностей, створених у л/р № 2, визначте типи зв'язків та встановіть з'єднання типу «один-до-багатьох». Налаштуйте параметри зовнішніх ключів.*

2. *Шляхом впровадження проміжних сутностей оптимізуйте зв'язки типу «багато-до-багатьох».*

3. *Складні атрибути, що можуть виступати як самостійні об'єкти, або зв'язки, характеристики яких потрібно окремо зберігати чи обробляти, слід виокремити у додаткові сутності.*

4. *Переконайтесь, що в результаті оптимізації між усіма сутностями на схемі БД встановлені зв'язки типу «один-до-багатьох».*

5. *Збережіть розроблену логічну модель бази даних.*

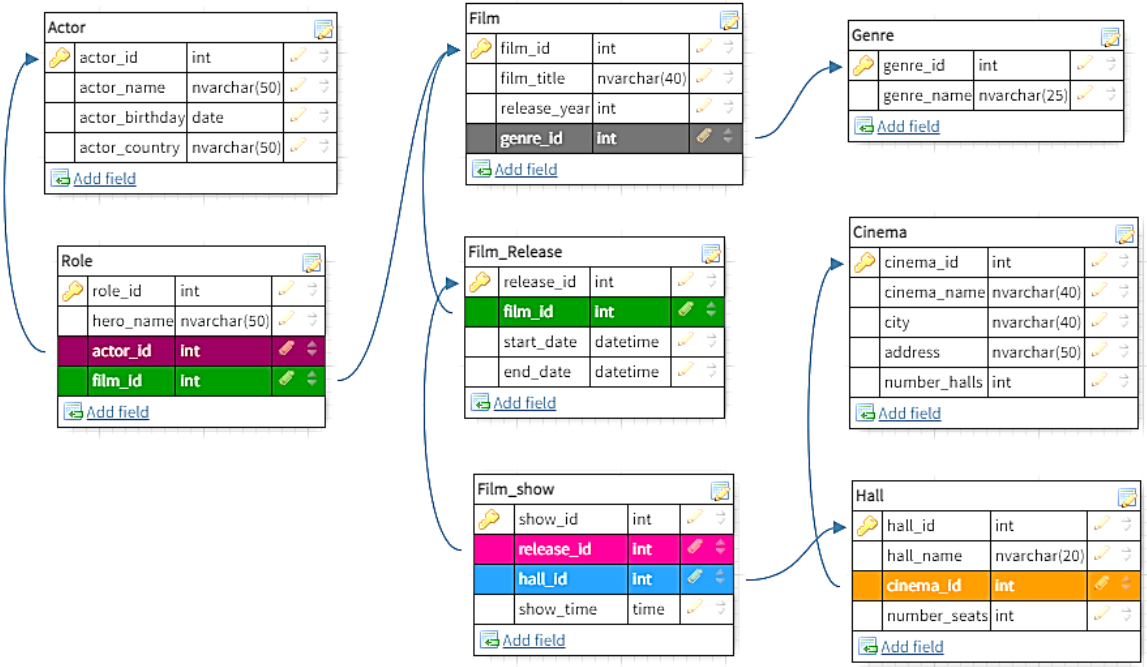


Рис. 3.19. Оптимізована концептуальна модель БД для предметної області «Мережа кінотеатрів»

Звіт з лабораторної роботи повинен містити:

1. Кінцеву логічну модель бази даних.
2. Відповіді на контрольні запитання.

Для захисту лабораторної роботи необхідно подати звіт та бути готовими відповісти на запитання для самрперевірки.

Запитання та завдання для самоперевірки

1. Для чого використовується реляційна модель даних і які її основні аспекти?
2. Що таке відношення? З чого воно складається? Що таке схема відношення?
3. Назвіть і поясніть властивості відношень.
4. Виконайте порівняльну характеристику типу даних і домену. Наведіть можливі приклади доменів для вашої моделі БД.
5. Що таке цілісність даних? Поясніть як вона забезпечується у реляційній моделі? Як ви дотрималися вимог цілісності у моделі БД, яку розробили?
6. Для чого використовується теорія нормалізації? Назвіть ознаки оптимального набора відношень.
7. Що позначають зв'язки між об'єктами концептуальної моделі? Які типи зв'язків бувають? Який зв'язок вважається найбільш оптимальним і чому?
8. Яке призначення первинних і зовнішніх ключів? Обґрунтуйте їх використання у розробленій вами схемі БД.
9. Якого типу зв'язок можна утворити між сутностями:
   а) Людина – Прописка;
   б) Лікарі – Пацієнти;
   в) Канали телебачення – Розважальні програми;
   г) Користувач – Профіль на сайті;
   д) Студенти – Керівники дипломних робіт.
10. У який спосіб набір відношень можна звести до більш оптимальної форми? Які принципи покладено в основу теорії нормалізації?

11. У яких нормальних формах перебувають відношення на вашій схемі БД? Які ознаки цього?
12. Із чим пов’язані аномалії, що виникають у неоптимізованих відношеннях та як їх усунути?

ТЕМА 4. Основи мови SQL. Створення таблиць бази даних

SQL (англ. *Structured Query Language* – мова структурованих запитів) – це декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, забезпечення системи контролю за доступом до бази даних і виконання операцій над нею [12].

Будучи ані системою керування базами даних, ані окремим програмним продуктом, SQL передбачає два способи застосування. Перший пропонує інтерактивний режим роботи, при якому користувач може вводити довільні SQL-оператори. Другий метод полягає у впровадженні SQL-команд до операторів мови програмування (C++, C#, Java, php, JavaScript, Python).

Перша версія SQL, розроблена на початку 1970-х років у IBM, мала назву SEQUEL і була призначена для обробки та пошуку даних, що містилися у реляційній базі даних IBM, System R. У 1986 році мова SQL була стандартизована Американськими Держстандартами (ANSI). Спочатку SQL розроблялась як мова запитів і керування даними, пізніші постачальники СУБД внесли деякі модифікації у вигляді процедурних конструкцій, control-of-flow команд і розширення мови. З випуском стандарту SQL:1999 такі розширення були формально запозичені як частина мови SQL через Persistent Stored Modules (SQL/PSM).

У стандартах SQL2003 і 2006 уведені і розширені можливості для роботи з XML-даними. Дотепер останнім стандартом вважається SQL2016, який реалізує захист на рівні рядків, поліморфні табличні функції та JSON.

Розрізняють такі групи SQL-операторів:

- оператори визначення даних (*Data Definition Language*, мова DDL): `Create`, `Alter`, `Drop` – для створення бази даних, визначення її структури, таблиць і обмежень, з можливістю подальшої модифікації або видалення;
- оператори маніпулювання даними (*Data Manipulation Language*, мова DML): `Select`, `Insert`, `Update`, `Delete` – для виконання вибірки, додавання нових записів, оновлення та видалення внесених даних;

• оператори визначення доступу до даних (*Data Control Language*, мова DCL): `Grant`, `Revoke`, `Deny` – надають користувачам (або групі) дозвіл на певні операції над об'єктом, відкликають або забороняють наданий раніше дозвіл;

• оператори управління транзакціями (*Transaction Control Language*, мова TCL): `Commit`, `Rollback`, `Savepoint` – для застосування транзакції, повернення її до вихідного стану або розбиття на кілька стадій.

Стандарт ISO SQL не підтримує таких формальних термінів, як «відношення», «атрибут» і «кортеж», натомість вживаються позначення «таблиця», «стовпчик» або «поле», «рядок» або «запис».

Створення таблиці. Типи даних мови SQL

Створення будь-якої таблиці в базі даних передбачає визначення властивостей її полів за допомогою оператора мови SQL у форматі [12, 14]:

`CREATE TABLE` Назва_таблиці

(назва_поля1 **тип_поля** *обмеження*,

…

назваполяN **тип_поля** *обмеження*);

Назви полів у межах однієї таблиці повинні бути унікальними. Поняття типу даних у реляційній моделі даних повністю відповідає типу даних у мовах програмування. Кожен елемент даних повинен мати свій тип, водночас накладати обмеження на значення полів необов'язково, про що йтиметься трохи далі.

Тип даних може бути скалярним або нескалярним. Нескалярні – це всі типи, явно визначені таким чином, що в них є компоненти, видимі для користувача. Зокрема, до нескалярних типів належать об'єкти відношень (таблиць), що всередині містіть поля та рядки.

Скалярні типи, навпаки, таких видимих складових не мають. До них належать, наприклад, поля таблиці, адже значення, яких вони набувають у кожному рядку, атомарні та неподільні. Сучасні реляційні БД допускають збереження бітових

послідовностей, символьних, числових, а також спеціалізованих даних, як-от «гроші» й «темпоральні» дані дати і часу. У мові запитів SQL користувачам не надається можливість для визначення власних типів. Підтримуються лише вбудовані типи даних, для позначення яких у мові SQL існують спеціальні зарезервовані слова. *Назви типів даних не залежать від регістру.* Після визначення тип поля буде зберігатися як його характеристика, яку бажано не змінювати.

Розглянемо найбільш поширені стандартні типи даних мови SQL [14, 16].

Цілочислові типи даних

Цілі числа утворюють великий клас даних, які можна зберігати у БД. Характеристики цілочислових типів, визначених у мові SQL, наведені у табл. 4.1.

Таблиця 4.1

Цілочислові типи даних мови SQL

Тип даних	**Розмір** (байти)	**Мін. знач.**	**Макс. знач.**
tinyint	1	0	255
smallint	2	-32768 (-2^{15})	32767 (2^{15} -1)
int (integer)	4	-2^{31}	2^{31} -1
bigint	8	-2^{63}	2^{63} -1
bit	1	0	1 (або будь-яке значення, відмінне від 0)

Дійсні тип даних

Дані цього типу дають змогу зберігати числові значення, які складаються з цілої та дробової частин, відокремлених комою.

Дійсним числам властива проблема із заокруглення як при використанні у виразах, так і при виведенні результату, адже ці значення можна подавати лише з певною точністю. При цьому в пам’яті їх значення незмінні. Для зберігання дійсних значень у мові SQL визначені наближені та точні типи даних, характеристики яких наведені у табл. 4.2.

Таблиця 4.2

Дійсні типи даних мови SQL

Тип даних	Розмір (байти)	Точність	Мін. знач.	Макс. ЗНАЧ.
real	4	до 7 цифр	-3,4E-38	+3,4E+38
float[(*n*)]	8	до 15 цифр	-1,7E-308	+1,7E+308
decimal[(*p*[, *s*])]	5 – 17	1 – 38	-10E38 +1	10E38 -1
numeric[(*p*[, *s*])]	5 – 17	1 – 38	-10E38 +1	10E38 -1
smallmoney	4	Decimal (10,4)	-214748,3648	214748,3647
money	8	Decimal (19,4)	-922337203685 477,5808	+922337203 685477,5807

Таблиця 4.3

Символьні типи даних мови SQL

Тип даних	Розмір	Довжина	Діапазон	Стандарт
char(*size*)	1 байт/символ	фіксована	до 8000 символів	ANSI
nchar(*size*)	2 байти/символ	фіксована	до 4000 символів	Unicode
varchar(*size*)	1 байт/символ	змінна	до 8000 символів	ANSI
nvarchar(*size*)	2 байти/символ	змінна	до 4000 символів	Unicode
text	1 байт/символ	змінна	до 2 ГБ на рядок	ANSI
ntext	2 байти/символ	змінна	до 1 ГБ на рядок	Unicode

Як бачимо, тип **real** забезпечує точність до 7 знаків у числі, тоді як для типу **float** завдяки параметру *n* її можна регулювати. Так, якщо вказати значення *n* в інтервалі від 1 до 24, поле займатиме 4 байти, а від 25 до 53 – 8 байтів. За замовчуванням точність складає 53 знаки.

Точні дійсні типи **decimal** і **numeric**, на відміну від двох попередніх, гарантують зберігання десяткових чисел з фіксованою точністю, без змін. При визначенні поля цього типу використовуються параметри *p* та *s*.

Параметр *p* визначає точність, з якою можна зберігати десяткове число (тобто загальну кількість цифр, що зберігаються). Точність можна задавати в діапазоні від 1 до 38, за замовчуванням встановлюється значення 18. Параметр *s* визначає кількість цифр після десяткової крапки. Це значення повинно бути меншим або рівним *p*. Якщо не зазначити явно, ці параметри набуватимуть стандартних значень ($p = 18$, $s = 0$).

Фінансові типи **money** і **smallmoney** передбачені для зберігання дробових грошових величин, аналогічно до типу **decimal** заданої точності [14].

Символьні типи даних

У мові SQL визначені типи даних для збереження рядків символів зі стандартного набору ANSI і для підтримки стандарту Unicode (табл. 4.3).

City **char** (25)

London_____ – (довжина 25 байтів)

City_Var **varchar**(20)

Tokyo – (довжина 5 байтів, за кількістю введених символів)

City_Nvar **nvarchar**(20)

Чернівці – (довжина 16 байтів, на кожен символ – 2 байти).

Таблиця 4.4

Темпоральні типи даних мови SQL

Тип даних	Розмір (байти)	Формат	Діапазон	Точність
time	3 – 5	гг:хх:сс [.нннннн]	від 00:00:00.0000000 до 23:59:59.9999999	100 нсек
date	3	РРРР-ММ-ДД	від 0001-01-01 до 31.12.99	1 день
datetime	8	РРРР-ММ-ДД гг:хх:сс[.ннн]	від 01.01.1753 до 31.12.9999	0,00333 сек
smalldatetime	4	ГГГГ-ММ-ДД чч:хх:сс	від. 01.01.1900 до 06.06.2079	1 хв

Як бачимо з табл. 4.3, для символьних стовпців можна явно зазначати максимальну довжину – параметр *size* у круглих дужках. Для типів з фіксованою довжиною, як-от **`char`** і **`nchar`**, усі значення матимуть однаковий розмір, незалежно від вмісту. Наприклад, якщо для стовпчика визначено тип **`char`**(25), а користувач вводитиме меншу кількість символів, решта позицій заповнюються пропусками.

Водночас, для символьних типів змінної довжини, до яких належать **`varchar`** і **`nvarchar`**, розмір ділянки пам'яті під значення поля цього типу визначатиметься за кількістю введених символів, а отже, для різних рядків може змінюватися. Це дозволяє суттєво заощадити місце на диску.

При перевищенні максимально допустимого (або заданого) розміру поля, СУБД автоматично і без попередження видаляє решту символів.

Типи дати і часу

Для зберігання комбінацій дати та часу замість символьних типів зручніше використовувати спеціальні темпоральні типи даних (табл. 4.4). У мові SQL визначені спеціальні функції для роботи з датою і часом, а також їх окремими складовими частинами, які ми розглянемо у темі 6.

Тип `timestamp`

Якщо для стовпця таблиці визначено цей тип, то кожного разу при додаванні нового або оновленні наявного запису в стовпці типу **`timestamp`** буде автоматично змінюватися значення лічильника, яке вказує на кількість виконаних операцій. Таблиця може містити лише одне поле цього типу. Його неможливо заповнити вручну і воно завжди набуватиме унікальних значень у межах таблиці та бази даних [14].

Обмеження

Крім зазначення безпосередньо типу та розміру, для полів таблиці можна задавати додаткові уточнення з метою звузити межі, визначені стандартними типами, та підтримки цілісності даних. При створенні таблиці (або при її зміні) можна накладати обмеження на значення, що вводяться до поля. У разі порушення

умови обмеження SQL відхилятиме будь-які значення, що не відповідають критеріям, визначеним для того чи іншого поля.

Обмеження, що використовуються в операторі створення таблиці **CREATE TABLE,** дозволяють визначити унікальність значень поля, забезпечити цілісність даних, захистити від внесення некоректних даних та підтримувати інформацію, що зберігається, в актуальному і несуперечливому стані. На одне поле можна накладати кілька обмежень.

Для визначення обмежень у мові SQL використовується зарезервоване слово **Constraint**.

У разі потреби до вже створеної таблиці можна додавати обмеження за допомогою оператора:

ALTER TABLE Назва_таблиці
Add Constraint назва_обмеження
назва_поля *обмеження*;

Обмеження даних можна розбити на такі групи:

• Цілісності сутностей (первинного ключа): **Primary Key**, **Unique**

• Обов'язкових значень: **Not Null, Identity, Default**

• Доменів атрибутів: **Check**

• Цілісності посилань (зовнішнього ключа): **Foreign Key**.

Унікальність. Цілісність сутностей

При створенні таблиці визначається набір потенційних ключів, тобто полів, значення яких унікальні для однозначної ідентифікації кожного рядка таблиці. Серед полів будь-якої таблиці обов'язково повинен бути **первинний ключ (Primary key, PK)**. Поле первинного ключа не може залишатися незаповненим і набувати для різних записів однакові значення. Первинний ключ зазвичай є полем цілого типу, яке за допомогою числового коду унікально позначає кожен рядок, тобто, екземпляр сутності, характеристики якого зберігаються у неключових стовпцях, та є першим кроком до забезпечення цілісності у всій БД. СУБД створює унікальний індекс для стовпця первинного ключа.

Можливі способи створення обмеження первинного ключа:

```
Create Table      Students
(       ID_stud       int   Primary Key,
        Sname         nvarchar(50),
);
```

АБО

```
Create Table Students
(       ID_stud       int,
        Sname         nvarchar(50),
…
        Primary Key (ID_stud)
);
```

АБО

```
Alter Table     Students
Add Constraint Nomer_Studenta Primary key (ID_stud);
```

Щодо інших потенційних ключів, то їх називають альтернативними ключами, і для позначення їх унікальності використовують **обмеження унікальності** (**Unique**), яке гарантує відсутність дублікатів даних стовпця.

Характеристики обмеження унікальності схожі з первинним ключем, проте є свої відмінності:

- В одній таблиці може бути лише один первинний ключ (**Primary Key**) але кілька альтернативних (**Unique**).
- Поле первинного ключа обов'язково потрібно заповнювати, водночас для альтернативних ключів допустимі пропуски значень.
- Обмеження `Unique` неможна вказувати для первинного ключа.
- Унікальний стовпець не може бути первинним ключем або входити до його складу

Приклад 1. При визначенні таблиці Працівники для поля жетон (Badge) зазначаємо обмеження **Unique**.

```
Create Table Employees
(       Name          char(20),
        Department    varchar(20),
        Badge         int   Unique);
```

Приклад 2. У таблиці Студенти вводимо альтернативний ключ – номер залікової книжки (Creditbook_id).

```
Create Table Students
(       ID_stud          int       Primary Key,
        Sname            nvarchar(50),
        Creditbook_id    int       Unique
);
```

Індекси

У базі даних індекси – це спеціальні структури, які забезпечують унікальність даних і допомагають прискорити пошук та сортування за визначеним полем або набором полів. Найпростішим прикладом індексів є зміст у книзі. Уявіть: якби не було змісту із зазначенням розділів та номерів сторінок, нам би довелося гортати кожну сторінку, щоб відшукати потрібне місце у книзі. Індекс виконує роль вказівника, який одразу скеровує нас на потрібну сторінку. Як відомо, у базі даних інформація зберігається у невпорядкованому вигляді. І тому, якби потрібно було робити пошук за іменами студентів або за номерами залікових книжок, то довелося би послідовно порівнювати кожне значення у полі з умовою пошуку.

Особливості використання індексів:

- зазвичай створюються як обмеження для полів, за якими виконуватиметься сортування, фільтрування або з'єднання рядків таблиці;
- завжди автоматично створюються для стовпців, на які накладається обмеження Primary key, а також Foreign key;
- обов'язково створюється СУБД для стовпців, на які накладено обмеження унікальності (Unique);
- найкраще створювати індекси для полів із мінімальною кількість повторюваних значень і рівномірним розподілом даних;
- при внесенні змін до таблиці також автоматично змінюються індекси, накладені на цю таблицю. У результаті індекс може бути сильно фрагментований, що позначається на продуктивності виконання операцій.

Для створення індекса у базі даних використовується команда SQL формату:

Create Index *назва_індекса*
On Назва_таблиці (назва_поля),

де *назва_індекса* зазвичай створюється за принципом: *таблиця_поле_idx*. Наприклад, для таблиці Students створимо індекс для імені студента:

Create Index *students_sname_idx*
On Students (Sname);

Якщо доведеться виконувати пошук за іменем студентів, створений індекс автоматично допоможе нам у цій операції.

Індекси бувають різних типів: унікальні, первинного ключа, кластеризовані, некластеризовані.

Кластеризовані індекси (**CLUSTERED**) фізично сортують рядки в таблиці на основі кластеризації (за замовчуванням − первинний ключ) і допомагають швидко вилучати дані з бази. Кластеризований індекс зберігає реальні набори даних. Тобто рядок даних пов'язаний зі значенням ключа, що його позначає, зберігатиметься у самому індексі. Важливою характеристикою кластеризованого індексу є те, що всі його значення відсортовані за зростанням або за спаданням. Тому кластеризований індекс у таблиці може бути лише один.

Некластеризовані (**NONCLUSTERED**) індекси містять лише ті стовпці, за якими визначено індекс, а також мають вказівник на рядки з реальними даними в таблиці. Це означає, що системі підзапитів буде потрібна додаткова операція для виявлення та одержання необхідних даних. У таблиці можуть бути кілька кластеризованих індексів. Самі дані не зберігаються в індексі та виводяться з таблиці, використовуючи ідентифікатор рядка або ключ кластерного індексу.

Різниця між цими двома типами індексів полягає в тому, що некластеризовані індекси створюються засобами СУБД за замовчуванням. Дані фізично розташовані у випадковому порядку, але логічно впорядковані за індексом. Такий тип індексування підходить для таблиць, де часто змінюються

значення. При кластерному індексуванні дані фізично впорядковані, що значно підвищує швидкість вибору даних.

