БАКАЛАВРСЬКА РОБОТА

БР.ІП – 20.00.00.000 ІПЗ

Група ІП-22-1

Нікутін Борис

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Нікутін Борис Андрійович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Розроблення програмного забезпечення з використанням реляційних баз даних
(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Нікутін Б.А._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник ____ Шекета Василь Іванович, професор_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Нікутіну Борису Андрійовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Розроблення програмного забезпечення з використанням реляційних баз даних "

керівник проекту (роботи) проф. Шекета В.І.

затверджені наказом вищого навчального закладу від “ 21_ ” _квітня 2026 р. № ___179/7____

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз вимог до програмного забезпечення
2. Аналіз вимог і постановка задачі
3. Проектування системи
4. Реалізація проекту
5. Тестування та впровадження

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1 NoSQL — основні особливості (рис. 1.1, ст. 11)

2 Комбінації CAP (рис. 1.2, ст. 13)

3 Частина описів класів для Країни та Полігону (рис. 2.5, ст. 28)

4 Стан, методи та повідомлення об'єкта. (рис. 2.6, ст. 29)

5 Складні об'єкти (рис. 2.8, ст. 32)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2026 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	17.03.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.04.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.05.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.05.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	09.06.2026	виконано

Студент _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 85 сторінок, 41 рисунок, 10 таблиць, список використаних джерел із 40 найменувань.

Метою роботи є розроблення програмного забезпечення з використанням реляційних баз даних.

Об'єкт дослідження: програмні системи, що використовують реляційні бази даних для зберігання та обробки інформації.

Предмет дослідження: методи, технології та інструменти проєктування програмного забезпечення з використанням реляційних БД, мови запитів SQL.

Результати дослідження: розроблено програмне забезпечення, яке реалізує взаємодію з реляційною базою даних, підтримує виконання SQL-запитів, забезпечує цілісність даних та ефективне управління ресурсами.

У першому розділі проведено аналіз теоретичних основ проєктування програмного забезпечення та реляційних баз даних, розглянуто моделі даних і принципи побудови реляційної моделі.

У другому розділі досліджено методи та інструменти розробки програмного забезпечення з використанням реляційних баз даних, розглянуто взаємодію з базами даних за допомогою SQL.

У третьому розділі реалізовано програмний додаток, досліджено механізми трансляції та оптимізації реляційних запитів, а також проведено оцінку ефективності роботи розробленого програмного забезпечення.

Висновок: розроблене програмне забезпечення демонструє ефективність використання реляційних баз даних для зберігання та обробки інформації, забезпечує надійність, структурованість даних і можливість масштабування.

КЛЮЧОВІ СЛОВА: РЕЛЯЦІЙНА БАЗА ДАНИХ, SQL, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МОДЕЛЬ ДАНИХ, СИСТЕМА КЕРУВАННЯ БАЗАМИ ДАНИХ, ПРОЄКТУВАННЯ БАЗ ДАНИХ, ОПТИМІЗАЦІЯ ЗАПИТІВ, ІНФОРМАЦІЙНА СИСТЕМА.

ANNOTATION

The bachelor's thesis contains of 85 pages, 41 figures, 10 tables, and a reference list containing 40 sources.

The aim of the work is to develop software using relational databases that provides efficient data storage, processing, and management within an information system.

Research object: software systems that use relational databases for storing and processing information.

Research subject: methods, technologies, and tools for designing software using relational databases, SQL query languages.

Research results: software was developed that implements interaction with a relational database, supports SQL query execution, ensures data integrity, and provides efficient management of information resources.

The first section analyzes the theoretical foundations of software design and relational databases, examines data models, and studies the principles of relational data modeling.

The second section investigates methods and tools for software development using relational databases, considers database interaction through SQL, object-oriented data models, and approaches to conceptual and logical database design.

The third section presents the implementation of a software application, explores mechanisms for translation and optimization of relational queries, and evaluates the efficiency of the developed software solution.

Conclusion: the developed software demonstrates the effectiveness of relational databases for data storage and processing, ensuring reliability, data consistency, and scalability according to user requirements.

KEY WORDS: RELATIONAL DATABASE, SQL, SOFTWARE, DATA MODEL, DATABASE MANAGEMENT SYSTEM, DATABASE DESIGN, QUERY OPTIMIZATION, INFORMATION SYSTEM.

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві дані є одним із найважливіших ресурсів, а ефективне їх зберігання, обробка та аналіз відіграють ключову роль у функціонуванні програмних систем різного призначення. Зростання обсягів інформації, потреба у швидкому доступі до даних та забезпеченні їхньої цілісності обумовлюють широке використання реляційних баз даних у процесі розроблення програмного забезпечення. Реляційна модель даних залишається однією з найбільш поширених завдяки своїй надійності, структурованості та підтримці потужних механізмів обробки запитів.

Актуальність теми зумовлена необхідністю створення програмного забезпечення, здатного ефективно працювати з великими обсягами структурованих даних, забезпечуючи їх цілісність, безпеку та швидке виконання операцій. Сучасні інформаційні системи потребують використання реляційних баз даних для підтримки бізнес-процесів, автоматизації обліку, управління ресурсами та обробки інформації в режимі реального часу. Незважаючи на появу альтернативних підходів до зберігання даних, реляційні бази даних залишаються основою більшості корпоративних програмних рішень.

Метою роботи є розроблення програмного забезпечення з використанням реляційних баз даних, яке забезпечує ефективне зберігання, обробку та керування даними в інформаційній системі.