Забезпечення обов'язкових значень

Деякі поля таблиці важливо не залишати незаповненими, а вносити до них дані вручну або автоматично. Прикладами таких полів можуть бути реєстраційні дані користувача, обов'язкові для оформлення замовлення, або номер замовлення – поле первинного ключа, яке ідентифікує запит від клієнта. Для цих та інших цілей використовуються обмеження обов'язкових значень:

- **Можливість пропусків `Null` і `Not Null`**

Визначений за замовчуванням для стовпців атрибут **`Null`** дозволяє при вводі явно не вносити значення до полів. Натомість, обмеження **`Not Null`** потрібно зазначати аби унеможливити пропуски значень у деякому стовпчику під час заповнення рядка. Атрибут **`Not Null`** – обов'язковий для поля первинного ключа.

Приклад 3. Значення первинного ключа повинні визначатися явно.

```
Create Table    Students
(       ID_stud     int   Primary Key Not Null,
        Sname       nvarchar(50));
```

- **Автозаповнення `Identity/Auto_increment`**

Поля таблиці, що виконують роль порядкового номера або лічильника, необов'язково повинні містити дані зі змістовим наповненням. Головне, щоб їх значення були унікальними і не повторювалися при введені нового запису. Такі поля можна заповнювати вручну й стежити, щоб при введенні не було дублікатів. Проте зручніше визначити для них властивість автоматичного збільшення значення поля. З цією метою у MS SQL Server передбачена властивість **`Identity`**, а в MySQL – **`Auto_increment`**. Для автозаповнення можна зазначати два параметри: **`Identity`**(*i, s*), де *i* – це початкове значення, *s* – крок, що додаватиметься до попереднього значення поля. Цю властивість можна надати стовпцям цілочислових типів, а також дійсним полям типу `decimal`(p,0) або `numeric`(p,0). Якщо для

атрибута **Identity** не вказати параметри, за замовчуванням встановлюватимуться значення (1, 1). При використанні цього атрибуту властивість **Not Null** забезпечується автоматично.

Приклад 4. Реалізація автозаповнення номера відділу (ID_dept).

```
Create Table      Departments
( ID_dept  int  Identity(100, 20) Primary Key,
  Dname   nvarchar(50)
);
```

При додаванні нового запису до таблиці Departments номер відділу явно зазначати не потрібно:

```
Insert Into Departments (ID_dept, Dept_name)
Values  ('Research & Development');
```

❖ **Значення за замовчуванням Default**

Коли при формуванні нового рядка таблиці одне або кілька його полів залишаються порожніми, для заповнення цих пропусків у мові SQL передбачається використання *значення за замовчуванням*, що позначається зарезервованим словом **Null**. Значення NULL відрізняється від числа нуль або символьного поля, заповненого пробілами, адже воно позначає, що комірка не містить конкретних даних. Проте користувач може визначити інше значення, що додаватиметься до поля в разі пропуску.

Властивість поля **Default** використовується при створенні таблиці у той самий спосіб, що й інші обмеження, хоча, з технічної точки зору, не має обмежуючих властивостей. Цей атрибут замінює традиційне **Null** на значення, що найчастіше зустрічатиметься у стовпці.

Приклад 5. У таблиці більшість покупців – з Чернівців, тому саме це значення за замовчуванням можна обрати для поля City:

```
Create Table Customers
(     Customer_id     int Primary Key Not Null,
      Custom_name     nvarchar(20),
      City            nvarchar(20) Default 'Чернівці'
);
```

Визначення доменів атрибутів

У мові SQL передбачена можливість запобігти введенню небажаних або помилкових значень. Для цього використовується обмеження **Check**. З його допомогою для одного чи кількох полів таблиці можна створити умову, що має справджуватися для значень, перш ніж долучити їх до запису. Саме це обмеження визначає домен атрибута, у вигляді діапазону, переліку або шаблону логічно припустимих значень.

Приклад 6. Для поля City з таблиці Працівників визначаємо множину допустимих значень, яких воно може набувати (написання слів має чітко відповідати назвам міст у переліку). Обмеження для поля Badge – номер жетона, запобігатиме введенню від'ємних значень.

```
Create Table Employees
(       Name     char(20),
        ID_dept  int,
        City     varchar(20)
        Check (City IN ('London', 'Paris', 'Rome', 'Athens')),
        Badge    int Check (Badge>0),
        Phone_no char(14)
);
```

Приклад 7. Додамо до таблиці Employees обмеження для поля номера телефону, таке щоб всі номери починалися з 0 та коду оператора у круглих дужках, а сам номер узгоджувався з певним шаблоном. При цьому поля, до яких вже були введені значення, залишатимуться без перевірки, навіть якщо їх значення не відповідають зразку:

```
Alter Table Employees
With Nocheck
Add Constraint empl_chk_num Check
(Phone_no Like '(0[5-9][0-9])[0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9]');
```

Утворення зв'язків. Цілісності посилань

На основі базових таблиць можна створювати пов'язані з ними дочірні таблиці. Для цього первинний ключ батьківської

таблиці переноситься як **зовнішній ключ (`Foreign Key, FK`)** до похідної таблиці для опису взаємодії між різними сутностями.

Властивості зовнішнього ключа:

- Посилається на ідентичне поле батьківської таблиці (найчастіше на поле `Primary Key, PK`), а отже використовується лише після її створення.
- Визначений на домені первинного ключа, за яким утворено зв'язок.
- Пов'язані поля можуть відрізнятися за назвою, проте повинні збігатися за типом і довжиною.
- Значення зовнішнього ключа можуть автоматично оновлюватися при зміні первинного.

Обмеження цілісності посилань або зовнішнього ключа розміщується у дочірній таблиці і має такий формат:

`Foreign Key` (поле_FK) **`References`**
Назва_батьків_табл (поле_PK) **`On Update/On Delete`** *правило*

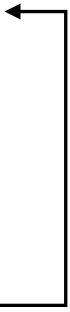
Правила:

- **`Cascade`** – видалення/оновлення рядків батьківської таблиці викликають автоматичні зміни у дочірній таблиці.
- **`Set Null`** – значення, видалені у батьківській таблиці, встановлюються в Null у дочірній.
- **`Set Default`** – видалені значення набуватимуть значення за замовчуванням.
- **`Restrict/No Action`** – забороняються будь-які зміни.

Приклад 8. Дочірні таблиця Співробітники (Employees) посилається на батьківську таблицю Відділи (Departments) завдяки створеному обмеженню зовнішнього ключа для поля Dept_id.

```
Create Table Departments
(
  ID_dept int Primary Key Not Null,
  Dname nvarchar(50)
);

Create Table Employees
(
  Name char(20),
  Dept_id int,
  Badge int
  Foreign Key (Dept_id) References Departments (ID_dept)
);
```



ЛАБОРАТОРНА РОБОТА № 4
Перевантаження моделі бази даних до СУБД

Мета

Використати онлайн-середовище **DB Designer** для перевантаження моделі бази даних до СУБД **MS SQL Server** з автоматичним генеруванням SQL-коду.

ПРАКТИЧНЕ ЗАВДАННЯ

Forward Engineering – це процес перенесення моделі бази даних до СУБД, що передбачає автоматичне створення таблиць, з полями, обмеженнями та зв'язками, визначеними у розробленій ER-діаграмі. Успішне перевантаження концептуальної моделі передбачає дотримання такої послідовності кроків:

1. Створення у відповідній системі бази даних, до якої буде перенесено ER-діаграму.
2. Підготовка логічної моделі до перевантаження шляхом уточнення характеристик сутностей, атрибутів, зв'язків.
3. Генерування і збереження SQL-коду зі створення розробленого набору таблиць.
4. Запуск на виконання згенерованого SQL-коду в СУБД.
5. Перевірка правильності створення таблиць у БД, у разі потреби, усунення невідповідностей перевантаженої бази даних.

1. Створення бази даних у СУБД MS SQL Server 2019

Увійдіть до середовища **Microsoft SQL Server Management Studio** з групи програм **Microsoft SQL Server Tools**. Оберіть з'єднання із сервером, що пропонується за замовчуванням. У вікні, що відображається ліворуч, клацніть правою кнопкою миші на папці **Databases** і оберіть опцію зі створення нової БД – **New Database** (рис. 4.1).

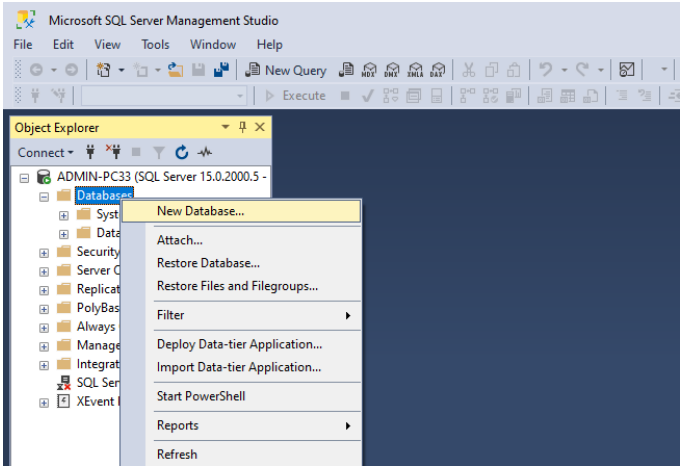


Рис. 4.1. Створення нової БД у середовищі Microsoft SQL Server Management Studio

У вікні, що з'явилося (рис. 4.2), надайте новій БД назву латинськими літерами, бажано без пробілів. Для збереження натисніть ОК.

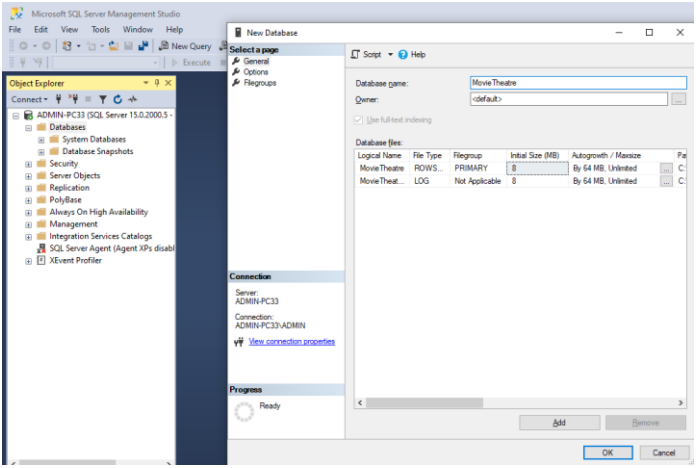


Рис. 4.2. Вікно для визначення параметрів нової БД

2. Підготовка логічної моделі до перевантаження

У веб-браузері перейдіть до онлайн-середовища **DB Designer** за посиланням https://www.dbdesigner.net/. Увійдіть до свого облікового запису через пункт меню **Login**. З переліку збережених проектів оберіть потрібну схему БД.

Середовище проектування схеми БД – це не просто засіб графічного відображення схеми БД. Завдяки його можливостям нам не доведеться вручну створювати таблиці, визначати їх склад та формувати з'єднання у СУБД. Натомість, на основі розробленої логічної моделі буде згенеровано SQL-скрипт зі створення відповідних компонентів, який можна зберегти та запустити на виконання у конкретному середовищі. Оскільки навіть найменші невідповідності викликатимуть помилки при створенні реальних таблиць, перш ніж братися за фізичну реалізацію розробленої моделі, необхідно пересвідчитися у правильності налаштування таких параметрів:

- назви сутностей (таблиць) і атрибутів (полів);
- типи даних для атрибутів;
- первинні ключі та їх властивості;
- зовнішні ключі та зв'язки між таблицями.

При підготовці до перевантаження логічної моделі бази даних до СУБД слід дотримуватися таких рекомендацій:

- ✓ **Переконайтесь, що ER-діаграма розроблена саме для тієї СУБД, до якої її буде перенесено**. Необхідна інформація міститься на панелі **Properties** (Властивості) у правому верхньому куті середовища DB Designer. Для зміни СУБД створіть нову схему БД, вказавши відповідну систему, скористайтесь пунктом меню **Schema>Load** і оберіть з переліку модель БД.
- ✓ Назви таблиць та атрибутів потрібно надавати **латинськими літерами**.
- ✓ Імена повинні бути змістовними, містити лише літери, цифри та позначку підкреслення, мати довжину не більше 10 символів (для зручності читання). Більше рекомендацій щодо іменування об'єктів у SQL можна знайти у [13].

- ✓ Назви об’єктів не можуть бути зарезервованими словами мови SQL, як-от *group*, *order*, *year*, *table*, *user.* Повний перелік можна переглянути за посиланням [15].
- ✓ Для уникнення небажаних збігів та уточнення змісту атрибуту краще при іменуванні поєднувати декілька слів, наприклад, *student_group*, *order_id*, *release_year*, *reserved_table*, *book_user*, тощо.
- ✓ Переконайтесь, що для атрибутів зазначені типи, що підтримуються СУБД, у якій створюватиметься база даних. Зокрема, перелік типів, визначених для MS SQL Server, можна переглянути у [16].
- ✓ Довжину слід вказувати лише для символьних атрибутів типу **`char`**, **`nchar`**, **`varchar`**, **`nvarchar`**.
- ✓ Первинні ключі повинні бути цілочисловими.
- ✓ Властивість **Auto Increment** краще обирати лише для первинних ключів.
- ✓ При утворенні зв’язків типи первинних і зовнішніх ключів повинні збігатися.
- ✓ Для полів первинного та зовнішнього ключів необхідно встановити властивість **Not Null**, що унеможливлює пропуски значень. Для решти атрибутів доцільно обрати опцію **Allow nulls**.

Будь-які виявлені невідповідності потрібно усунути на самій ER-діаграмі, адже не всі з них вдасться виправити після перенесення до СУБД.

3. Генерування і збереження SQL-коду

Аби розпочати перенесення готової схеми БД, у меню середовища DB Designer оберіть пункт **Export >Sql** (рис.4.3).

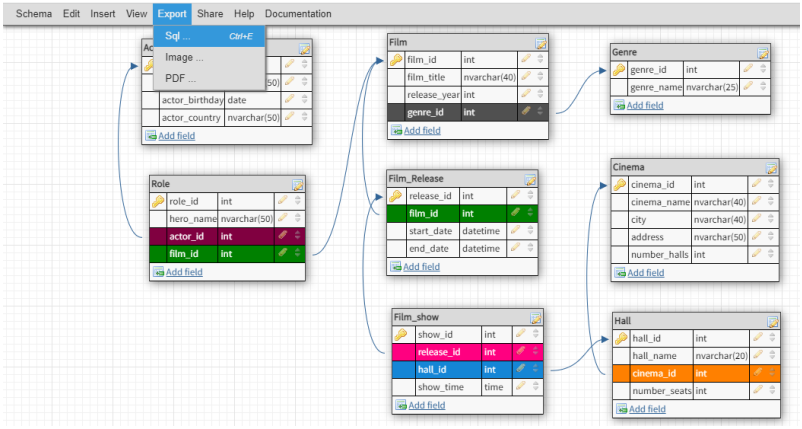


Рис. 4.3. Опція Export в онлайн-середовищі проектування бази даних DB Designer

У вікні, що з'явилося, оберіть СУБД, у якій на першому кроці була створена база даних для перенесення логічної моделі. Як бачимо на рис. 4.4, це MS SQL Server. Аби згенерувати SQL-код зі створення відповідної схеми БД, позначаємо опцію **Create script** і натискаємо кнопку **Generate SQL**.

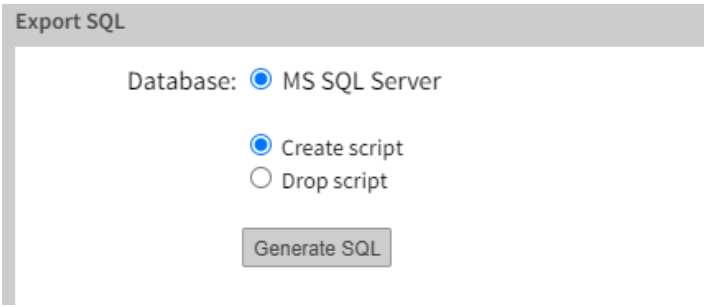


Рис. 4.4. Налаштування параметрів для генерування SQL-скрипта зі створення вмісту БД

Згенерований SQL-код, що відповідає розробленій логічній моделі БД, з'являється в окремому вікні (рис. 4.5). Його можна зберегти у SQL-файл (**Download SQL file**), виділити і скопіювати вручну до текстового документа або повернутися до попереднього кроку (**Back**).

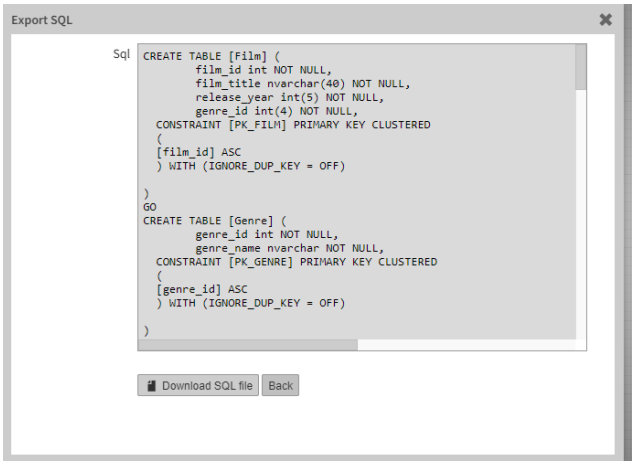


Рис. 4.5. Вікно з SQL-скриптом, згенерованим на основі розробленої логічної моделі БД

Перегляньте згенерований SQL-скрипт, зверніть увагу на синтаксис команд зі створення таблиць, визначення їх складу, формування зв'язків та інших обмежень.

Увесь згенерований SQL-код виділіть, скопіюйте у буфер і спершу додайте до звіту з лабораторної роботи (рис. 4.6).

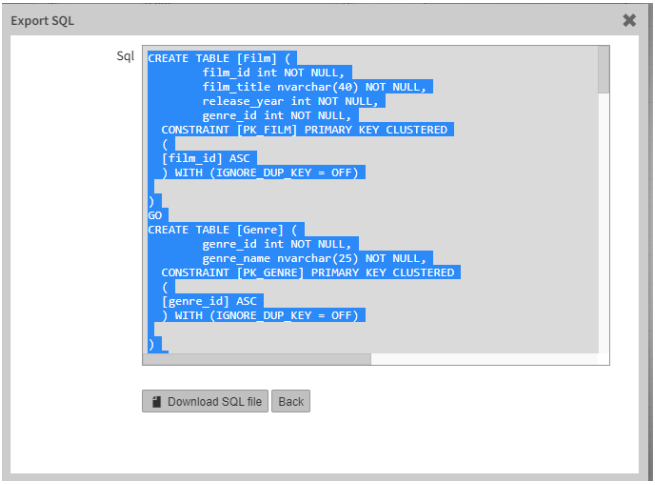


Рис. 4.6. Копіювання згенерованого SQL-коду

4. Запуск на виконання у СУБД згенерованого SQL-коду

Перейдіть до середовища Microsoft SQL Server Management Studio, оберіть створену базу даних. Як показано на рис. 4.7, у меню оберіть опцію **New Query** (Новий запит).

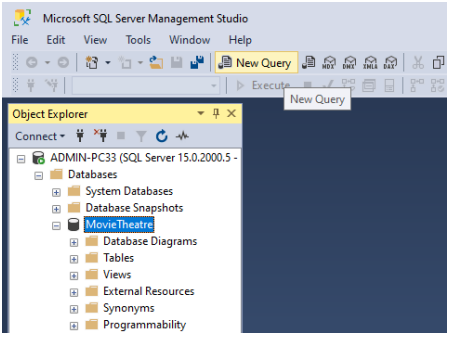


Рис. 4.7. Створення у базі даних нового запиту для запуску згенерованого SQL-коду

Додайте попередньо скопійований SQL-скрипт до вікна зі створення нового запиту до бази даних (рис. 4.8). Переконайтесь, що додано весь код без втрат. Як бачите, команди можна редагувати, проте не варто вдаватися до цього без нагальної потреби.

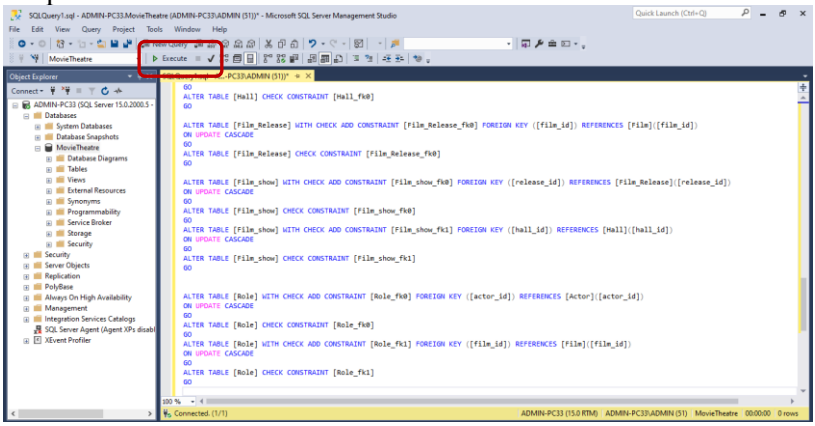


Рис. 4.8. Додавання згенерованого SQL-коду до нового запиту в базі даних

Запуск на виконання доданого набору SQL-команд дасть змогу створити у базі спроектовані на ER-діаграми таблиці з визначеними властивостями та зв'язками. Запит можна запустити за допомогою кнопки **Execute** (Виконати), позначеної на рис. 4.8.

Перелік перенесених до БД таблиць відображатиметься у папці Tables у вікні зліва. Інакше, натисніть правою кнопкою миші на назві робчої бази даних або на вільному місці у вікні зліва, та оберіть опцію **Refresh** (Оновити). Відгук про результат виконання запиту відображатиметься у нижній частині вікна (рис. 4.9). Інакше, натисніть правою кнопкою миші на назві робчої бази даних або на вільному полі у вікні злів та оберіть опцію Refresh (Оновити). У нашому випадку повідомлення свідчить про успішне виконання.

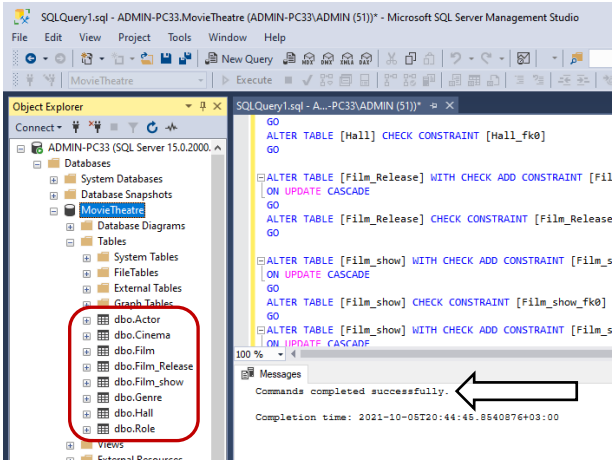


Рис. 4.9. Результат виконання запиту зі створення таблиць у базі даних

5. Перевірка правильності створення таблиць у БД

Перелік створених таблиць можна переглянути у відповідній базі даних, відкривши папку **Tables**. У нашому випадку в MovieTheatre утворено вісім таблиць (рис. 4.9), що узгоджується з розробленою схемою бази даних.

Поява після запуску SQL-коду повідомлень, виділених червоним кольором, свідчить про наявність помилок при виконанні запиту. Перегляньте зміст повідомлень, зазвичай вони

вказують на невідповідності, які виникли при створенні таблиць, полів або обмежень, визначених на схемі. **Помилка у визначенні параметрів одного атрибута може унеможливити створення цілої таблиці, а також її дочірніх таблиць**. Поверніться до середовища розробки ER-діаграми. Перевірте та виправіть параметри, які спричинили конфлікт. Повторно згенеруйте код та запустіть його на виконання у СУБД.

6. Створення діаграми бази даних, перенесеної до СУБД

Для зручності перевірки результатів перевантаження логічної моделі бази даних до СУБД створимо її діаграму, яка графічно відобразить наявний набір користувацьких таблиць із повним набором полів та зв'язків. Щоб створити діаграму бази даних, оберіть відповідну папку **Database Diagram**, клацніть на ній правою кнопкою миші та у контекстному меню оберіть пункт **New Database Diagram** (рис. 4.10).

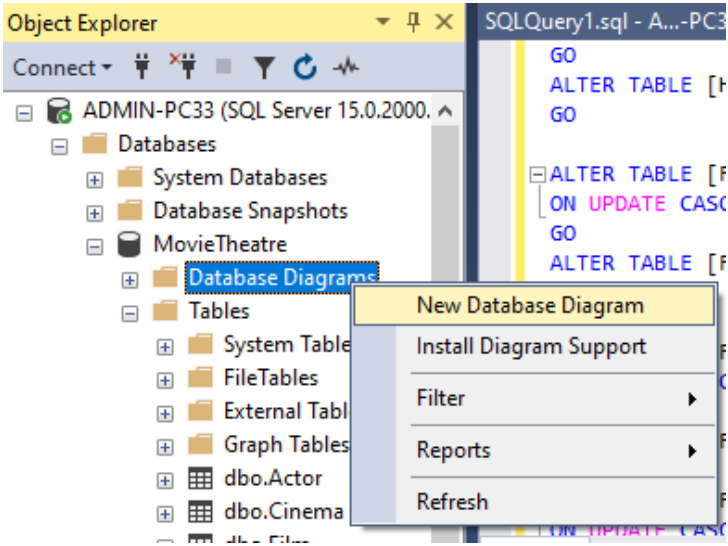


Рис. 4.10. Створення нової діаграми бази даних у Microsoft SQL Server Management Studio

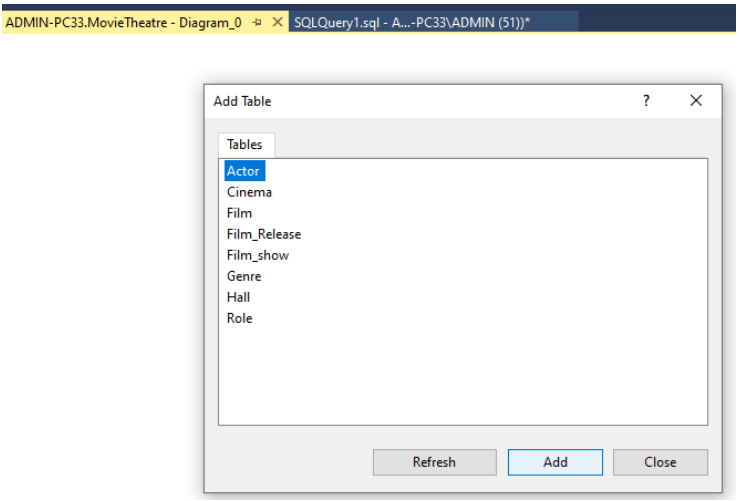


Рис. 4.11. Додавання таблиць до діаграми бази даних у СУБД

Далі з'являється вікно **Add Table**, у якому можна обрати таблиці для залучення до діаграми (рис. 4.11). Додана таблиця автоматично вилучається з переліку.

У результаті буде одержано діаграму, приблизний вигляд якої наведений на рис. 4.12. На ній одразу можна перевірити наявність усіх таблиць, їх склад, а також переконатися у правильності утворення зв'язків.

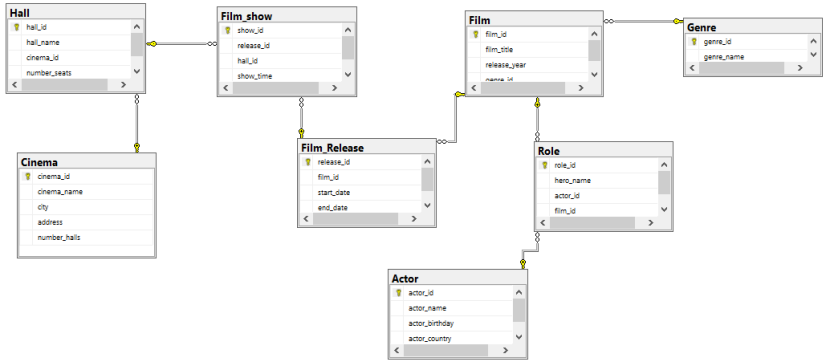


Рис. 4.12. Результуюча діаграма створеної бази даних

Утворена діаграма входить до складу бази даних. Для її збереження при закритті діаграми зазначте назву, як показано на рис. 4.13, і натисніть ОК.

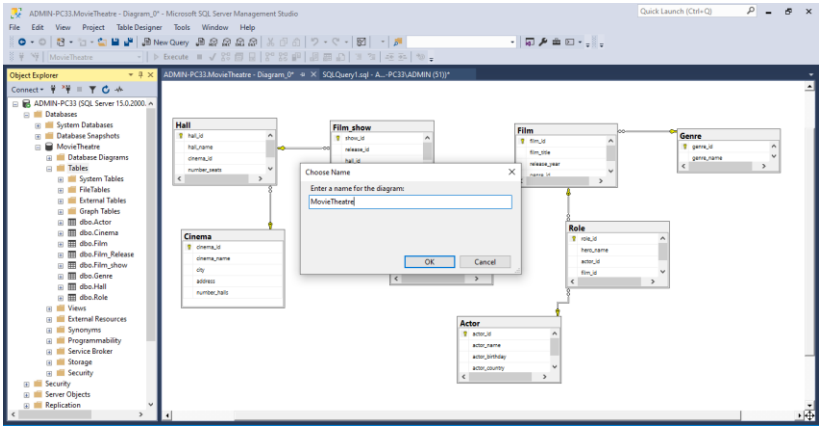


Рис. 4.13. Збереження нової діаграми у базі даних

Якщо при перевантаженні моделі деякі таблиці були втрачені, їх можна створити вручну безпосередньо у базі даних. Також можна відновити зв’язки між таблицями. Припустимо, що було втрачено зв’язки між таблицями Film, Actor та Role (рис. 4.14).

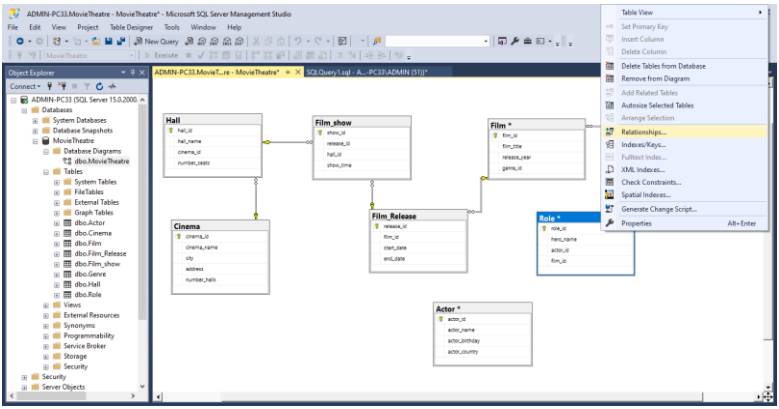


Рис. 4.14. Відновлення зв’язків між таблицями Film, Role та Actor

Нагадаємо, що, згідно з розробленою діаграмою, сутність Role походить від батьківських сутностей Actor та Film і утворює з ними зв'язок типу «один-до-багатьох». Для цього у дочірній сутності Role є два зовнішні ключі: **actor_id** та **film_id**, пов'язані з первинними ключами відповідних базових сутностей. Щоб відновити з'єднання, клацніть правою кнопкою миші на похідній таблиці Role і оберіть пункт меню **Relationships** (Зв'язки), як показано на рис. 4.14.

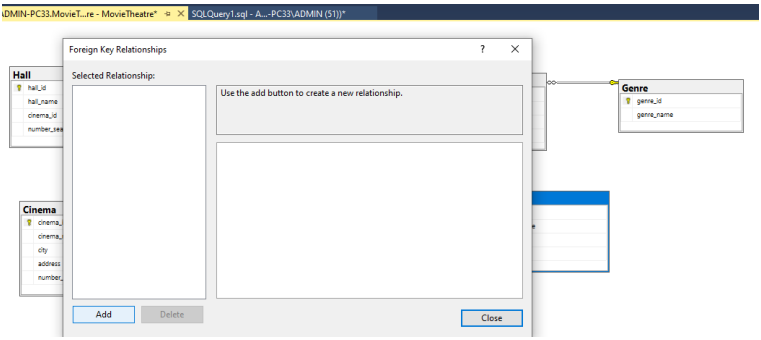
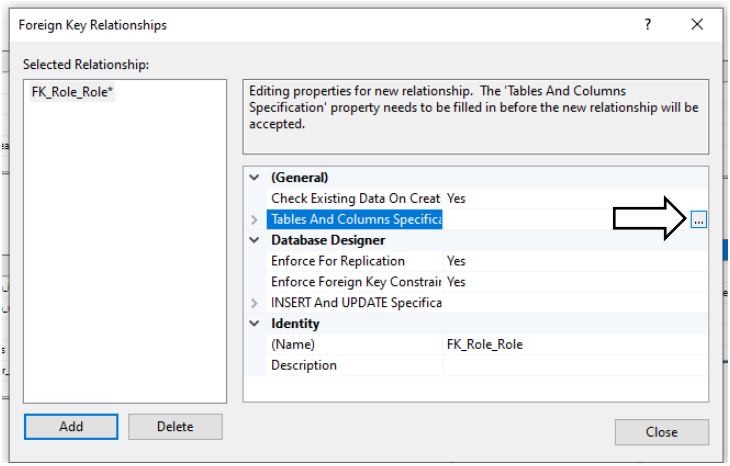
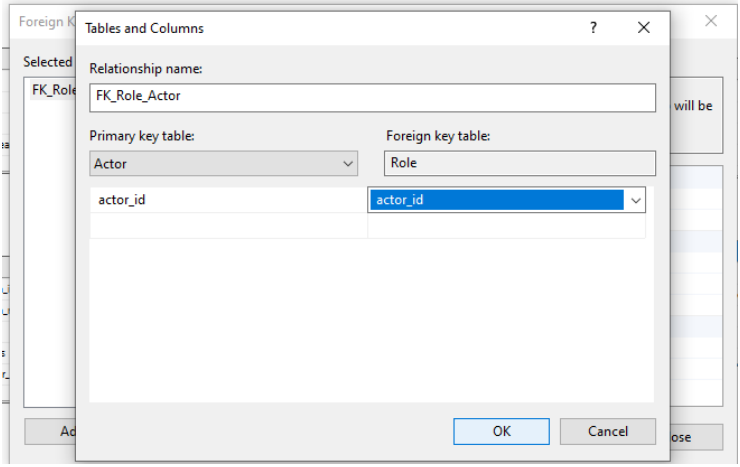


Рис. 4.15. Створення зв'язку в дочірній таблиці

У вікні **Foreign Key Relationships** (Зв'язки зовнішнього ключа) (рис. 4.15) для утворення нового зв'язку натискаємо кнопку **Add** (Додати). У лівій частині вікна з'являється умовне позначення нового зв'язку. Для визначення пов'язаних полів і таблиць для нього обираємо у правій частині вікна пункт **Table And Column Specification** (Специфікація таблиць і полів) і натискаємо кнопку з трьома крапками (рис. 4.16, *а*).



а



б

Рис. 4.16. Визначення властивостей зв’язку між таблицями

При відновленні зв’язку з таблицею Actor у вікні з визначення таблиць і полів (**Tables and Columns**), як видно на рис. 4.16, *б*, праворуч відображається дочірня таблиця Role (**Foreign key table**). Обираємо серед її полів зовнішній ключ **actor_id**. Ліворуч обираємо з переліку батьківську таблицю Actor (**Primary key table**) і з набору полів обираємо поле первинного

ключа – **actor_id**, до якого веде посилання. Зверніть увагу, що зв'язок змінив свою назву на FK_Role_Actor. Для збереження змін натискаємо ОК.

Аналогічно у вікні, зображеному на рис. 4.16, *а*, можна додати ще одне з'єднання із таблицею Film і, обравши його у розділі **Selected Relationships**, визначити відповідні параметри (рис. 4.17).

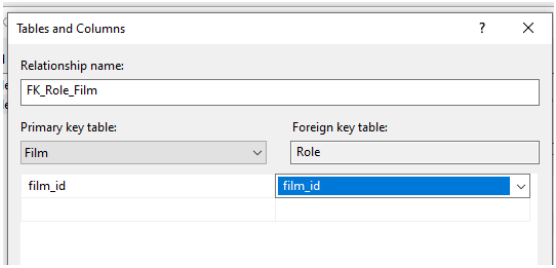
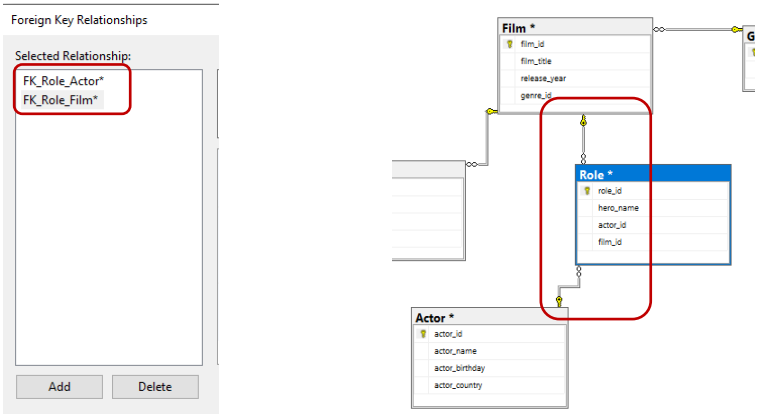


Рис. 4.17. Параметри зв'язку між таблицями Film та Role



а *б*

Рис. 4.18. Відновлені зв'язки між дочірньою та батьківськими таблицями у Microsoft SQL Server Management Studio: *а* – позначення створених зв'язків, *б* – їх зображення на діаграмі

У результаті виконаних налаштувань до таблиці Role відновлено два зв'язки, які можна побачити як у переліку **Selected Relationships** (рис. 4.18, *а*), так і графічно на діаграмі бази даних (рис. 4.18, *б*).

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. *Створіть у відповідній СУБД базу даних, до якої буде перевантажено спроектовану в DB Designer схему БД.*

2. *Перевірте логічну модель БД та підготуйте її для перевантаження згідно з наданими рекомендаціями.*

3. Згенеруйте та збережіть у звіті SQL-скрипт зі створення спроектованої моделі БД.

4. Запустіть на виконання згенерований SQL-код у СУБД, перевірте правильність створення таблиць, у разі потреби, внесіть корективи до розробленої моделі.

5. *Створіть та збережіть діаграму перевантаженої бази даних у СУБД.*

Звіт з лабораторної роботи повинен містити:

1. Остаточний варіант ER-діаграми, що використовується для перевантаження.
2. SQL-код, згенерований при перевантаженні моделі бази даних до СУБД.
3. Скріншот діаграми бази даних у СУБД.
4. Відповіді на запитання для самоперевірки.

Для захисту лабораторної роботи необхідно підготувати звіт та бути готовими відповісти на запитання, наведені нижче.

Запитання та завдання для самоперевірки

1. Який вигляд має оператор створення нової таблиці та змінення вже існуючої?
2. У який спосіб тип, зазначений для атрибута, впливає на його допустимі значення та на спосіб зберігання їх у пам'яті?

3. Які типи даних мови SQL визначені для збереження цілочислової інформації?
4. Назвіть точні та наближені дійсні типи даних мови SQL і вкажіть їх параметри.
5. Які типи даних передбачені для збереження символьної інформації? Чим вони відрізняються?
6. Які типи дати та часу визначені у мові SQL? Як працювати з даними цих типів?
7. Для чого передбачений тип даних **`timestamp`**? Які особливості його використання?
8. Які типи обмежень визначені при роботі з таблицями БД?
9. Для чого використовується обмеження **`Check`**? Наведіть приклади його можливого використання у вашій базі даних.
10. У який спосіб можна запобігти появі незаповнених полів у таблиці? Чи завжди для цього підійде автозаповнення?
11. Порівняйте характеристики полів з обмеженнями Unique та Primary Key.
12. Як засобами мови SQL при створенні таблиць реалізувати зв'язок між ними? Продемонструйте на прикладі вашої бази даних.

ТЕМА 5. Базові оператори мови SQL

У розділі 3 йшлося про два з трьох аспектів реляційної моделі даних – структурний і цілісний, а цю тему присвячено способам маніпулювання даними. Як зазначалося, аспект обробки ґрунтується на двох базових механізмах маніпулювання реляційними даними: основаній на теорії множин реляційній алгебрі та реляційному численні, що базується на математичній логіці. Обидва механізми володіють однією важливою властивістю: **реляційною замкненістю** відносно поняття відношення. Це означає, що *вирази реляційної алгебри і формули реляційного числення застосовуються до відношень реляційної БД і результатами обчислень також є відношення*. Зокрема, кожна операція повинна породжувати відношення із заголовком – визначеним переліком атрибутів, та тілом, що йому відповідає. Основна причина цієї вимоги – можливість посилатися на ці атрибути у наступних операціях.

Розгляньмо детальніше вирази реляційної алгебри.

Реляційна алгебра

Оскільки відношення – це множини, засоби маніпулювання ними спираються на традиційні теоретико-множинні операції, доповнені операціями, специфічними для баз даних. Запропонований Коддом [1] набір основних операцій над відношеннями реляційної алгебри, складається з восьми операцій, що належать до таких двох класів.

Теоретико-множинні операції:

- об′єднання
- перетин
- різниця (віднімання)
- декартовий добуток

Спеціальні реляційні операції:

- обмеження (вибірка)
- проекція
- з′єднання
- ділення

Теоретико-множинні операції

Особливості найбільш поширених операцій реляційної алгебри та їх реалізацію мовою SQL ми розглянемо на прикладі таблиць з БД MovieTheatre (рис. 5.1). Для зручності подання даних структуру таблиці Films дещо змінено.

Actors

actor_id	actor_name	actor_bday	actor_country
1	Джонні Депп	1963-06-09	США
2	Омар Сі	1978-01-20	Франція
3	Кіра Найтлі	1985-03-26	Велика Британія
4	Бенедикт Кембербетч	1956-07-09	Велика Британія
5	Том Генкс	1956-07-09	США
6	Одрі Тоту	1976-08-09	Франція

Films

film_id	film_title	release_year	genre
1	Пірати Карибського моря	2003	Фентезі
2	Гра в імітацію	2014	Трилер
3	Втеча з Шоушенка	1994	Драма
4	Зелена миля	1999	Драма
5	Недоторканні	2011	Комедія

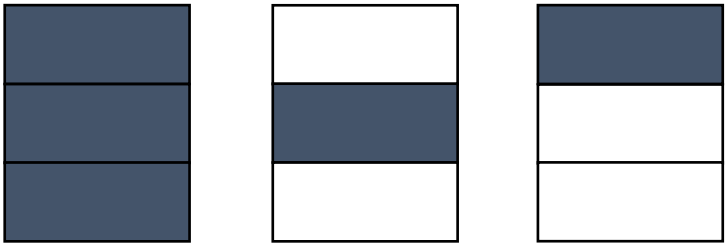
Roles

role_id	hero_name	actor_id	film_id
1	Джек Горобець	1	1
2	Дріс	2	5
3	Алан Тюрінг	4	2
4	Елізабет Свон	3	1
5	Джоан Кларк	3	2
6	Пол Еджкомб	5	4

Рис. 5.1. Фрагмент БД «Мережа кінотеатрів»

Оскільки теоретико-множинні операції реляційної алгебри ґрунтуються на класичній теорії множин, вони володіють деякими особливостями.

❖ Операції ***об'єднання***, ***перетину*** і ***різниці*** виконуються над двома відношеннями, що мають бути **сумісні за об'єднанням**, тобто володіти однаковими заголовками. Ба більше, у заголовках обох відношень міститься один і той самий набір імен атрибутів, визначених на однакових доменах. *Для виконання зазначених вище операцій достатньо, щоб у двох відношеннях був принаймні один спільний атрибут.* У результаті виконання операцій одержується відношення. Якщо деякий кортеж належить одразу до двох відношень-операндів, у результуюче відношення він потрапляє в єдиному екземплярі.



а *б* *в*

Рис. 5.2. Ілюстрація виконання теоретико-множинних операцій реляційної алгебри: *а* – об'єднання, *б* – перетин, *в* – різниця

- При **об′єднанні** (**`Union`**) двох відношень A і B одержується відношення, що містять всі кортежі, наявні у відношенні A, відношенні B та обох відношеннях одночасно.
- Операція **перетину** (**`Intersect`**) двох відношень A і B формує відношення, що містить всі кортежі, спільні для обох відношень-операндів.
- Відношення, що є **різницею (`Except`)** двох відношень, містить всі кортежі, що містяться у відношенні A, проте відсутні у відношенні B.

Зауважимо, що залучення до складу операцій реляційної алгебри трьох операцій об'єднання, перетину і різниці

надлишкове, оскільки будь-яку з цих операцій можна виразити через дві інші.

Загалом, реалізація об'єднання мовою SQL має такий формат:

```
Select * From A Union Select * From B;
```

Розглянемо приклад створення запиту для виконання операції об'єднання. Попри те, що відношення з бази даних MovieTheatre мають різні заголовки, у них є спільні атрибути завдяки зв'язкам таблиці Role із батьківськими таблицями Actors і Films.

Приклад 1. Знайти номери англійських акторів, що грали у фільмі з номером 2.

```
Select actor_id
From Actors
Where actor_country='Велика Британія'
Union
Select actor_id
From Roles
Where film_id=2;
```

Результат:

actror_id
3
4

У результаті об'єднання одержується унарне відношення з атрибутом actor_id, кортежі якого містять номери акторів, що відповідають зазначеним умовам пошуку. Попри те, що номери 3 і 4 зустрічаються в обох відношеннях-операндах, до результуючої таблиці вони потрапляють у єдиному екземплярі. На жаль, номер актора неможливо доповнити іменем, оскільки цей атрибут не міститься також у таблиці Roles, з якою відбувається об'єднання.

❖ Операція ***декартового добутку*** двох відношень A та B дає у результаті відношення, заголовок якого є злиттям заголовків відношень-операндів, а тіло складається з усіх можливих комбінацій кортежів відношень A та B (рис. 5.3):

$$A*B=\{(A1B1)(A1B2)...(AnBn)\}$$

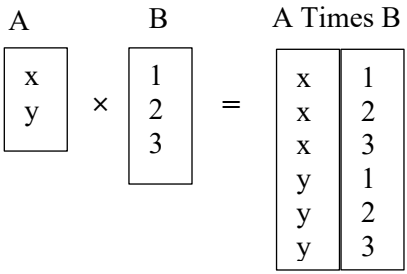


Рис. 5.3. Схематичне подання результату виконання декартового добутку

Відношення-операнди, що беруть участь в операції декартового добутку не можуть мати однакові заголовки або спільні атрибути. Цю операцію використовують для відношень, що взаємодіють за принципом «багато-до-багатьох».

Реалізація декартового добутку мовою SQL має вигляд:

```
Select A.*, B.* From A, B;
```

Приклад 2. Визначити максимальне залучення усіх акторів у наведених фільмах. Для спрощення результату виконання з двох таблиць обрано по одному атрибуту.

```
Select actor_name, film_title
From Actors, Films;
```

Операція декартового добутку виконується над таблицями Actors та Films, які безпосередньо не пов'язані між собою, та не мають однакових атрибутів. Результуюче відношення міститиме два атрибути, зазначені після `Select`. У тілі відношення ім'я кожного актора повторюватиметься стільки разів, скільки фільмів є у таблиці Films, як схематично зображено на рис. 5.3.

Спеціальні реляційні операції

❖ ***Обмеження (вибірка)*** – це спеціальна операція реляційної алгебри, яка працює з одним відношенням A і визначає результуюче відношенням із тим самим заголовком і тілом, що містить кортежі відношення A, які задовольняють задану умову.

Фактично, це операція вибірки, яка у позначенням мови SQL записується як:

Select * From A Where XΘY,

де Θ – це умова вибірки за участі одного або кількох атрибутів відношення А.

Приклад 3. Використання простої умови: вибрати повну інформацію про акторів зі Сполучених Штатів Америки.

```
Select *
From Actors
Where actor_country='США';
```

Приклад 4. Об'єднання кількох умов у логічний вираз: одержати дані про фільми у жанрі драми, що вийшли у прокат після 1995 року.

```
Select *
From Films
Where release_year>1995 AND genre='Драма',
```

❖ ***Проекція*** виконується над одним відношенням і визначає нове відношення з набором атрибутів, по яких виконується проекція, за винятком кортежів-дублікатів.

Нехай задане відношення А з певним набором атрибутів: (A{X,Y,…, Z}). Тоді проекцією відношення А за атрибутами X, Y, Z (A[X,Y,Z]) називається відношення, що задовольняє таким вимогам:

- Заголовок результуючого відношення одержується із заголовка А шляхом видалення з нього всіх атрибутів, що не входять до множини {X,Y, Z}.

- Тіло відношення-результату містить множину всіх кортежів вигляду {X:x, Y:y, Z:z}, де x, y, z – значеннями відповідних атрибутів вихідного відношення А.

Загалом проекція мовою SQL виконується як:

Select Distinct X, Y, Z From A;

Тут Distinct використовується саме для усунення рядків-дублікатів.

Приклад 5. Проекція відношення Films, тобто вибірка даних за назвою кінофільму та роком його випуску.

```
Select Distinct film_title, release_year From Films;
```

❖ **Упорядкування даних**. За визначенням відношення його кортежі невпорядковані. Проте при виконанні оператора вибірки подекуди зручніше подавати результати у відсортованому вигляді. Тобто дані запиту можна впорядкувати відносно одного або навіть кількох полів. Для цього в кінці оператора `Select` слід застосувати оператор SQL **`Order By`**:

`Order By` назва_поля **`[ASC | DESC]`**

Тут необов'язкові параметри **`ASC`** і **`DESC`** визначають спосіб сортування за зростанням і за спаданням відповідно. Причому поле, відносно якого відбуватиметься упорядкування, може бути не лише числовим, але й рядком або датою. Якщо жоден параметр не зазначено, кортежі впорядковуватимуться за зростанням (**`ASC`**).

Приклад 6. Вивести дані про акторів, імена яких впорядковані за алфавітом.

```
Select *
From Actors
Order By actor_name;
```

Якщо при виконанні вибірки потрібно, щоб до результуючої таблиці потрапило N рядків, можна використати команду **`Limit`** у форматі:

`Select * From` A **`Limit`** N;

Наприклад, якщо потрібно вивести дані лише перших трьох акторів із впорядкованого переліку з Прикладу 6:

```
Select * Limit 3
From Actors
Order By actor_name;
```

До того ж запит можна доповнити зміщенням (**`Offset`**), зазначивши кількість рядків, після яких виводитимуться дані:

```
Select actor_id, actor_name Limit 3 Offset 2
From Actors
Order By actor_name;
```

Тут результат виконання міститиме три записи з номерами actor_id 3, 6 і 2, до яких не потрапили перші два актори з відсотрованого переліку – Б. Кембербетч та Дж. Депп.

У багатьох запитах доводиться поєднувати операції обмеження та проекції. Розглянемо приклади такого застосування.

❖ **Діапазон значень** можна задавати за допомогою логічного виразу за участі двох умов або предиката **Between**.

Приклад 7. Одержати назви фільмів, що вийшли у прокат з 1995 по 2010 роки. Обидві умови повинні справджуватися, аби загальний логічний вираз дав результат true.

```
Select film_title
From Films
Where release_year>=1995 AND release_year<=2010;
```

Приклад 8. Визначення меж діапазону з попереднього прикладу з використанням Between.

```
Select film_title
From Films
Where release_year Between 1995 AND 2010;
```

Слід зазначити, що для використання предиката **Between**, значення, що позначають межі діапазону, повинні мати однаковий тип та належати одному домену. Окрім числових значень, граничні дані можуть бути типу дати та часу або навіть символьними величинами.

❖ **Перелік значень** для пошуку можна визначати через кілька умов рівності, об'єднаних за допомогою логічної операції **OR**, або з використанням предиката **IN**. Для виконання умови пошуку достатньо, щоб справдилася принаймні одна умова.

Приклад 9. Одержати назву та рік випуску фільмів у жанрі Драма або Комедія.

```
Select  film_title, release_year
From Films
Where genre='Драма' OR genre='Комедія';
```

АБО

```
Select  film_title, release_year
From Films
Where genre IN ('Драма', 'Комедія');
```

Зауважте, що символьні значення в умові пошуку записуються в одинарних лапках саме так, як їх було внесено до таблиці (зокрема, у назві жанру перша літера – велика). Будь-яка відмінність у написанні призведе до невиконання умови.

Для створення протилежних умов за участі предикатів можна застосовувати операцію логічного заперечення **NOT** у форматі
NOT Between або **NOT IN**.

❖ **Пошук пропусків.** Поля, до яких явно не введено дані, набувають значення Null, тобто залишаються порожніми. Подекуди виникає потреба вивести інформацію з рядків із відсутніми даними у деякому полі. Порожні поля не заповнені нулями або символами пробілу, які можна було б використати в умові пошук. Для створення умови з пошуку або відсутності порожніх значень використовується логічна функція **IS NULL** / **IS NOT NULL** відповідно.

Приклад 10. Вивести номер та ім'я акторів, для яких немає даних про дату народження.

```
Select actor_id, actor_name
From Actors
Where actor_bday IS NULL ;
```

❖ **Шаблон пошуку.** При роботі з символьною інформацією у базі даних подекуди потрібно використовувати в умові певний шаблон. На відміну від операцій порівняння, які вимагають точного і повного значення, шаблон може містити лише фрагмент вихідного рядка, доповнений символами підстановки:

- символом нижнього підкреслення '_' позначає один пропущений символ у рядку пошуку;
- символ відсотка '%' заміщує послідовність з нуль або більше символів у рядку.

Для роботи з шаблонами після `Where` використовується оператор **Like**.

Приклад 11. Вивести повну інформацію про кіногероїв, ім'я або прізвище яких починається на «Дж».

```
Select *
From Roles
Where hero_name Like '%Дж%';
```

❖ ***З'єднання відношень*** **–** це ще одна спеціальна операція реляційної алгебри, яка виконується над двома відношеннями. При з′єднанні відношень за деякою умовою утворюється результуюче відношення, кортежі якого є конкатенацією (злиттям) кортежів першого і другого відношень, для яких умова з'єднання справджується.

Нехай з'єднуються відношення A і B, що мають спільні атрибути. Тоді Θ-з'єднання відношення A за атрибутом X з відношенням B за атрибутом У позначається виразом:

A[XΘY] B або A Join B Where XΘY

При роботі з базами даних найчастіше доводиться мати справу з операцією **еквіз'єднання**: коли умовою з'єднання є рівність двох атрибутів відношень A і B: A[X=Y]B .

Традиційно одне відношення з'єднується з іншим за спільним атрибутом, зокрема первинним та зовнішнім ключем. Аби розрізнити однойменні атрибути в умові з'єднання, перед їх назвою через крапку зазначають ім'я таблиці: A Join B Where A.c=B.c (рис. 5.4).

A (батьківська)

c	d
1	d1
2	d2
3	d3

B (дочірня)

e	**c**	f
i	**1**	x
j	**1**	y
k	**3**	x

A [A.C=B.C] B

e	c	f	d
i	1	x	d1
j	1	y	d1
k	3	x	d3

Рис. 5.4. Схематичне зображення виконання операції внутрішнього еквіз'єднання

Операція з'єднання буває двох типів: внутрішнє та зовнішнє.

Внутрішнє з'єднання (**`Inner Join`**) використовується для доповнення дочірньої таблиці інформацією з батьківської. Результат виконання містить лише ті рядки, в яких зустрічаються однакові значення у спільних полях. Схематично результат внутрішнього з'єднання зображений на рис. 5.4. Так, у спільному

полі **c** дочірньої таблиці B немає запису зі значенням 2, а отже, до результату не потрапляє відповідний кортеж з таблиці A.

Мовою SQL операцію внутрішнього з'єднання можна реалізувати двома способами у форматі:

```
Select A.*, B.*
From A, B
Where A.c=B.c;
```

АБО

```
Select A.*, B.*
From A Inner Join B
ON A.c=B.c;
```

Приклад 12. Вивести назви ролей та імена акторів, які їх виконали.

```
Select hero_name, actor_name
From Actors, Roles
Where Actors.actor_id=Roles.actor_id;
```

АБО

```
Select hero_name, actor_name
From Actors Inner Join Roles
ON Actors.actor_id=Roles.actor_id;
```

Результат виконання запиту матиме вигляд:

hero_name	**actor_name**
Джек Горобець	Джонні Депп
Дріс	Омар Сі
Алан Тюрінг	Бенедикт Кембербетч
Елізабет Свон	Кіра Найтлі
Джоан Кларк	Кіра Найтлі
Пол Еджкомб	Том Генкс

Зауважте, що до результуючої таблиці потрапили імена не всіх акторів, а лише тих, чиї номери були зазначені у полі зовнішнього ключа (actor_id) таблиці Roles.

Зовнішнє з'єднання (**Outer Join**) дозволяє долучити до батьківської таблиці дані з дочірньої (рис. 5.5). Операція зовнішнього з'єднання позиційна, тобто залежить від того, до

якої таблиці приєднуються рядки і яка таблиця є тією, що приєднується.

A (батьківська)

C	D
1	a
2	b
3	c

B (дочірня)

E	**C**	F
i	**1**	x
j	**1**	y
k	**3**	x

A [A.C*=B.C] B

C	D	E	F
1	a	i	x
1	a	j	y
2	b	NULL	NULL
3	c	k	x

Рис. 5.5. Схематичне зображення виконання операції зовнішнього лівостороннього з'єднання

Зовнішнє з'єднання буває ліво- (**`Left Outer Join`**) і правостороннім (**`Right Outer Join`**), коли основна таблиця розташовується, відповідно, зліва або справа.

Загальний формат запиту мовою SQL:

```
Select A.*, B.*
From A Left | Right Outer Join B
ON A.c=B.c;
```

Приклад 13. Доповнити інформацію про фільми іменами кіногероїв. Результат виконання запиту матиме вигляд:

film_id	film_title	release_year	hero_name
1	Пірати Карибського моря	2003	Джек Горобець
1	Пірати Карибського моря	2003	Елізабет Свон
2	Гра в імітацію	2014	Алан Тюрінг
2	Гра в імітацію	2014	Джоан Кларк
3	Втеча з Шоушенка	1994	NULL
4	Зелена миля	1999	Пол Еджкомб
5	Недоторканні	2011	Дріс

```
Select Films.film_id, film_title, release_year, hero_name
From Films Left Outer Join Roles
ON Films.film_id=Roles.film_id;
```

Як бачимо, при зовнішньому лівосторонньому з'єднанні основною є батьківська таблиця Films, яка доповнюється інформацією про героїв фільмів з дочірньої таблиці Roles. Зауважте, що для кожного персонажу одного фільму додається окремий кортеж, як у випадку фільмів з номерами 1 та 2. Очевидно, що, окрім порядкового номера, повторюються решта даних про фільм. Саме для уникнення такого дублювання інформацію про кіногероїв, акторів та фільми було розподілено по різних таблицях.

До того ж для фільму з номером 3 поле hero_name порожнє, оскільки серед кіногероїв немає інформації стосовно саме цього фільму. Тобто для невідповідних значень рядків лівої таблиці долучаються порожні значення полів правої таблиці.

Ще раз наголосимо про важливість послідовності розташування таблиць при виконанні зовнішнього з'єднання. *Якщо у **Прикладі 13** поміняти місцями таблиці Films та Roles, результат операції буде аналогічний внутрішньому з'єднанню.*

Окрім умови з'єднання оператори можна доповнювати додатковими предикатами.

Приклад 14. Знайти акторів, для яких немає інформації щодо зіграних ролей. Застосуємо правостороннє з'єднання, де батьківська таблиця Actors розташується праворуч, і відфільтруємо кортежі з порожнім полем hero_name.

```
Select Actors.actor_id, actor_name, hero_name
From Roles Right Outer Join Actors
ON Roles.actor_id=Actors. actor_id
Where hero_name is Null;
```

Результат виконання:

actor_id	actor_name	hero_name
6	Одрі Тоту	NULL

Аналогічний результат можна одержати, якщо переставити місцями таблиці та застосувати лівостороннє з'єднання. Проте внутрішнє з'єднання не дозволить виконати цю операцію, через те що не може повернути рядки, для яких немає значень у спільних полях.

Створіть запит, який виведе дані про фільм, у якому одного з героїв звали Алан Тюрінг.

Агрегатні функції. Групування

За допомогою запиту, крім пошуку та виведення окремих значень полів, можна одержувати узагальнене групове значення у межах деякого поля. Для цього використовуються **агрегатні функції.**

Агрегатні функції SQL діють по відношенню до значень одного поля, назва якого передається до функції як аргумент у круглих дужках, з метою одержання єдиного підсумкового результату для таблиці або групи. Нижче наведено найпоштреніші агрегатні функції.

- **`COUNT`** – визначає кількість рядків таблиці або не-Null значень поля.
- **`SUM`** – знаходить арифметичну суму всіх значень поля.
- **`AVG`** – визначає середнє арифметичне для усіх значень деякого поля.
- **`MAX`** – знаходить найбільше серед усіх значень поля.
- **`MIN`** – визначає найменше з усіх вибраних значень поля.

Агрегатні функції застосовується подібно до імен полів у команді `Select`:

`Select` агр_функц1(поле) [, агр_функц2(поле), … поле]

`From` назва_таблиці

[`Where` |**`Having`** умова_вибірки]

[**`Group By`** поле]

Вирізняють два способи використання агрегатних функцій. Перший: агрегатні функції застосовуються як самостійні об’єкти і повертають одне результуюче значення для поля. Другий: агрегатні функції використовуються у запиті разом з оператором **`Group By`** з групуванням по полях для отримання підсумкового значення для кожної групи. Спершу розглянемо приклад застосування агрегатної функції без групування.

Приклад 15. Знайти загальну кількість акторів.

```
Select Count(actor_id) as actor_number
From Actors;
```

Результат:

actor_number
6

Результат, хоча і складається з одного значення, проте подається у вигляді таблиці. Щоб поле цієї таблиці не залишалося безіменним, після **as** для нього визначили умовну назву.

Приклад 16. Знайти кількість французьких акторів.

```
Select Count(actor_id) as french_actors
From Actors
Where actor_country='Франція';
```

Результат:

french_actors
2

Отже, за допомогою умови можна обмежувати набір значень, що оброблятимуться функцією.

Приклад 17. Скільки жанрів кіно представлено у базі даних?

```
Select Count(Distinct genre)
as film_genre
From Films;
```

Результат:

film_genre
4

Використання `Distinct` перед назвою аргумента функції забезпечить врахування при підрахунку лише унікальних значень.

Оператор SQL **Group By** слугує для розподілення рядків таблиці по групах відносно однакових значень стовпця, по якому відбувається групування.

Наприклад, рядки таблиці Actors можна умовно згрупувати за країною:

actor_id	actor_name	actor_bday	actor_country
1	Джонні Депп	1963-06-09	США
5	Том Генкс	1956-07-09	США
3	Кіра Найтлі	1985-03-26	Велика Британія
4	Бенедикт Кембербетч	1956-07-09	Велика Британія
2	Омар Сі	1978-01-20	Франція
6	Одрі Тоту	1976-08-09	Франція

Як бачимо, акторів у таблиці можна розподілити по трьох групах за країною походження: група акторів з США, Великої Британії та Франції.

Які групи можна виокремити у таблицях Films та Roles?

Якщо в запиті потрібно вивести значення всіх наявних груп, по стовпчику, відносно якого сортуються кортежі, виконується групування за допомогою оператора **`Group By`**. При цьому до результату запиту потраплятимуть лише унікальні значення цього поля, аналогічно застосуванню `Distinct`.

Приклад 18. Знайти країни походження акторів.

```
Select actor_country
From Actors
Group By actor_country;
```

Результат:

actor_country
США
Велика Британія
Франція

Отже, поєднавши групування з агрегатними функціями, можна одержати узагальнене значення поля для кожної групи.

Приклад 19. Знайти кількість акторів, що походять з кожної країни.

```
Select actor_country, Count (actor_id) as “К-ть акторів”
From Actors
Group By actor_country;
```

Зауважте, що умовну назву поля або результат агрегатної функції слід брати у лапки, якщо вони містять пробіли або символи, відмінні від латиниці.

Результат:

actor_country	К-ть акторів
США	2
Велика Британія	2
Франція	2

Як бачимо, тепер агрегатна функція працює не в межах всієї таблиці, а окремої групи – рахує кількість акторів у кожній країні.

Зазвичай в умові запиту поле, по якому слід групувати дані, супроводжують слова «кожен», «окремий» та ін.

Як у таблиці **Films** *визначити кількість фільмів, представлених у* ***кожному*** *жанрі?*

Приклад 20. Знайти, у скількох фільмах зіграв **кожен** актор.

```
Select actor_id, Count (film_id) as "К-ть фільмів"
From Roles
Group By actor_id;
```

Результат:

actor_id	К-ть фільмів
1	1
2	1
3	2
4	1
5	1

Як визначити, скільки акторів зіграло в кожному фільмі?

Зазвичай агрегатні функції не можуть бути вкладеними, а отже результат однієї функції не може стати аргументом іншої. Проте, це завдання можна вирішити, поєднавши групування з іншими директивами мови SQL, як-от `Order By` та `Limit`.

Приклад 21. Вивести номер актора, який зіграв у найбільшій кількості фільмів. Цей запит можна виконати, впорядкувавши

номери акторів за спаданням кількості зіграних ролей і обмеживши результат до одного запису:

```
Select actor_id, Count (film_id) Limit 1
From Roles
Group By actor_id
Order By Count(film_id) DESC;
```

Результати запитів з групуванням також можна відфільтровувати. Для цього застосовуються умови за участі двох зарезервованих слів: **Where** та **Having**:

- Умови з **Where** застосовуються до окремих значень і дозволяють відібрати дані, які опрацьовуватиме агрегатна функція.
- Умови з **Having** використовуються для вибірки одержаних групових значень. Тобто коли **в умові** використовується ***агрегатна функція*** або ***поле, за яким виконується групування***.

Приклад 22. Знайти кількість фільмів кожного жанру, знятих після 2000 року.

```
Select genre, Count(film_title)
From Films
Group By genre
Where release_year >2000;
```

Приклад 23. Знайти номери фільмів, для яких є інформація про щонайменше двох головних героїв.

```
Select film_id
From Roles
Group By film_id
Having Count(actor_id) >=2;
```

Приклад 24. Знайти кількість фільмів, у яких зіграли актори з номерами від 2 до 4.

```
Select actor_id, Count(film_id)
From Roles
Group By actor_id
Having actor_id Between 2 And 4;
```

Звісно, що зручніше працювати не з номерами, а з іменами героїв або з назвами фільмів. Для цього можна поєднати операції з'єднання таблиць та групування.

Приклад 25. Вивести номери та імена акторів, а також кількість фільмів, у яких вони зіграли, згідно з інформацією у базі даних.

```
Select Actors.actor_id, actor_name, Count(film_id)
As film_count
From Actors Left Outer Join Roles
ON Actors. actor_id = Roles.actor_id
Group By actor_id, actor_name;
```

Результат:

actor_id	actor_name	film_count
1	Джонні Депп	1
2	Омар Сі	1
3	Кіра Найтлі	2
4	Бенедикт Кембербетч	1
5	Том Генкс	1
6	Одрі Тоту	NULL

Як визначити кількість ролей для кожного фільму?

Запит у Прикладі 25 можна доповнювати умовами вибірки, зазначивши після `Group By`:

```
Having Count(film_id) >=2
```

Як зміниться результат виконання запиту? Як записати умови вибірки для акторів, для яких немає даних про фільми?

ЛАБОРАТОРНА РОБОТА № 5
Створення типових запитів до бази даних

Мета

Набути навичок заповнення таблиць і виконання базових запитів мовою SQL у СУБД Microsoft SQL Server 20x.

ПРАКТИЧНЕ ЗАВДАННЯ

Після перевантаження моделі бази даних до СУБД MS SQL Server необхідно заповнити створені таблиці.

1. Заповнення таблиць БД

1.1. Увійдіть до середовища **Microsoft SQL Server Management Studio**.

1.2. У вікні **Object Explorer** (Провідник об'єктів) оберіть створену базу даних. Розгорніть перелік таблиць, з яких вона складається.

1.3. Внесення даних потрібно розпочати з базових (батьківських) таблиць, а лише після їх заповнення переходити до таблиць, які від них залежать.

1.4. До кожної таблиці внесіть від 5 до 10 **ЗМІСТОВНИХ** даних.

1.5. Аби внести дані до таблиці, клацніть на ній правою кнопкою миші та у контекстному меню оберіть пункт **Edit Top 200 Rows** (Редагувати перші 200 рядків). З'являється порожня таблиця з переліком визначених полів.

1.6. Стежте за відповідністю значень пов'язаних полів первинного та зовнішнього ключа.

1.7. При закритті вікна дані в таблиці зберігатимуться автоматично.

2. Створення запитів

2.1. Для створення запитів та збереження результатів їх виконання скористаємося ще одним компонентом бази даних під назвою **View** (Вигляд/Представлення) (рис. 5.6).

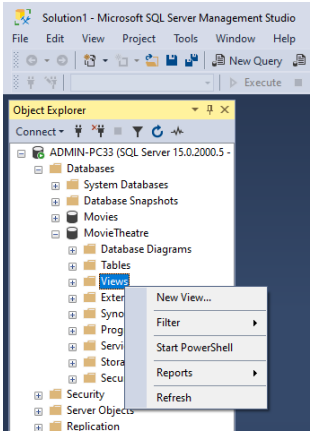


Рис. 5.6. Компонент бази даних – View

2.2. Клацніть правою кнопкою миші по компонентові **View** та оберіть у контекстному меню пункт **New View**.

2.3. Як подано на рис. 5.7, з'являється вікно з переліком таблиць, які входять до складу бази даних. Оберіть таблицю, до якої потрібно створити запит і натисніть кнопку **Add.**

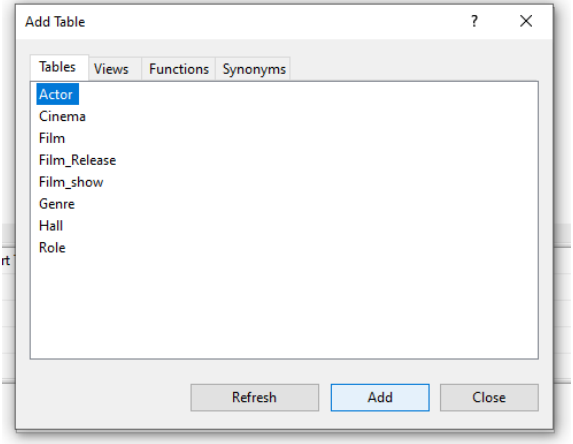


Рис. 5.7. Створення нового представлення (View) для виконання запиту

2.4. Обрана таблиця з переліком усіх її атрибутів додається до верхнього вікна. Як бачимо на рис. 5.8, в окремих вікнах відображається шаблон для створення запитів, який можна модифікувати та доповнювати. Готовий запит можна запустити за допомогою кнопки на панелі інструментів або натиснувши CTRL+R. Результат виконання запиту відображається у нижньому вікні.

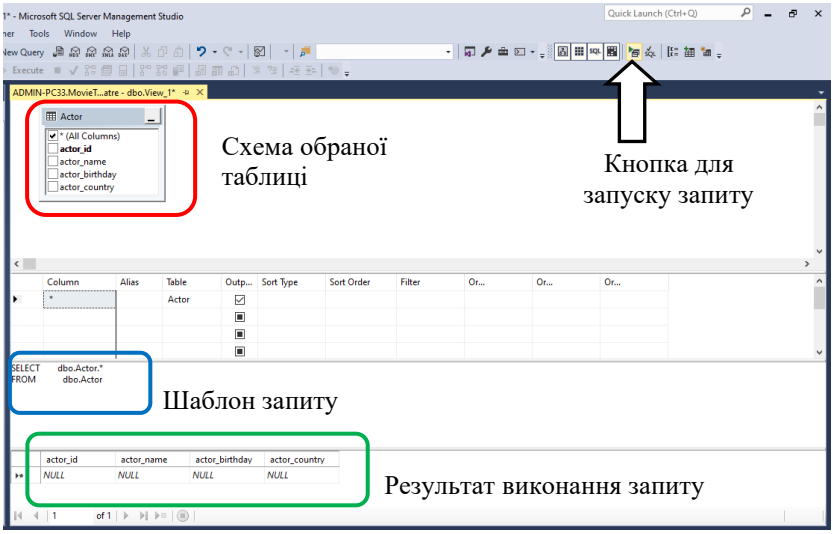


Рис. 5.8. Виконання запиту в представленні

2.5. При закритті поточного View з'явиться вікно із пропозицією зберегти Представлення. Натисніть Yes та збережіть представлення зі змістовною назвою. Отже, кожен запит можна зберігати у окремому View під унікальним іменем і переглядати у **Object Explorer**.

2.6. Створіть наведені нижче типи запитів та збережіть результати їх виконання у представленнях. **Запити повинні бути змістовниими та виконувати деяку корисну дію над даними у базі.**

1. Вибірка всього вмісту таблиці.
2. Об'єднання даних двох таблиць (`Union`).
3. Виконання декартового добутку.

4. Зміна порядку розташування полів таблиці при вибірці.
5. Проекція.
6. Впорядкування результатів запиту відносно деякого поля.
7. Усунення дублікатів при вибірці даних з одного поля.
8. Обмеження (вибірка даних за умовою).
9. Вибірка даних за участі кількох умов, об'єднаних разом у логічний вираз.
10. Задання діапазону в умові вибірки (два способи).
11. Визначення значень в умові вибірки (два способи).
12. Створення шаблону пошуку за допомогою оператора `Like`.
13. Застосування хоча б двох агрегатних функцій.
14. Групування даних у таблиці.
15. Створення умови для групового значенні за допомогою директиви `Having`.
16. Виконання внутрішнього з'єднання кількох таблиць (`Inner Join`).
17. Виконання зовнішнього з'єднання для таблиць з п. 16 (`Outer Join`). *Порівняйте та поясніть результати.*
18. Додавання нового запису до таблиці.
19. Видалення запису.
20. Оновлення вже введених даних.

Зауважимо: *для запитів з п. 18-20 результат не зберігається у вигляді* ***View****, його можна переглянути після оновлення таблиці.*

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. *В обраній СУБД заповніть таблиці змістовними та різноманітними даними, що відповідають предметній області створеної БД.*

2. Реалізуйте базові запити із запропонованого у п. 2.6 переліку. Переконайтесь, що запит виконано успішно.

3. SQL-скрипт та результат виконання кожної операції збережіть в окремому представленні.

Звіт з лабораторної роботи повинен містити:

1. Скрипт кожної SQL-операції.
2. Пояснення дії, яку виконує запит.
3. Екранні форми результату виконання запиту.
4. Відповіді на запитання для самоперевірки.

Для захисту лабораторної роботи необхідно продемонструвати одержані результати у СУБД, подати оформлений звіт та бути готовими відповісти на запитання з переліку нижче.

Запитання та завдання для самоперевірки

1. На чому ґрунтується аспект обробки реляційної моделі даних?
2. У чому полягає реляційна замкненість операцій реляційної алгебри?
3. Чим відрізняються операції об’єднання та з’єднання? Які їх передумови та результат виконання?
4. Поясніть, як виконується операція декартового добутку та як її реалізувати мовою SQL.
5. Як поєднати в одному запиті операції обмеження та проекції?
6. Назвіть відомі вам способи вибірки даних за декількома значеннями.
7. Для чого може знадобитися пошук порожніх значень?
8. Який вигляд матиме шаблон пошуку користувачів електронної пошти ukr.net? Як створити шаблон для пошуку значень фіксованої довжини або формату?
9. За яким принципом виконуються операції внутрішнього та зовнішнього з’єднання? Що між ними спільного та чим вони відрізняються? Яка мета кожної з них?
10. Посніть, чому використання агрегатних функції доцільніше за впровадження до таблиць окремих атрибутів для зберігання кількості, загальної суми,

середнього, мінімального або максимального значення поля.

11. Яку корисну функцію при створенні запитів виконує оператор `Group By`?
12. Як визначити, що обрати для створення умови вибірки: `Where` чи `Having`?

ТЕМА 6. Розширені можливості мови SQL

Функції для роботи з датою та часом

У мові SQL визначено цілу низку спеціальних функцій для обробки числових даних, рядків, а також показників дати та часу. Саме останні потребують особливої обробки, оскільки поля типу `datetime` зберігають дані у форматі РРРР-ММ-ДД гг:хх:сс[.ннн]. Подекуди доводиться працювати окремо з датою, а окремо – з часом, визначати день, місяць, рік, порівнювати часові межі. Розглянемо приклади застосування нових можливостей мови SQL для таблиць бази даних «Мережа кінотеатрів», зображених на рис. 5.1.

Для роботи з датою та часом корисними можуть бути такі функції:

`Getdate()` – визначення поточної дати та часу на основі системного годинника у вигляді об'єкта типу `datetime`.

Наприклад: `Select Getdate()`

`Day/Month/Year``(об'єкти типу datetime)` – повернення дня, місяця або року для параметра-дати.

Приклад 1. Обрати дані про фільми, випущені понад 20 років тому.

```
Select film_title, release_year
From Films
Where Year(Getdate()) - release_year>20;
```

`Datename``(параметр, об'єкти типу datetime)` – повертає частину дати у вигляді *рядка* (назви). Як параметр можуть використовуватися такі зарезервовані слова: `year` (рік), `quarter` (квартал), `month` (назва місяця), `day` (день), `weekday` (назва дня тижня), `hour` (година), `minute` (хвилина) тощо.

Приклад 2. Визначити назву місяця, в якому народилися актори.

```
Select actor_name, Datename(month, actor_bday)
From Actors;
```

`Datepart``(year/month/day, об'єкти типу datetime)` – повертає з дати рік, місяць або день у вигляді *числа*.

Приклад 3. Вивести дані про акторів, що народилися у липні.

```
Select actor_name, actor_bday
From Actors
Where Datepart(month, actor_bday)=7;
```

Datediff (параметр, дата1, дата1) – повертає різницю між двома датами в одиницях, що визначаються першим параметром-інтервалом (див. функцію Datename()).

Приклад 4. Вивести повну інформацію про акторів, віком старших за 40 років.

```
Select *
From Actors
Where Datediff(year, actor_bday, Getdate())>40;
```

Вкладені запити

Оскільки в результаті операції реляційної алгебри генерується таблиця, її можна застосовувати всередині інших операторів. За допомогою SQL можна вкладати запити всередину один одного. Зазвичай внутрішній запит генерує одне або кілька значень, які передаються до логічного виразу зовнішнього запиту та впливають на результат його виконання.

Вкладений запит необхідно записувати в круглих дужках у правій частині оператора порівняння зовнішнього запиту.

Узагальнено синтаксис оператора вибірки за участі вкладеного запиту має такий вигляд:

```
Select перелік_полів
From таблиця
Where вираз оператор (Select перелік_полів
                      From таблиця
                      Where умова);
```

Вкладені запити можуть повертати як одне значення, так і цілу низку.

Для обробки результатів підзапитів, що повертають рівно одне значення одного поля, використовуються оператори порівняння: =; >;>=;<;<=; !=.

При роботі з підзапитами, які повертають кілька значень, застосовують такі **оператори**:

- **IN** – для порівняння зовнішнього виразу зі значеннями з переліку, який повертає підзапит.
- **ANY** – порівняння виконується з усіма значеннями, одержаними з підзапиту, і повертає true у разі збігу з будь-яким ОДНИМ значенням зі списку.
- **ALL** – порівнює значення виразу з усіма елементами підзапиту і повертає true, якщо умова справджується для ВСІХ значень зі списку.
- **EXISTS** (**NOT EXISTS**) – не потребує умови та використовується для перевірки того, чи повертає внутрішній запит хоча б одне значення (результат виконання порожній).

Оператори `BETWEEN`, `LIKE`, та `IS NULL` із підзапитами не використовуються.

Подекуди вкладені запити використовують як альтернативу операторам з'єднання, проте повністю ототожнювати їх неможна. Розглянемо приклади виконання вкладених запитів для таблиць бази даних «Мережа кінотеатрів», зображених на рис. 5.1.

Приклад 5. Вивести повну інформацію про актора, що виконав роль Джека Горобця.

Оператор вибірки для виконання цього завдання за участі вкладеного запиту матиме вигляд:

```
Select *
From Actors
Where actor_id=(Select actor_id
                From Roles
                Where  hero_name='Джек Горобець');
```

Для виконання зовнішнього (основного) запиту SQL спочатку повинен оцінити внутрішній запит, розміщений у круглих дужнах у правій частини умови після `Where`. Зокрема, проаналізувати в таблиці Roles всі рядки, у яких поле hero_name набуває значення Джек Горобець, а потім як результат вилучити з відповідних рядків номер актора (actor_id). Як бачимо, єдиним виконавцем цієї ролі виявився актор із номером 1. Тому SQL розміщує знайдене значення до логічного виразу основного

запиту замість підзапиту, так щоб умова набула вигляду **Where actor_id = 1**. А далі основний запит повертає з таблиці Actors повну інформацію, яка відповідає актору з номером 1.

Результат виконання запиту:

actor_id	actor_name	actor_bday	actor_country
1	Джонні Депп	1963-06-09	США

Альтернативно, такий запит можна виконати за допомогою операції внутрішнього з'єднання, проте, на відміну від попереднього варіанту, результат можна доповнити даними з таблиці Roles:

```
Select Actors.*, hero_name, film_id
From Actors, Roles
Where Actors.actor_id=Roles.actor_id
And hero_name='Джек Горобець';
```

Підзапит – гарний варіант, коли потрібно здійснити пошук у таблиці відносно її власних даних.

Приклад 6. Вивести дані про фільми, зняті в тому самому жанрі, що і «Втеча з Шоушенка».

```
Select *
From Films
Where genre=(Select genre
                From Films
                Where film_title=' Втеча з Шоушенка');
```

film_id	film_title	release_year	genre
3	Втеча з Шоушенка	1994	Драма
4	Зелена миля	1999	Драма

Коли вкладений запит повертає більше ніж одне значення, для створення умови використовується оператор **IN**. До того ж один запит може містити кілька вкладених.

Приклад 7. Вивести повну інформацію про фільми, в яких зіграла актриса Кіра Найтлі.

```
Select *
From Films
Where film_id IN
```

```
(Select film_id
 From Roles
  Where actor_id= (Select actor_id
                   From Actors
                   Where actor_name=’Кіра Найтлі’));
```

При виконанні цього запиту спершу в таблиці Actors буде визначено номер актора, що відповідає імені. Далі в таблиці Ролі буде знайдено номери фільмів, для яких зазначено виявлений номер. Нарешті, виводитиметься повна інформація про фільми за їх номерами. Оскільки фільмів може бути більше ніж один, в умові зовнішнього запиту використано оператор **IN**.

Забезпечити вироблення підзапитом одиночного значення для будь-якої кількості рядків можна за допомогою агрегатної функції. Будь-який запит, що використовує одиночну агрегатну функцію без пропозиції `Group By` повертатиме одиночне значення, придатне для подальшого використання в основному логічному виразі.

Приклад 8. Вивести дані про акторів, які зіграли більше ніж в одному фільмі.

```
Select actor_id, actor_name, actor_country
From Actors as A
Where 1<( Select Count(*)
                  From Roles as R
                  Where A.actor_id=R.actor_id);
```

Тут A, R – умовні скорочені позначення основних таблиць.

При виконанні цього запиту застосовується така процедура:

1. Вибирається рядок з таблиці, зазначеної у зовнішньому запиті. Назвемо його поточним рядком-кандидатом.
2. Значення цього рядка-кандидата зберігається у буфері.
3. Виконується підзапит. Для добору записів, що братимуть участь у функції `Count`, перевіряється умова за участі значення з рядка-кандидата.
4. Умова зовнішнього запиту даватиме результат true, якщо вкладений запит поверне 2 і більше значення, і саме виконання умови визначатиме, чи потрапить рядок-кандидат до остаточного результату.

5. Така процедура повторюється для всіх наступних рядків таблиці в зовнішньому запиті.

Як бачимо, процес вибірки за допомогою підзапиту послідовний і виконується в кілька етапів. Тому, коли можливо, варто застосовувати операції з'єднання пов'язаних таблиць.

Оператори **ANY** та **ALL** використовуються разом з умовою для порівняння значення у зовнішньому запиті зі значеннями, що повертає підзапит за таким принципами:

- Використання = разом з **ANY** еквівалентне оператору **IN**.
- При застосуванні умови **значення < ANY** обиратиметься значення, менше за мінімальне зі значень у переліку, що повертає внутрішній запит.
- Використання **> ANY** обиратиме значення, більші за максимальне зі значень, одержаних у підзапиті.
- При використанні **ALL** умова справджується, якщо *кожне* значення, обране підзапитом, задовольняє умові зовнішнього запиту.

Трапляються випадки, коли неможливо звести умову вибірки до якогось конкретного значення, або внутрішній запит може повертати значення не одного поля, а рядки зі значеннями кількох атрибутів. При цьому для створення підзапиту може знадобитися застосування предиката **EXISTS**.

EXISTS (англ. «існує») – це оператор, який використовує підзапит як аргумент і повертає результат true, якщо той повертає хоча б одне значення, і як false, якщо виконання підзапиту не містить жодного значення. Відповідно, оператор **NOT EXISTS** дає позитивний результат, коли підзапит не виводить жодного значення.

Приклад 9. Вивести з таблиці Ролі номери акторів, які грали в одному фільмі.

```
Select Distinct actor_id
From Roles as outerR
Where EXISTS (Select *
                  From Roles as innerR
                  Where outerR.film_id=innerR.film_id
                  And outerR.role_id!=innerR.role_id);
```

Для кожного рядка-кандидата зовнішнього запиту (позначений номером актора, який наразі перевірятиметься) внутрішній запит знаходить рядки, у яких значення поля actor_id та film_id збігаються зі значеннями у рядку-кандидаті. Якщо внутрішній запит знайде принаймні один такий рядок, умова основного запиту справдиться для поточного рядка-кандидата, а відповідний йому номер актора потрапить до результату. Якщо не використовувати `Distinct`, номер актора виводився би стільки разів, у скількох фільмах він брав участь.

Збережені процедури

Збережена процедура (англ. *stored procedure*) – це іменований об'єкт бази даних, який містить фрагмент коду з власною логікою, що визначає операції над таблицями у вигляді набору SQL-інструкцій. Інакше кажучи, це функція або програма всередині бази даних. Збережені процедури використовуються для зберігання на сервері коду, який можна багаторазово запускати на виконання. При створенні запиту необхідно прописувати поля, до яких іде звернення, значення, що фігурують в умовах вибірки, та власне операції над даними. При цьому неодноразово доводиться зазначати назви полів та таблиць. Щоб не відтворювати запит щоразу, коли його потрібно застосувати, можна оформити SQL-код відповідної операції у вигляді збереженої процедури. При цьому її скомпільований код зберігається на сервері і запускається на виконання при виклику процедури.

Такий іменований блок операцій можна багаторазово викликати у різних частинах коду програмного застосунку, через який відбувається взаємодія з базою даних, з довільними значеннями параметрів. Збережені процедури можна створювати для найбільш поширених операцій мови SQL, як-от `Select, Update, Delete, Insert, Create Table`.

Переваги застосування збережених процедур:

- підвищення швидкості виконання операцій;
- компактність виклику коду довільної складності;
- зменшення навантаження на мережу (код процедури зберігається на сервері) та розміру запитів;

- посилення безпеки;
- реалізація принципів модульного проектування.

Для створення збереженої процедури використовується інструкція, спрощений формат якої має вигляд:

```
CREATE PROC[EDURE] назва_процедури
[(@параметр1 тип_даних, …, @параметрN тип_даних)]
AS
BEGIN
   sql_оператор [...n]
END
GO;
```

У цій інструкції для процедури потрібно вказати унікальну та змістовну назву. Далі, в разі потреби, у круглих дужках через кому зазначають параметри, що передаватимуться до процедури зовні при її виклику. Після зарезервованого слова **AS** розміщується SQL-оператор. Якщо таких операторів кілька, їх необхідно об'єднати в блок, що обмежується ключовими словам **BEGIN** та **END**.

Запуск процедури:

```
EXEC[UTE] назва_процедури [значення_параметру/ів];
```

Приклад 10. Збережена процедура без параметрів. Одержати загальну кількість фільмів, наявних у базі даних «Мережа кінотеатрів» (рис. 5.1).

```
Create Procedure Count_Films
   As
      Select Count(film_id) From Films
   Go;
```

Запуск процедури: `Exec[ute] Count_Films;`

Приклад 11. Збережена процедура з параметром. Одержати кількість ролей, які зіграв актор із заданим номером.

```
Create Procedure Count_Roles @actor_num  int
   As
      Select Count(role_id) From Roles
      Where actor_id=@actor_num
   Go;
```

Запуск: `Exec Count_Roles 5`

Процедура у прикладі 11 має один параметр – actor_num цілого типу, який дає змогу при виклику зазначити номер актора, для якого потрібно порахувати кількість ролей. На відміну від звичайного запиту, цей номер не фіксований і може задаватися користувачем при зверненні до процедури, зокрема, у тексті програми, що взаємодіє з базою даних. При виклику процедури для параметра зазначається номер 5.

За аналогією з розглянутим прикладом, який вигляд матиме збережена процедура для визначення кількості акторів, які зіграли у заданому фільмі?

Збережені процедури також можуть застосовуватися для реалізації операцій оновлення, видалення та додавання нового запису.

Приклад 12. Додати дані про новий фільм із забезпеченням унікальності порядкового номера. Зауважте, що типи параметрів повинні узгоджуватися із типами відповідних полів таблиці.

```
Create Procedure AddFilm
@Ftitle nvarchar(40), @Fyear int, @Fgenre nvarchar(25)
    AS
-- оголошення локальної змінної, що відповідає номеру фільму
   Declare @Film_num int;
 -- значення змінної встановлюється на одиницю більшим
 --за максимальний порядковий номер у таблиці Films
   SET @Film_num=(Select Max(film_id) From Films);
   SET @Film_num=@Film_num+1;
-- перевірка наявності запису про фільм
   IF EXISTS (Select *
                 From Films
                 Where film_title=@Ftitle
                 And release_year=@Fyear)
        Print 'Фільм вже існує'
   ELSE
--додавання даних про новий фільм із визначеним номером
    Begin
     Insert Into Films
     Values (@Film_num, @Ftitle, @Fyear, @Fgenre)
     Print 'Фільм додано'
```

```
End
Go;
```

При виклику процедури передаються значення лише трьох параметрів, вказаних у її заголовку. Номер запису визначається автоматично.

```
EXEC AddFilm 'Код да Вінчі', 2000, 'Трилер'
```

Створіть збережену процедуру для додавання до таблиці Roles *даних про нового кіногероя. Запишіть оператори виклику цієї процедури аби записати до таблиці під унікальними номерами дані про ролі, які у фільмі «Код да Вінчі» зіграли Том Генкс і Одрі Тоту.*

Приклад 13. Виправити рік випуску фільму, доданого у попередньому прикладі.

```
Create Procedure UpdateFilm @id int , @y int=2005
  As
  Update Films
  Set release_year=@y
  Where film_id=@id
Go;
```

Виклик процедури:

```
Exec UpdateFilm 6, 2006
```

Тут для другого параметра встановлене значення за замовчуванням (2005). Для того, щоб оновлення торкнулося конкретного запису, застосовується умова за участі першого параметра *id*. При виклику процедури з кількома параметрами вони зазначаються після її назви у порядку, визначеному при створенні.

Тригери

Тригер – це спеціальний тип збережених процедур, який не потребує виклику, а спрацьовує автоматично при виконанні певної дії над таблицею. Це іменований механізм, який зберігається на сервері як окремий об'єкт бази даних.

Використання тригерів:

- підтримка цілісності даних;
- супровід або попередження дій над даними;
- автоматизація обробки даних;

- фудит дій користувачів;

Узагальнений формат оператора визначення тригера має вигляд:

```
CREATE TRIGGER назва_тригера
    ON назва_таблиці [WITH ENCRYPTION]
    {For |After |Instead Of}{[Insert] [,]|[Update] [,]
    [Delete]}
    AS
     BEGIN
       IF Подія_тригера1 [ { AND | OR } Подія_тригера2 [...n]
             BEGIN
                   Дія/sql_оператори [.. .n]
             END
     END
    GO
```

Призначення параметрів, що використовуються:

- ❖ **WITH ENCRYPTION** – вказівка серверу виконувати шифрування коду тригера для приховання авторських алгоритмів обробки даних.
- ❖ **FOR** або **AFTER** – тригер буде спрацьовувати лише після повного виконання всіх команд, які його викликали.
- ❖ **INSTEAD OF** – тригер буде виконуватися замість відповідних запитів користувачів, що спричинили його виклик.
- ❖ **DELETE | INSERT | UPDATE** визначають, на які команди (видалення, додавання або оновлення) реагуватиме тригер. Потребує зазначення принаймні однієї з команд. Кілька команд слід вказувати через кому.
- ❖ Для однієї таблиці можна створювати кілька тригерів.

При виконанні тригера необхідно перевіряти умови, які порівнюють значення полів таблиці до та після виконання дії. З цією метою для тестування наслідків інструкції, через яку спрацьовує тригер, використовуються дві віртуальні таблиці з визначеними назвами:

- ✓ **Deleted** – містить копії видалених або оновлених рядків таблиці.

- ✓ **Inserted** – містить рядки, що додаються до таблиці.

Структура цих таблиць еквівалентна структурі вихідної таблиці, для якої визначено тригер.

Розглянемо приклади створення тригерів.

Приклад 14. Заборонити видалення з таблиці фільмів, випущених після 2000 року.

```
Create Trigger FilmDel_2000
    On Films
    For Delete
    As
    Begin
      If Exists (Select* From Deleted
                  Where Deleted.release_year>2000)
           Begin Rollback;
                 Print 'Забороняється видалення фільмів
                 після 2000 року' return;
          End;
      End;
    Go;
```

Перш ніж видалити дані з Films, тригер перевірятиме, чи серед видалених рядків (віртуальна таблиця **Deleted**) немає даних про фільми, що вийшли після 2000 року. Для рядків, що повертатиме вкладений запит, дію з видалення буде відхилено завдяки оператору мови SQL **Rollback** зі скасування транзакції.

Приклад 15. Заборонити видалення даних про актора, якого зазначено як виконавця ролі у таблиці Roles.

```
Create Trigger Del_actors
   On Actors
   For Delete
    As
      Begin
      If Exists (Select*
                  From Roles R Inner Join Deleted D
                  On R.actor_id=D.actor_id)
           Begin    Rollback;
```

```
                Print 'Неможливо видалити дані залученого актора' return;
            End;
      End;
Go;
```

Для перевірки наявності ролей у актора вкладений запит, який визначає умову спрацювання тригера, виконує внутрішнє з'єднання дочірньої таблиці Roles із віртуальною таблицею Deleted, що містить видалені рядки батьківської таблиці Actors.

Приклад 16. Дозволити додавання до таблиці Actors даних про акторів лише з тих країн, що наразі містяться в таблиці. Цей приклад радше демонстраційний, аніж реальний.

```
Create Trigger Insert_actors
   On Actors
   For Insert
 As
    Begin
      If  Not Exists (Select * From Actors
      Where Actors.actor_country=Inserted.actor_country)
       Begin  Rollback;
        Print 'Не можна додати дані актора з введеної країни'
        return;
       End;
  End;
Go;
```

ЛАБОРАТОРНА РОБОТА № 6
Застосування процедур для виконання SQL-операцій

Мета

Уніфікувати виконання операцій на випадок створення програмного інтерфейсу та забезпечити додаткову підтримку цілісності бази даних у СУБД Microsoft SQL Server 20x.

Завдання

Набути навичок виконання довільних операцій обробки у базі даних та оформлення їх у вигляді збережених процедур та тригерів.

ПРАКТИЧНЕ ЗАВДАННЯ

Як зазначалося раніше, SQL-скрипт запитів можна зберігати як самостійний об'єкт бази даних з можливістю багаторазового виклику та передавання довільних значень параметрів. Подекуди операції з даними можуть передбачати виконання визначеної послідовності дій із додатковою перевіркою коректності їх застосування. З цією метою доцільно оформити такий набір операцій у вигляді окремих об'єктів бази даних, як-от збережені процедури та тригери. Розглянемо створення цих об'єктів на прикладі скороченого варіанта бази даних «Мережа кінотеатрів», схема якої зображена на рис. 5.1.

1. Створення збережених процедур

1.1. У середовищі **Microsoft SQL Server Management Studio** оберіть та розгорніть в **Object Explorer** робочу базу даних.

1.2. Розкрийте розділ **Programmability**, у якому міститься папка **Stored Procedures** (Збережені процедури). Натисніть на ній правою кнопкою миші та в контекстному меню оперіть опцію **New> Stored Procedure...** , як зображено на рис. 6.1.

1.3. У вікні, що з'явилося (рис. 6.2), можна побачити інструкцію щодо створення збереженої процедури у вигляді коментарів. У мові SQL коментар починається із двох послідовних символів '-' без пробілів, а завершується переходом на новий рядок. Серед іншого можна зазначити автора процедури, дату створення та її короткий опис, що сприяє

кращому орієнтуванню в авторському коді. До того ж у вікні наявний шаблон створення коду, який можна редагувати.

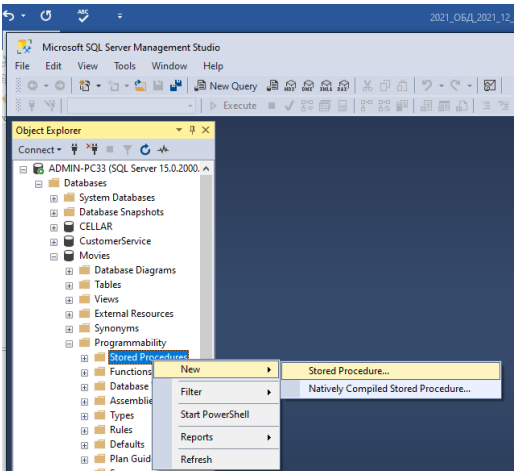


Рис. 6.1. Створення збереженої процедури у базі даних

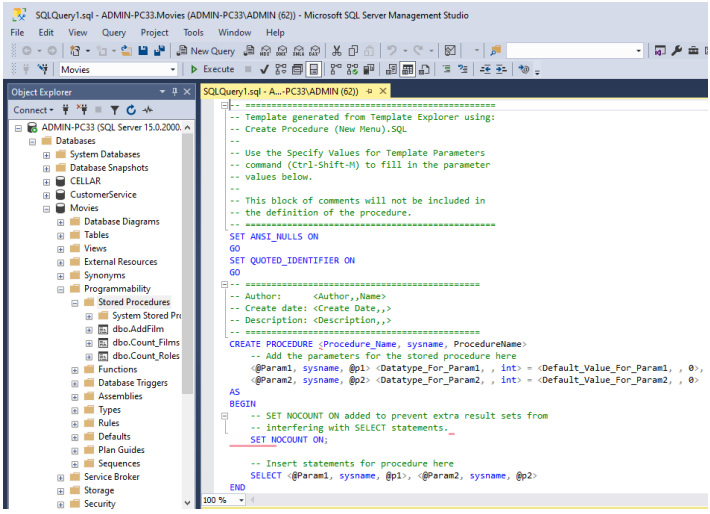


Рис. 6.2. Вікно для створення нової процедури за визначеним шаблоном

1.4. Припустимо, що описану в прикладі 12 процедуру AddFilm було реалізовано та застосовано для додавання до таблиці Films даних про новий фільм ‘Код да Вінчі’.

1.5. Розгляньмо створення збереженої процедури для додавання даних про нового кіногероя до таблиці Roles із наступним виведенням її повного вмісту. Один з можливих варіантів SQL-скрипта наведений на рис. 6.3.

```
-- Author:      Yuliya Tanasyuk
-- Create date: Dec 11, 2021
-- Description: Додавання даних про нового кіногероя
-- =============================================
CREATE PROCEDURE AddNewRole
    -- Add the parameters for the stored procedure here
    @Rname nvarchar(40), @Actor_num int, @Film_num int
AS
BEGIN
    Declare @Rnum int;
    Set @Rnum=(Select Max(role_id) from Roles)+1;

        -- Додавання нового запису
        Insert into Roles
        Values(@Rnum, @Rname, @Actor_num, @Film_num)
        Print 'Дані про нову роль успішно додано'
        -- Вибірка повного вмісту таблиці
    SELECT * From Roles;
END
GO
```

Рис. 6.3. SQL-код збереженої процедури

1.6. Перш ніж зберігати процедуру, необхідно переконатися, що її код не містить помилок. Для цього натисніть кнопку **Execute** на панелі інструментів або клавішу **F5**. Як показано на рис. 6.4, результат компіляції, що відображається у нижньому вікні **Messages**, має сповіщати про успішне виконання введених команд.

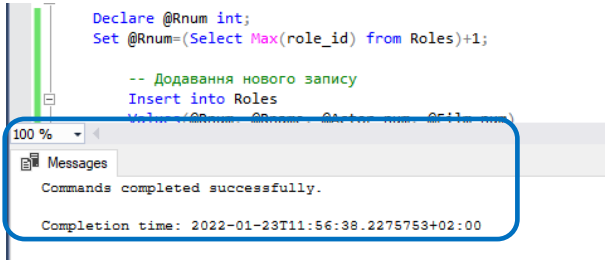


Рис. 6.4. Сповіщення про успішну компіляцію SQL-коду створеної збереженої процедури

1.7. Зберегти код процедури як об'єкт бази даних з унікальним іменем можна, натиснувши на піктограму із зображенням дискети на панелі інструментів або при закритті вікна з кодом.

1.8. Щоб побачити створену процедуру серед об'єктів бази даних (рис. 6.5), необхідно натиснути правою кнопкою миші на загальній папці **Stored Procedure** та обрати в контекстному меню опцію **Refresh** (Оновити).

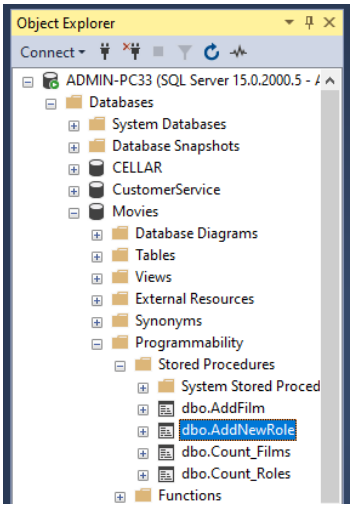


Рис. 6.5. Відображення нової збереженої процедури в переліку об'єктів поточної бази даних Movies

1.9. Перш ніж запустити щойно створену процедуру на виконання, переглянемо вміст таблиць у базі даних (рис. 6.6). У таблиці Films (рис. 6.6, *б*) можна побачити запис під номером 6 про фільм 'Код да Вінчі'. Тепер потрібно додати до таблиці Roles (рис. 6.6, *в*) дані про героїв цього фільму: Роберта Ленгдона та Софі Неве, яких втілили на екрані актори Том Генкс та Одрі Тоту з номерами 5 і 6 відповідно (рис. 6.6, *а*).

1.10. Перейдемо до таблиці Roles, натиснувши правою кнопкою миші на таблиці в **Object Explorer** і обравши, наприклад, пункт меню **Select Top 1000 Rows** (Вибрати перші 1000 рядків).

-PC33.Movies - dbo.Actors | ADMIN-PC33.Movies - dbo.Roles

actor_id	actor_name	actor_bday	actor_country
1	Джонні Депп	1963-06-09 00:0...	США
2	Омар Сі	1978-01-20 00:0...	Франція
3	Кіра Найтлі	1985-03-26 00:0...	Велика Британія
4	Бенедикт Кемб...	1956-07-09 00:0...	Велика Британія
5	Том Генкс	1956-07-09 00:0...	США
6	Одрі Тоту	1976-08-09 00:0...	Франція

а

-PC33.Movies - dbo.Films | ADMIN-PC33.Movies - dbo.Actors

film_id	film_title	release_year	genre
1	Пірати Карибськ...	2003	Фентезі
2	Гра в імітацію	2014	Трилер
3	Втеча з Шоушенка	1994	Драма
4	Зелена миля	1999	Драма
5	Недоторканні	2011	Комедія
6	Код да Вінчі	2006	Трилер

б

-PC33.Movies - dbo.Roles | ADMIN-PC33.Movies - dbo.Film

role_id	hero_name	actor_id	film_id
1	Джек Горобець	1	1
2	Дріс	2	5
3	Алан Тюрінг	4	2
4	Елізабет Свон	3	1
5	Джоан Кларк	3	2
6	Пол Еджкомб	5	4

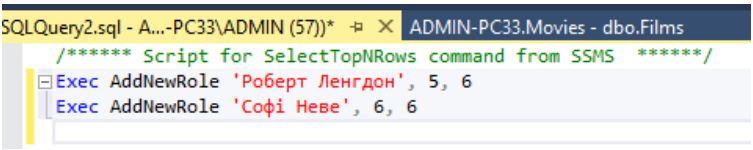
в

Рис. 6.6. Вміст таблиць бази даних Movies:
а – Actors, *б* – Films, *в* – Roles

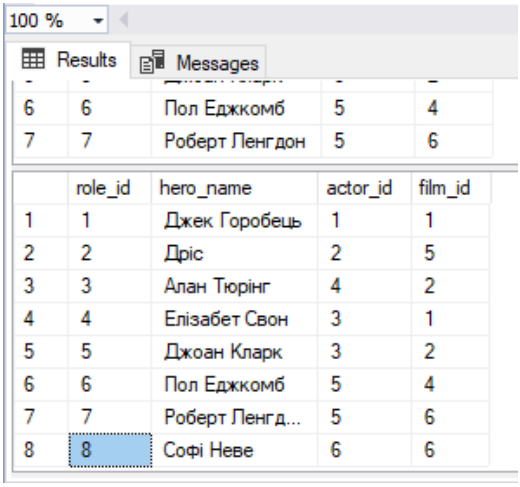
1.11. У вікні, що відкрилося, за замовчуванням міститься шаблон команди `Select` з виведення перших 1000 рядків таблиці Ролі. Цю команду можна видалити або здійснити виклик процедури нижче.

1.12. Щоб додати два записи про кожного з героїв названого фільму, створену процедуру потрібно буде викликати двічі. Як параметри будемо вказувати назву кіногероя, номер актора і номер фільму. Порядковий номер ролі визначатиметься

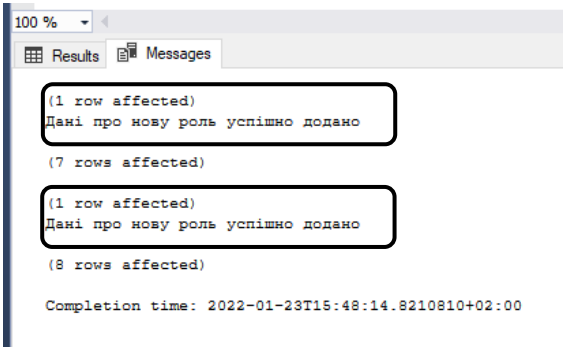
автоматично, згідно з кодом процедури. Результат виклику процедури зображений на рис. 6.7.



а



б



в

Рис. 6.7. Застосування збереженої процедури: *а* – виклики процедури, *б* – оновлені дані в таблиці Roles, *в* – повідомлення про результати виконання процедури

1.13. Як можна побачити на рис. 6.7, *а*, подано команди виклику на виконання процедури для двох нових записів. Оскільки наша процедура, крім оператора `Insert`, виконує вибірку всіх даних, бачимо два результати послідовного оновлення таблиці Roles (рис. 6.7, *б*). У вкладці **Messages** відображаються повідомлення про результати виконання операцій, зокрема, передбачені в тілі процедури (рис. 6.7, *в*).

1.14. У збереженій процедурі можна було б також реалізувати перевірку id актора та фільму, які зазначаються при додаванні нового запису про роль. Проте, завдяки тому, що це поля зовнішнього ключа, їх відповідність значенням первинних ключів, на які вони посилаються, виконується автоматично.

2. Створення тригерів

2.1. В **Object Explorer** для поточної бази даних розгорніть розділ **Programmability** та клацніть правою кнопкою миші на папці **Database Triggers**. У контекстному меню оберіть пункт **New Database Trigger** (рис. 6.8).

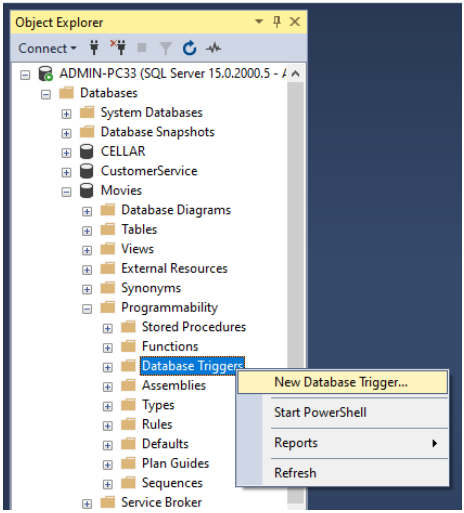


Рис. 6.8. Створення нового тригера у базі даних

Інший варіант передбачає створення тригера безпосередньо для тієї таблиці, якої він стосується.

2.2. Створимо тригер з прикладу 15, який заборонятиме видалення даних про акторів, якщо ролі, які вони зіграли, містяться у таблиці Roles. Тригер спрацьовуватиме для таблиці Actors, тому розгорнемо її і серед доступних елементів обиремо тригери, як показано на рис. 6.9.

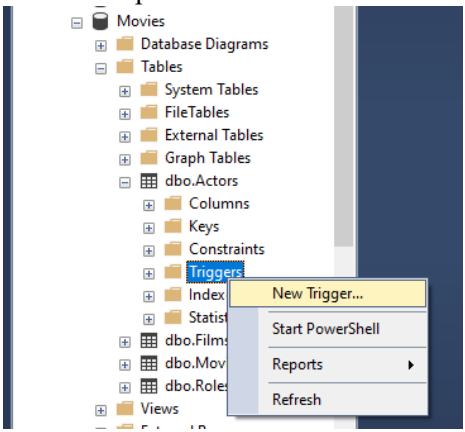


Рис. 6.9. Створення нового тригера для таблиці Actors

2.3. У вікні, що відкрилося, за замочуванням міститься шаблон для створення коду, який можна змінювати. Додамо скрипт тригера з прикладу 15, як показано на рис. 6.10.

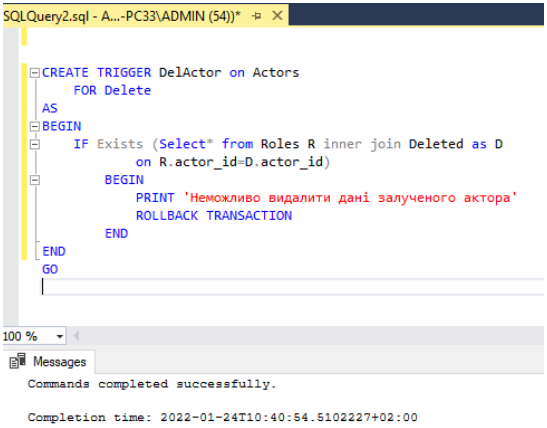


Рис. 6.10. SQL-скрипт тригера, створеного для Actors

2.4. Для перевірки правильності написання коду його потрібно відкомпілювати, натиснувши кнопку **Execute** на панелі інструментів або клавішу **F5**. Результат перевірки відображається у вікні **Messages** (рис. 6.10).

2.5. Зберегти створений SQL-код, як окремий об'єкт бази даних, можна натисканням на піктограмі з зображенням дискети на панелі інструментів, за допомогою комбінації клавіш **Ctrl+S** або при закритті вікна з кодом. Після цього його можна побачити у папці **Triggers** відповідної таблиці.

2.6. Оскільки тригер – це збережена процедура, що спрацьовує автоматично при настанні події, перевірити його на практиці можна, виконавши операцію видалення в таблиці Actors.

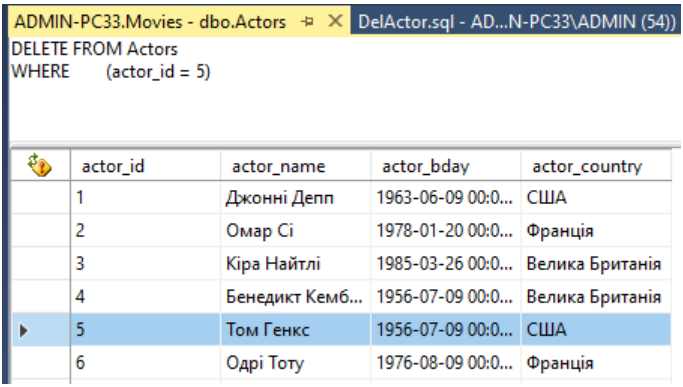


Рис. 6.11. Видалення даних з таблиці Actors для перевірки роботи тригера

2.7. Відкриємо таблицю Actors (**Select Top 1000 Rows** або **Edit Top 200 Rows**) та у вікні для введення SQL-коду створимо команду для видалення даних про актора з номером 5, як показано на рис. 6.11. Нагадаємо, що в нашій базі даних міститься інформація про дві ролі, які виконав Том Генкс. Тому, за логікою тригера, дані про цього актора видаляти не можна.

2.8. Після запуску запиту з'являється повідомлення про конфлікт, який виник при виконанні операції Delete через

наявність у таблиці Roles зовнішнього ключа actor_id (рис. 6.12), тобто так, як ми і запрограмували. У результаті дані залишаються у таблиці.

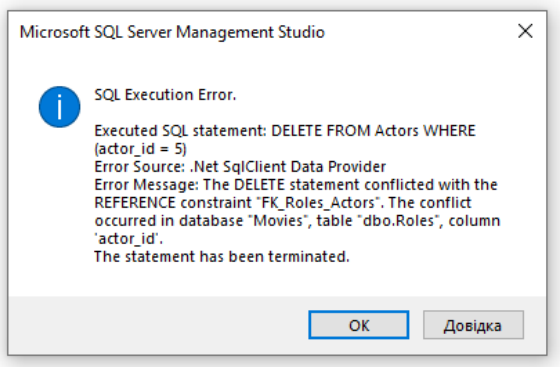


Рис. 6.12. Повідомлення про неможливість виконання операції видалення через спрацьовування тригера

2.9. Якщо до таблиці Actors додати запис про нового актора, для якого немає інформації про жодного кіногероя у таблиці Ролі, його видалення буде виконано успішно.

2.10. Слід зауважити, що інформаційне повідомлення, яке міститься в тілі тригера, безпосередньо не відображається у вікні поряд із таблицею, як у випадку збережених процедур, а заноситься до log-файлу бази даних.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. *Ознайомтесь із запитами за варіантом індивідуального завдання, які необхідно реалізувати.*

2. *Відповідно до завдання, оформіть запити у вигляді об'єктів бази даних: представлень (View), збережених процедур або тригерів.*

3. *Перевірте коректність виконання запитів для різних наборів даних.*

4. *SQL-скрипт операцій, команди їх виклику (для процедур) та результат виконання збережуть для оформлення звіту.*

Звіт з лабораторної роботи повинен містити:

1. Формулювання запиту за завданням.
2. Скрипт кожного SQL-запиту.
3. Екранні форми результату виконання запиту.
4. Відповіді на запитання для самоперевірки.

Для захисту лабораторної роботи необхідно продемонструвати одержані результати у СУБД, обґрунтувати обраний спосіб реалізації запитів, оформити звіт та бути готовими відповісти на запитання з переліку нижче.

Запитання та завдання для самоперевірки

1. Для чого використовуються вкладені запити?
2. Яка властивість операцій реляційної моделі уможливила застосування вкладених запитів?
3. У який спосіб оператор вибірки може поєднувати в собі кілька запитів?
4. Порівняйте використання вкладених запитів та операцій з'єднання. У яких випадках доцільне використання кожного зі способів?
5. Як результат вкладеного запиту впливає на утворення загальної умови вибірки? Які оператори при цьому можуть застосовуватися?
6. Поясніть особливість оператора `Exists/Not Exists`.
7. Що таке збережена процедура мови SQL? Як вона створюється та використовується?
8. Яким чином за допомогою запитів, оформлених у вигляді процедур, можна передавати значення параметрів?
9. Поясніть доцільність використання тригерів у базі даних. Що вони забезпечують та чим відрізняються від збереженої процедури?
10. Як визначити дію, що викликає спрацьовування тригера? Яке призначення таблиць `deleted` та `inserted`?

ВАРІАНТИ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ

Варіант 1
«Книжковий магазин»

Ви проектуєте інформаційну систему для магазину з продажу друкованої продукції (книжки, щоденники, календарі, настільні ігри), яка розподілена по окремих відділах магазину (підручники, художня література, детективи, дитяча книга, канцтовари тощо). Продукція, потрапляє до магазину від окремих фірм-постачальників, з якими на визначений термін укладається договір з унікальним номером. З одним постачальником може бути укладено кілька договорів, а в одному договорі можуть бути різні найменування товарів. Потрібно виконувати облік проданої продукції. Продавці магазину працюють по днях тижня у визначених відділах і беруть участь у оформленні продажу друкованої продукції.

Інформаційна система, що проектується, повинна вести облік: продукції, співробітників, постачальників, укладених договорів, проданої продукції, графіка роботи працівників тощо.

Варіант 1.1

1. Отримати повну інформацію про постачальника продукції за номером договору (номер договору – параметр).
2. Отримати інформацію про постачальників, які не доправили жодної гри.
3. Підрахувати кількість співробітників, які працюють у заданий день тижня у визначеному відділі.
4. Додати інформацію про новий товар.
5. Видалити інформацію про договори на постачання календарів.

Варіант 1.2

1. Отримати інформацію про те, в які дні тижня працює заданий співробітник.
2. Вивести інформацію про продавців, які за один день продали товар на суму, більшу за 400 грн.

3. Підрахувати вартість продукції, проданої за останній тиждень (від поточної дати).
4. Збільшити вартість щоденників на 10 відсотків.
5. Додати до розкладу дані робітника, що працюватиме у відділі художньої літератури.

Варіант 1.3

1. Знайти продукцію, яку продали у вказану дату.
2. Отримати інформацію про те, хто працював в утому ж відділі, що і деякий пробавець.
3. Підрахувати вартість продукції, яку надав визначений постачальник у вказаний термін.
4. Додати інформацію про створення нового відділу з унікальною назвою.
5. Видалити інформацію про заданого постачальника, якщо з ним не укладено договорів.

Варіант 2
«Курси іноземних мов»

Ви проектуєте інформаційну систему для курсів іноземних мов. Ці курси надають бажаючим послуги з вивчення різних іноземних мов. Процес навчання розбито на кілька етапів (рівнів), на які здійснюється набір слухачів у групи. Для подання заявки на курси потрібно заповнити анкету і надати свою контактну інформацію (телефон, email, місто проживання). Вартість навчання на кожному рівні для різних мов може відрізнятися. Навчання в групі за певним рівнем проводить окремий викладач. Слід врахувати, що один викладач на курсах може викладати у кількох групах (необов'язково з однієї мови). По закінченні кожного етапу слухачі складають фінальний іспит, який слід скласти не менше ніж на 70 балів для отримання відзнаки про рівень володіння мовою.

Інформаційна система, що проектується, повинна вести облік: доступних для вивчення іноземних мов, викладачів, рівнів мови, слухачів, оплати за навчання, розкладу занять у групах тощо.

Варіант 2.1

1. Отримати розклад занять конкретної групи.
2. Знайти всіх викладачів, які викладали англійську мову.
3. Одержати інформацію про слухачів, які успішно вивчили всі доступні рівні деякої іноземної мови.
4. Додати в групу з німецької мови рівня B1 нового слухача.
5. Визначити повну вартість вивчення однієї мови на всіх рівнях.

Варіант 2.2

1. Отримати інформацію про групи, в яких заняття проводить певний викладач.
2. Знайти всіх слухачів нового набору на іспанську мову рівня A2.
3. Знайти слухачів, які вивчають більше однієї мови.
4. Якщо студент вже прослухав на курсах більше трьох рівнів, суму його оплати за навчання зменшити на 10 відсотків.
5. Видалити інформацію про групу, до якої набрали менше 5 слухачів.

Варіант 2.3

1. Знайти інформацію про всі груп, які вивчали задану мову.
2. Знайти всіх слухачів, що не сплатили за навчання.
3. Вивести інформацію про учасників групи, до якої записано визначеного слухача (повне ім’я слухача – параметр запиту).
4. Замінити одного викладача у групі на іншого, якщо у цей час викладач вільний від занять.
5. Зменшити плату за навчання для всіх слухачів груп, де кількість студентів рівна або більше 20.

Варіант 3
«Приймальна комісія»

Ви проектуєте інформаційну систему для приймальної комісії університету. Для подання заявки на вступ абітурієнти спершу заповнюють анкету з інформацію про себе, яка зберігається у базі під унікальним номером. Після реєстрації,

абітурієнт може подавати заявки на кілька спеціальностей. Зауважте, що спеціальності мають свій окремий номер і в межах університету забезпечуються на визначених кафедрах і факультетах. Вступ на кожну спеціальність передбачає перелік предметів, з яких абітурієнти подають до системи реєстрації результати ЗНО. Терміни подання документів чітко визначені. Для кожної спеціальності вказується прохідний середній бал.

Інформаційна система, що проектується, повинна вести облік: абітурієнтів, спеціальностей, дисциплін, поданих заявок тощо.

Варіант 3.1

1. Знайти кількість абітурієнтів, що вступають на заданий факультет у поточному році.
2. Знайти всі спеціальності, для вступу на які необхідно подати результати з математики.
3. Для заданої спеціальності підрахувати середню оцінку з визначеного предмета.
4. Додати запис про нового абітурієнта, якщо він попередньо не реєструвався в системі.
5. Видалити інформацію про предмет, якщо він не враховується при вступі на жодну спеціальність.

Варіант 3.2

1. Знайти дисципліни, результати яких враховуються при вступі на задану спеціальність.
2. Знайти всіх абітурієнтів, результат з математики в яких вищий, ніж у заданого абітурієнта.
3. Знайти всі назви спеціальностей, для вступу на які необхідно подати результати більше трьох предметів.
4. Вивести повну інформацію про подання заяв на кожну спеціальність.
5. Видалити дані про абітурієнта, якщо він ще не подав жодної заяви на спеціальність.

Варіант 3.3

1. Знайти повну інформацію про спеціальності, які забезпечує визначений факультет.

2. Визначити кафедри, які виконують підготовку фахівців з двох та більше спеціальностей.
3. Знайти абітурієнтів, середній бал яких вищий за прохідний на обрану спеціальність.
4. Перевести абітурієнта на іншу спеціальність, якщо перелік предметів для них збігається.
5. Зареєструвати заяву на вступ від нового абітурієнта, якщо термін подання ще не сплинув.

Варіант 4
«Комп'ютерна фірма»

Ви розробляєте інформаційну систему для підтримки роботи фірми, яка займається ремонтом та обслуговуванням комп'ютерної техніки. Компанія має чіткий прейскурант послуг, що надаються, та фіксований штат працівників. Необхідно вести реєстр замовлених послуг (тип послуги, відповідальна особа, термін виконання), а також надавати гарантійні зобов'язання клієнтам. Інформаційна система дозволяє здійснювати облік комп'ютерних комплектуючих різного типу, що надходять від різних постачальників, згідно з укладеними договорами. Окремо слід зберігати інформацію про продаж продукції. У компанії діє програма лояльності для зареєстрованих клієнтів.

Інформаційна система, що проектується, повинна вести облік: співробітників, постачальників, клієнтів, товарів, послуг тощо.

Варіант 4.1

1. Знайти всі ремонтні роботи, які виконуються за гарантією.
2. Вивести повну інформацію про клієнтів, які придбали комплектуючі фірми Samsung.
3. Визначити максимальну суму виторгу за визначений місяць.
4. Запропонувати клієнтові перелік необхідних комплектуючих, вартість яких не перевищує заданої суми.
5. Вивести повну інформацію про постачальників продукції фірми ASUS.

Варіант 4.2

1. Знайти комплектуючі, які постачаються двома і більше постачальниками.
2. Отримати інформацію про те, скільки замовлень на переустановлення ОС було одержано за визначений період.
3. Вивести інформацію про те, скільки послуг було надано кожним співробітником.
4. Вивести інформацію про постачальників, а також продукцію, яку вони постачають.
5. Видалити інформацію про товари, які було відремонтовано за гарантією.

Варіант 4.3

1. Знайти повну інформацію про договори, укладені для постачання принтерів протягом визначеного періоду.
2. Отримати інформацію про те, скільки разів замовляли кожну послугу.
3. Вивести інформацію про клієнтів, які більше двох разів зверталися за визначеною послугою.
4. Зменшити на 10% вартість очистки ноутбука для кожного третього звернення від клієнта.
5. Видалити дані договору, за яким не було здійснено постачання продукції.

Варіант 5
«Розробка програмних продуктів»

В IT-компанії фіксований штат співробітників. Кожен працівник обіймає визначену посаду (розробник, дизайнер, аналітик QA, DevOps тощо) та має рівень майстерності (trainee, junior, middle, senior, expert та ін.). Компанія співпрацює з рядом фірм і виконує на їх замовлення проекти з фіксованим бюджетом та визначеною тривалістю. Над кожним проектом працює команда з 5- 7 співробітників. Проект розбито на низку завдань, які можна розподіляти між учасниками команди. Завдання послідовно проходять такі стадії виконання: to do (потрібно зробити), in process (в процесі), in test (на перевірці), done (зроблено). Над одним завданням на різних стадіях можуть

працювати кілька учасників команди. Необхідно фіксувати дату переходу з однієї стадії завдання до іншої. Кожен завершений проект потрібно позначати датою схвалення замовником.

Інформаційна система, що проектується, повинна зберігати дані про: співробітників, їх посади та рівні, замовників, проекти, команди, фіксувати обсяг робіт за проектом, відстежувати розподіл завдань та стан їх виконання тощо.

Варіант 5.1

1. Знайти всіх співробітників, які працюють над визначеним проектом.
2. Отримати інформацію про фірми, які замовляли більше трьох проектів (назва фірми – параметр запиту).
3. Знайти проекти, у яких не розпочато більше трьох завдань.
4. Перевірити, чи деякий користувач бере участь у виконанні кількох завдань з різних проектів.
5. Видалити інформацію про співробітника, якщо в нього немає незавершених завдань.

Варіант 5.2

1. Знайти дані усіх співробітників, що наразі працюють над завданнями на стадії тестування (in test).
2. Отримати інформацію про DevOps-ів, які беруть участь у заданому проекті.
3. Знайти повну інформацію про завдання деякого проекту, їхні стадії та виконавців.
4. Зареєструвати завершення виконання деякого завдання, якщо дата завершення не перевищує дату подання проекту загалом.
5. Видалити інформацію про завдання, якщо його не розпочато та не обрано деяким співробітником.

Варіант 5.3

1. Отримати повну інформацію про проекти та завдання, що перебувають у розробці (in process) із листопада поточного року.
2. Знайти імена та рівень майстерності всіх учасників команди, які наразі не завершили виконання завдань.

3. Відобразити для визначеного проекту кількість завдань, що перебувають на кожній стадії.
4. Визначити проекти, на яких працює більше 5 учасників.
5. Змінити статус виконання завдання, якщо для нього заповнено дату завершення.

Варіант 6
«Готель»

Ви розробляєте інформаційну систему для готелю, яка забезпечує виконання завдань обліку поселення і завантаження приміщень. У готелі наявна певна кількість номерів визначених типів: одномісний, стандарт, студія, люкс, за якими закріплена вартість проживання за ніч. Для кожного номера варто зберігати інформацію про кількість спальних місць та кімнат, а також поверх, на якому вони розташовуються. Для проживання у готелі клієнт повинен попередньо зарезервувати потрібний тип номера на визначений період, вказавши свої контактні дані. Поселення та оплати за проживання фіксується окремо працівником готелю.

Інформаційна система, що проектується, повинна вести облік: клієнтів, персоналу, номерів, проживання тощо.

Варіант 6.1

1. Отримати інформацію про тих, хто зупинився наразі у певному номері готелю.
2. Знайти номери, що розташовуються на тому ж поверсі, що й вказаний номер.
3. Перевірити чи є клієнти, що проживали в готелі більше одного разу в одному й тому ж номері.
4. Збільшити на 15 % вартість номерів визначеного типу.
5. Видалити інформацію про клієнтів, якщо вони не поселилися в готелі та не сплатили за проживання.

Варіант 6.2

1. Отримати інформацію про те, скільки разів замовляли кожен номер.
2. Вивести інформацію про номери, заброньовані на визначену дату.
3. Знайти номери, які наразі незаброньовані.

4. Забронювати будь-який одномісний номер, вільний протягом вказаного періоду.
5. Видалити інформацію про бронювання номера, якщо на заявлену дату за нього ще не сплачено.

Варіант 6.3

1. Отримати інформацію про визначеного клієнта: коли, в яких номерах і як довго він проживав.
2. Знайти вільні номери, вартість проживання в яких за один день менша за вказану суму.
3. Підрахувати вартість проживання клієнта у номері типу «люкс» за тривалістю його перебування.
4. Перевести клієнта до іншого вільного номера.
5. Видалити дані про номер, якщо в ньому ніхто не проживає і він не був заброньований.

Варіант 7
«Служба доставки»

Мережу служби доставки створено, аби легко та швидко доправляти посилки користувачів до відділень у межах країни. Вартість доставки вантажу залежить від його ваги та термінів доправлення (терміново або стандартно). Тарифи та категорії вантажів можете визначити за власним бажанням. Працівники служби працюють у відділеннях приймання/відправлення вантажів по днях тижня у визначені зміни. Клієнтів, що користуються послугами, ідентифікують за номером телефону. Отримання та надсилання посилок здійснюється за участі працівника та клієнта у відділенні служби доставки із зазначенням дати і часу, контактних даних адресатів та цільового відділення. Слід твкож вести облік оплати послуг доставки.

Інформаційна система, що проектується, повинна вести облік: відділень, працівників, тарифів, клієнтів, оформлення та отримання вантажів тощо.

Варіант 7.1

1. Знайти повну інформацію про всі відділення служби у місті Ужгород.
2. Визначити, скільки відправлень було опрацьовано в кожному відділенні на визначену дату.

3. Вивести інформацію про клієнтів, які за весь час роботи служби скористалися її послугами більше трьох разів.
4. Додати запис про нову посилку, якщо дата її доставки перевищує поточну дату.
5. Вивести повний розклад роботи працівників визначеного відділення.

Варіант 7.2

1. Знайти за номером телефону дані про всі оформлені відправлення клієнта.
2. Перевірити наявність термінових посилок, надходження яких очікує конкретне відділення вказаного міста.
3. Одержати загальну вартість отриманих посилок у відділенні на вказану дату.
4. Знайти повну інформацію про співробітників, які не оформили жодного цінного вантажу.
5. Видалити інформацію про клієнта, якщо в нього немає жодного несплаченого відправлення.

Варіант 7.3

1. Вивести повну інформацію про працівників відділення та дані відправлень, які вони оформили.
2. Знайти адреси відділень, у яких працює менше трьох співробітників.
3. Визначити повну інформацію про клієнта, який оформив найбільшу кількість відправлень.
4. Зменшити на 10% вартість термінових посилок, які чекають на відправлення у визначеному місті.
5. Визначити дані співробітників, які не працювали по суботах.

Варіант 8
«Туристична фірма»

Ви розробляєте інформаційну систему для туристичної фірми, яка здійснює перевезення за визначеними маршрутами. Кожен маршрут має свою визначену назву, тривалість, вартість та може охоплювати декілька міст у різних країнах. Сформовано графік виїзду туристичних груп за визначеним маршрутом. Для

бронювання туру кожен клієнт повинен попередньо зареєструватися, після чого матиме змогу долучитися до групи. Туристична фірма здійснює перевезення автобусами з різною кількістю місць. Кожну подорож за маршрутом обслуговують працівники фірми: керівник групи, гід та два водії. Оплата від клієнтів отримується у гривнях.

Інформаційна система, що проектується, повинна вести облік: співробітників, клієнтів, маршрутів, перевезень, бронювання, оплати тощо.

Варіант 8.1

1. Знайти тури до заданої країни на визначені дати.
2. Одержати інформацію про водіїв, шо обслуговують деякий маршрут у поточному місяці.
3. Підрахувати кількість людей у кожній групі, які формуються для поїздок до Іспанії.
4. Перерахувати суми, сплачені клієнтами за конкретну путівку, в євро, згідно з курсом валюти.
5. Видалити дані про клієнта, якщо він ще не сплатив за поїздку, або вже повернувся з оплаченої подорожі.

Варіант 8.2

1. Знайти всі маршрути, що охоплюють більше двох країн.
2. Знайти клієнтів, які замовили путівки до Туреччини на вересень.
3. Знайти автобус, яким у зазначений день можна дістатися до Венеції.
4. Додати нового клієнта до групи за наявності вільних місць.
5. Видалити інформацію про виїзд групи в тур, якщо на нього немає замовлень.

Варіант 8.3

1. Знайти дані про співробітників турфірми, що обслуговували вказаний маршрут.
2. Одержати повну інформацію про чисельність груп, які в зазначений день перебувають у дорозі.

3. Знайти відомості про тури, що охоплює ті ж країни, що й маршрут під назвою «Фантастичний вікенд».
4. Додати дані про оплату поїздки, якщо вона здійснена до початку туру.
5. Знайти тури, на які не було попиту.

Варіант 9
«Фітнес-центр»

Ви розробляєте інформаційну систему для фітнес-центру. Заняття можуть проходити у різних приміщеннях, як-от тренажерний зал, йога-студія, басейн, корти тощо. Колектив складається з професійних тренерів, які проводять заняття у групах за різними спортивними напрямками, а саме: аеробіка, йога, бокс, плавання тощо. Максимальна чисельність кожної групи визначається місцем проведення та видом спорту. Заняття у групах проводять по днях тижня згідно зі сформованим графіком. Аби скористатися послугами клубу, клієнт повинен попередньо зареєструватися та придбати абонемент за обраним спортивним напрямком на визначений термін. Відеться облік відвідування занять за придбаним абоніментом.

Інформаційна система, що проектується, повинна зберігати інформацію про: клієнтів, тренерів, види спорту, приміщення, придбання абонементів, розклад, відвідування тощо.

Варіант 9.1

1. Знайти клієнтів, які займаються певним видом спорту.
2. Знайти тренерів, які проводять заняття у всіх групах з плавання.
3. Підрахувати кількість людей у кожній групі, що займається аеробікою у групах з 10 осіб та більше.
4. Додати запис про нового клієнта у групи з певного спортивного напрямку.
5. Одержати розклад проведення занять у визначеному залі в конкретний день тижня.

Варіант 9.2

1. Знайти всі види занять, що проходять у вказаному спортзалі.

2. Знайти клієнтів, які придбали два та більше абонементів з певного виду спорту.
3. Знайти тренера, який проводить заняття у заданий день тижня у тренажерному залі о певній годині.
4. Додати дані про відвідування клієнтом заняття у своїй групі на поточну дату, якщо в нього дійсний абонемент.
5. Видалити інформацію про групу, якщо до неї записалося не більше трьох клієнтів.

Варіант 9.3

1. Знайти клієнта, який займається різними видами спорту.
2. Знайти повну інформацію про спортивні зали, у яких проводять заняття більше трьох тренерів.
3. Знайти кількість спортсменів у кожній групі з боксу.
4. Одержати повну інформацію про відвідування занять з йоги.
5. Додати дані про придбання абонемента, якщо сума оплати відповідає встановленій вартості.

Варіант 10
«Електронний журнал студентів»

Ви розробляєте інформаційну систему для обліку відвідувань занять студентами певної академічної групи. В обов'язки старости входить зберігання даних розкладу занять по парах та днях тижня. У розкладі варто зазначати тип заняття: лекція, лабораторна робота або практичне заняття, аудиторію та викладача. Окремо необхідно зберігати список студентів групи. Для кожного заняття потрібно проводити облік відвідування занять за розкладом на конкретну дату із зазначенням причини відсутності.

Інформаційна система, що проектується, повинна вести облік: студентів, викладачів, дисциплін, розкладу занять, пропусків тощо.

Варіант 10.1

1. Знайти студентів, які не пропустили жодного заняття від початку семестру.

2. Знайти викладачів, що проводять заняття в рамках однієї дисципліни.
3. Додати запис про нового студента групи.
4. Виявити студентів, що пропустили 10 годин лекцій з поважних причин.
5. Видалити дані про студента, якщо кількість пропущених ним занять перевищує 70 годин.

Варіант 10.2

1. Знайти студентів, які пропустили більше 20 годин занять протягом місяця.
2. Одержати розклад проведення занять на тиждень з певної дисципліни.
3. Знайти студентів, які пропустили більше 15% від загальної кількості лабораторних з заданого предмета.
4. Сформувати звіт із кількістю пропусків для кожного студента на поточну дату.
5. Знайти інформацію про заняття, що проходять у тій самій аудиторії, що й лекції з вищої математики.

Варіант 10.3

1. Знайти предмет, заняття з якого не пропускав з неповажних причин жоден студент.
2. Знайти кількість пропущених практичних занять з певного предмета студентами у жовтні.
3. Створити звіт із зазначенням кількості пар на тиждень з кожної дисципліни.
4. Визначити студента, що пропустив максимальну кількість занять з неповажних причин.
5. Скласти рейтинг предметів за рівнем їх відвідування студентами.

Варіант 11
«Ресторан»

Розробіть інформаційну систему, яка б автоматизувала роботу ресторану. Зокрема, у системі слід проводити облік працівників закладу, страв, запропонованих у меню, індивідуальних замовлень клієнтів. Для кожної страви

зазначається назва, вага та вартість. Страви розбито на декілька категорій, як-от закуски, салати, гарячі страви, гарніри, десерти, напої тощо. Зауважте, що офіціанти ресторану працюють за встановленим розкладом і обслуговують визначені столики. Замовлення та його оплата закріплюються за певним столиком.

Інформаційна система, що проектується, повинна вести облік: працівників, страв, замовлень, розкладу роботи та ін.

Варіант 11.1

1. Знайти прізвища офіціантів, які обслуговують столик з номерами від 3 до 5.
2. Визначити кількість страв, які були видані на кожен столик.
3. Обчислити загальну суму виконаних замовлень на задану дату.
4. Вивести повну інформацію про замовлення десертів.
5. Встановити знижку 10 % на оплату всіх замовлень, зроблених на день незалежності.

Варіант 11.2

1. Знайти прізвища офіціантів, які працюють у середу та п’ятницю.
2. Перевірити, чи наявні страви, які не замовляли жодного разу.
3. Вивести прізвища офіціантів та повну інформацію про столики та страви, які вони подавали у визначений день.
4. Визначити максимальну суму замовлення для деякого столика.
5. Перевірити, чи збігається оплата за замовлення із сумарною вартістю замовлених за обраним столиком страв.

Варіант 11.3

1. Вивести інформацію про всі гарячі страви, вартість яких не перевищує 145 грн.
2. Вивести страви, які перебувають у тій самій категорії, що й банош.
3. Визначити столик, за яким у визначений день не замовляли жодного десерту.

4. Скласти рейтинг найбільш популярних страв закладу.
5. Зменшити суми оплати на 15 % при замовленні страв на 500 грн і більше для одного столика.

Варіант 12
«Гірськолижний курорт»

Розробіть інформаційну систему для автоматизації та централізованого обліку інформації гірськолижного комплексу. Передбачте можливість збереження інформації про персонал комплексу, ведення обліку гірськолижного спорядження, та його видачі клієнтам за наявності документів, що засвідчують особу. При видачі спорядження встановлюється кінцевий термін користування. Катання дозволене за умови придбання абонемента з визначеним тарифом. На курорті працює декілька підйомників, що характеризуються складністю траси (зелена, синя, червона, чорна), протяжністю, кількістю місць. При заході на підйомник автоматично зчитується номер абонемента та час. Завдяки цьому можна вести облік кількості спусків.

Інформаційна система, що проектується, повинна вести облік: клієнтів, працівників, підйомників, спорядження, продаж абонементів та ін.

Варіант 12.1

1. Знайти повну інформацію про клієнтів, які каталися з 2 по 12 січня.
2. Знайти, скільки сноубордів було видано визначеним працівником.
3. Обчислити загальну суму за прокат спорядження у січні.
4. Вивести повну інформацію про під'йомник, на якому було найбільше користувачів у визначений день.
5. Дізнатися повну інформацію про клієнтів, що придбали абонемент, проте не брали в оренду спорядження.

Варіант 12.2

1. Вивести інформацію про робітників, які працювали у зазначений день.
2. Дізнатися, скільки абонементів кожного типу було видано.

3. Визначити, на яке гірськолижне спорядження не було попиту.
4. Вивести повну інформацію про під'йомники, якими скористався клієнт із визначеним номером абонемента.
5. Якщо безлімітний абонемент було придбано у лютому, зменшити його вартість на 10 %.

Варіант 12.3

1. Вивести інформацію про клієнтів, які у січні брали на прокат лижі фірми Fisher.
2. Визначити, хто з працівників обслуговував конкретного клієнта.
3. Вивести перелік гірськолижного спорядження, яке замовляли більше одного разу.
4. Знайти дані про клієнтів, які на поточну дату вичерпали кількість підйомів, передбачених у абонементі.
5. Вивести дані про клієнтів, які виконали більше 15 підйомів у зазначений день.

Варіант 13
«Служба таксі»

Розробіть інформаційну систему централізованої диспетчерської служби таксі. Передбачте можливість збереження інформації про водіїв, автомобілі (номер, марка, рік випуску, строк дії техогляду тощо), постійних клієнтів (номер клієнта, ПІБ, адреса проживання, телефон та ін.). Водії працюють у визначені дні тижня по змінах та протягом дня можуть перебувати в різних локаціях в очікуванні виклику. Система повинна надавати можливість клієнтові завчасно замовити таксі, одержати інформацію про вартість послуги та час очікування, зберігати інформацію про виклики, відстежувати виконання замовлення та їх оплату.

Інформаційна система, що проектується, повинна вести облік: працівників, клієнтів, авто, локацій, замовлень тощо.

Варіант 13.1

1. Знайти повну інформацію про клієнтів, які 5 жовтня замовляли таксі на вул. Заставнянську.

2. Визначити кількість викликів, прийнятих на визначеній локації.
3. Вивести інформацію про клієнтів, які перевозили на авто з визначеним номерним знаком.
4. Вивести детальну інформацію про те, за якими адресами виїзджали таксисти, що працювали у нічну зміну.
5. Заборонити видалення інформацію про таксиста, якщо для нього складено розклад роботи.

Варіант 13.2

1. Вивести інформацію про водіїв, які їздять на автомобілях марки Audi.
2. Дізнатися, скільки викликів таксі було прийнято від кожного постійного клієнта за останній місяць.
3. Визначити, які авто очікували на виклик у тій самій локації, що й автомобіль конкретного водія.
4. Вивести інформацію про клієнтів, які викликали таксі в різних локаціях.
5. Вивести повну інформацію про водіїв, які за день не виконали жодного замовлення.

Варіант 13.3

1. Вивести інформацію про автомобілі, рік випуску яких менший за 2001 р.
2. Скільки таксистів працює на кожній локації у визначений день тижня.
3. Видалити інформацію про автомобілі, у яких сплинув термін техогляду.
4. Вивести повну інформацію про клієнтів, які скористалися послугою служби таксі більше п’яти разів.
5. Одержати інформацію про адреси, за якими виїжджав деякий таксист протягом дня.

Варіант 14
Інформаційна система «Поліклініка»

Розробіть інформаційну систему, яка б дала змогу автоматизувати, централізовано зберігати та обробляти інформацію в міській поліклініці. Система повинна

опрацьовувати загальну інформацію про лікарів, пацієнтів, кабінети та лікувальні процедури, а також здійснювати облік звернень, обстежень та лікування пацієнтів (історія хвороби, діагноз, призначені аналізи, процедури, ліки тощо). У поліклініці діє визначений графік роботи лікарів та процедурних кабінетів (позмінно у визначений день тижня).

Інформаційна система, що проектується, повинна вести облік: пацієнтів, лікарів, лікувальних процедур та можливих діагнозів, розкладу роботи та звернень пацієнтів.

Варіант 14.1

1. Знайти повну інформацію про графік роботи хірургів на тиждень.
2. Визначити, скільки звернень від пацієнтів прийнято кожним лікарем.
3. Знайти інформацію про хворих, які 15 листопада робили флюорографію.
4. Вивести детальну інформацію про лікарів та лікування, яке вони призначали пацієнтам.
5. Одержати інформацію про результати аналізу крові пацієнтів за останню добу.

Варіант 14.2

1. Вивести інформацію про лікування, які приймали пацієнти, що проживають на вул. Головній.
2. Дізнатися, скільком хворим було поставлено діагноз «ангіна» протягом місяця.
3. Визначити, чи є лікарі, які протягом тижня працюють у різних кабінетах.
4. Вивести інформацію про пацієнтів, яким визначений лікар дав направлення на ЕКГ.
5. Додати для пацієнта новий курс лікування препаратом, який ще не був призначений лікарем-терапевтом.

Варіант 14.3

1. Виведіть інформацію про кабінети фізіотерапії, що працюють у середу та п’ятницю у другу зміну.

2. Знайти лікарів, до яких звернулося більше пацієнтів аніж до кардіолога.
3. Видалити інформацію про пацієнтів, які жодного разу не зверталися до лікарів поліклініки.
4. Виведіть повну інформацію про про лікарів, які призначали своїм пацієнтам антибіотики.
5. Додайте до розкладу нову заявку від пацієнта на прийом до визначеного лікаря згідно з розкладом, якщо на цей час немає інших записів.

Варіант 15
«Супермаркет»

Розробіть інформаційну систему для підтримки роботи супермаркету. Товари у магазині розподілені по окремих відділах для різних категорій продукції (наприклад, побутова хімія, канцтовари, кухонне приладдя, косметика, м'ясо, риба, молочні продукти, борошняні вироби, овочі тощо). Кожен товар, окрім інших характеристик, має визначений термін зберігання. У різні дні тижня працівники можуть працювати позмінно у різних відділах згідно з розкладом. Супермаркет співпрацює з декількома фірмами-постачальниками, з якими укладають угоди на постачання продукції. Продаж товарів здійснюється лише після їх завезення до супермаркету. Періодично на певні види продукції воддяться знижки.

Інформаційна система, що проектується, повинна вести облік: категорій товарів, продукції, працівників, постачальників, розкладу роботи, проданих товарів тощо.

Варіант 15.1

1. Знайти повну інформацію про постачальників, які відвантажують більше трьох видів продукції.
2. Визначити кількість проданих товарів одного типу, зокрема деякої марки прального порошку.
3. Обчислити загальну суму виторгу за день у кожному відділі супермаркету.
4. Вивести повну інформацію про постачальників, які не надали продукцію на дату, визначену в укладеному договорі.

5. Одержати інформацію про продавців, які працювали більше ніж в одному відділі супермаркету.

Варіант 15.2

1. Вивести інформацію про продавців, які працюють у відділі «Овочі та фрукти» у першу зміну в зазначений день.
2. Дізнатися про запаси продукції марки «Дві корівки» у молочних відділах на поточну дату.
3. Визначити відділ супермаркету з максимальною сумою виторгу за місяць.
4. Вивести повну інформацію про постачальників визначеного виду продукції.
5. Додати інформацію про 15 % знижку на м’ясні вироби, термін придатності яких спливає за 5 днів.

Варіант 15.3

1. Вивести інформацію про товари, у яких закінчується термін дії на визначену дату.
2. Визначити, скільки разів кожен працівник виходив у третю зміну.
3. Видалити інформацію про товари, на постачання яких не укладено договір.
4. Вивести повну інформацію про постачальників, які постачають більше товарів, аніж компанія «Файний газда».
5. Зменшити вартість усіх кондитерських виробів на 20 % у грудні.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ANSI – *American National Standards Institute*, Американський інститут національних стандартів

ACID – *Atomicity, Consistency, Isolation, Durability,* атомарність, узгодженість, ізоляція, надійність

BLOB – *Binary Large Objects*, великі бінарні об'єкти

CAP – *Consistancy, Availability, Partition tolerance,* узгодженість, доступність, стійкість до розподілення

CASE – *Computer-Aided Software Engineering*, набір інструментів і методів програмної інженерії для проектування програмного забезпечення

CRUD – *Create, Read, Update, Delete,* створення, зчитування, оновлення, видалення

CSV – *Comma-Separated Values*, текстові файли для зберігання структурованої інформації

DCL – *Data Control Language*, мова керування даними

DDL – *Data Definition Language*, мова визначення даних

DML – *Data Manipulation Language*, мова маніпулювання даними

ERD – *Entity Relationship Diagram*, діаграма «сутність–зв'язок»

FK – *Foreign Key*, зовнішній ключ

NF – нормальна форма

NoSQL – *non (relational) SQL, not only SQL*, нереляційні БД

PK – *Primary Key*, первинний ключ

SPARC – *Standards Planning and Requirements Committee*, Комітет зі стандартів планування та вимог

SQL – *Structured Query Language*, мова структурованих запитів

TCL – *Transaction Control Language*, мова керування транзакціями

БД – база даних

ЕОМ – електронно-обчислювальна машина

МД – модель даних

НФ – нормальна форма

ПрО – предметна область

РМ – реляційна модель

СУБД – система управління базами даних

ТФС – традиційна файлова система

ФЗ – функціональна залежність

1:1 – зв’язок «один-до-одного»

1:M – зв’язок «один-до-багатьох»

M:N – зв’язок «багато-до-багатьох»

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Date C. J An introduction to database systems (8th edition). Pearson Education Inc., 2020. 1328 p.
2. Товстий та тонкий клієнт. Що це таке та в чому різниця? URL : https://training.qatestlab.com/blog/technical-articles/thick-and-thin-client/
3. 11 типів сучасних баз даних: короткий опис, схеми і приклади БД. URL : https://senior.ua/articles/11-tipv-suchasnih-baz-danih-korotkiy-opis-shemi--prikladi-bd .
4. Comprehensive NoSQL tutorial for beginners. URL : https://www.gologica.com/elearning/comprehensive-nosql-tutorial-for-beginners/ .
5. NoSQL – переваги та недоліки нереляційних баз даних. URL : https://qagroup.com.ua/publications/nosql-perevagy-ta-nedoliky-nereliatcijnykh-baz-danykh/
6. Бази даних. URL : https://www.bestprog.net/uk/sitemap_ua/бази-даних/ .
7. Introduction to databases. URL : https://www.prisma.io/dataguide/intro .
8. Кураєв Я.Г. Реляційні бази даних. URL : https://rdb.dp.ua/uk/mnp .
9. Підручник з нормалізації баз даних. URL : https://uk.myservername.com/database-normalization-tutorial .
10. Гайдаржи В., Ізварін І. Бази даних в інформаційних системах. Київ : Університет «Україна», 2018. 418 с.
11. Пасічник В.В., Резніченко В.А. Організація баз даних. Київ : Видавнича група BHV, 2006. 384 с.
12. Beaulieu A. Lerning SQL: Generate, Manipulate, and Retrieve Data, 3rd edition. O'Reilly Media, 2020. 532 p.
13. Holywell S. Посібник зі стиль-коду SQL – SQL style guide. URL : https://www.sqlstyle.guide/ua/ .
14. Gruber M. Understanding SQL – Sybex, 2003. 434 p.

15. SQL reserved words. URL : https://docs.actian.com/psql/psqlv13/index.html#page/sqlref/sqlkword.htm .
16. Date types (Transact-SQL). URL : https://learn.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15 .
17. Shields W. SQL QuickStart Guide: The simplified begginers guide to Managing, Analyzing, and manipulating data with SQL. ClydeBank Media LLC, 2019. 249 p.
18. SQL tutorial. URL : https://www.w3schools.com/sql/default.asp .
19. SQL interactive courses. URL : https://academy.vertabelo.com/.
20. Into to SQL: querying and managing data. URL : https://www.khanacademy.org/computing/computer-programming/sql .
21. SQL coding questions. URL : https://sqlpad.io/?via=geekflare .
22. 17 sites for SQL practice. URL : https://www.databasestar.com/sql-practice/ .
23. SQLBolt: Learn SQL with simple, interactive exercises. URL : https://sqlbolt.com/lesson/introduction .
24. SQL Tutorial. URL : https://sqlzoo.net/wiki/SQL_Tutorial .
25. The Data School. Learn SQL. URL : https://dataschool.com/learn-sql/ .

Навчальне видання

Танасюк Юлія Володимирівна
Одайська Христина Савеліївна

ОРГАНІЗАЦІЯ БАЗ ДАНИХ

Навчальний посібник

Літературний редактор О. В. Колодій

Підписано до друку 09.12.2022 р. Формат 60 х 84/16
Ум. друк. арк. 11,4.
Обл.-вид. арк. 12,2. Тираж 100