Для досягнення поставленої мети необхідно виконати **такі завдання:** проаналізувати теоретичні основи реляційних баз даних та моделей даних; дослідити методи взаємодії програмного забезпечення з базами даних за допомогою SQL; розглянути особливості об'єктно-орієнтованих моделей даних; виконати концептуальне та логічне проєктування бази даних; реалізувати тестовий програмний додаток; дослідити методи трансляції та оптимізації реляційних запитів; провести оцінку ефективності розробленого програмного забезпечення.

Об’єктом дослідження є процеси розроблення програмного забезпечення з використанням реляційних баз даних.

Предметом дослідження є методи, моделі, технології та інструменти проєктування, реалізації та оптимізації програмного забезпечення, що використовує реляційні бази даних.

Методи дослідження включають аналіз наукової та технічної літератури, методи моделювання даних, порівняльний аналіз технологій роботи з базами даних, проєктування програмних систем, експериментальні методи реалізації та тестування програмного забезпечення.

Практичне значення роботи полягає у створенні програмного забезпечення, що демонструє ефективне використання реляційних баз даних для зберігання та обробки інформації, а також застосування методів оптимізації запитів для підвищення продуктивності системи. Особлива увага приділяється забезпеченню цілісності даних, надійності функціонування та можливості подальшого масштабування програмного продукту.

Бакалаврська робота містить 85 сторінок, 41 рисунок, 10 таблиць, 3 розділи, висновки та список використаних джерел із 40 найменувань.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ РЕЛЯЦІЙНИХ БАЗ ДАНИХ

1.1 Основи проєктування програмного забезпечення та баз даних

Архітектура конкретного програмного застосунку враховує нефункціональні характеристики, такі як надійність, зручність використання, масштабованість, продуктивність, сумісність, портативність, адаптивність та шардування даних. Завжди існують компроміси між набором атрибутів якості, і архітектор програмного забезпечення стикається зі складним завданням їх балансування. Системи великих даних [1] є внутрішньо розподіленими. Труднощі з доступністю та узгодженістю даних виникають через шардування даних та реплікацію у величезних системах даних. Через зростаюче розширення застосунків для обробки даних технології баз даних зазнали суттєвих змін . Протягом понад десятиліття бази даних NoSQL зростали в геометричній прогресії, хоча класична автоматизація баз даних зберігалася. Традиційні макети змушують використовувати жорстку структуру схеми, що призводить до неясності масштабування та перешкоджає модифікації даних між кластерами. На противагу цьому, бази даних NoSQL підтримують прості прототипи. Основні властивості проектів баз даних NoSQL включають:

- Структура без схеми
- Дозвіл на ефективне та динамічне зростання представлень даних
- Горизонтальне масштабування шляхом колекцій реплікації даних та шардингу на масивних кластерах.

За останні роки численні організації накопичили величезні обсяги даних, які реляційні бази даних не можуть ефективно обробляти. За останні чотири десятиліття було спостерігається величезне зростання використання реляційних баз даних. Вони дотримуються атрибута ACID (доступність, узгодженість, ізоляція та довговічність) та розроблені для структурованих даних. Хоча «великі

дані» включають інструменти та технології, які керують величезними обсягами даних у будь-якому масштабі, ці інструменти та технології є масштабованими. Великі дані включають 5V (об'єм, швидкість, різноманітність, достовірність та цінність) та величезну кількість неструктурованих даних з різноманітною природою. Численні фреймворки, включаючи Hadoop/MapReduce, Spark, Flink та Samza, використовуються для обробки великих даних [2].

Тема продуктивності та оптимізації SQL-запитів для корпоративних, виробничих, паралельних баз даних та великих даних отримала підвищену увагу в останні роки. Неефективні та неоптимізовані запити можуть споживати системні та серверні ресурси, що призводить до блокування бази даних та проблем втрати даних. Інформаційний інтелектуальний аналіз передбачає вилучення фактів та логічної кореляційної структури з початкового дельта-множини, на відміну від самої інформації. Оптимізація запитів стосується вибору оптимальної стратегії виконання запитів з мінімальними витратами та споживанням системних ресурсів. Алгоритми інтелектуального аналізу даних виконують глибокі та розширені запити до бази даних для вилучення шаблонів та знань з комплексних даних [3]. Загальнопромислові методології, такі як XML та об'єктні бази даних, ніколи не досягали такого ж рівня популярності, як технологія RDBMS.

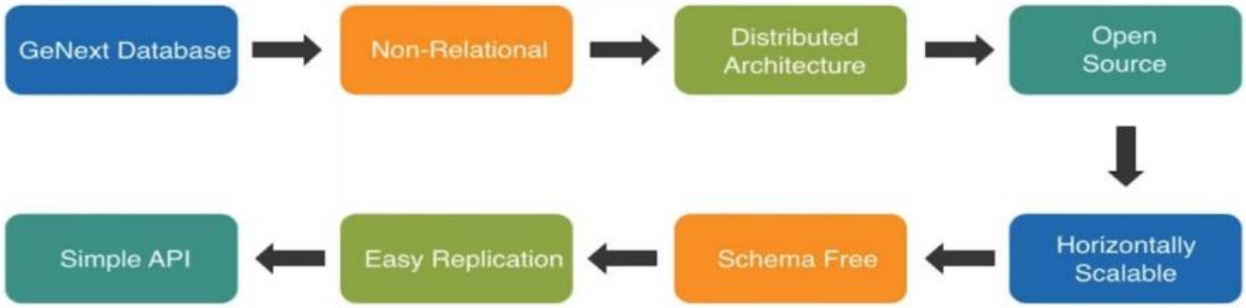

Рисунок 1.1 - NoSQL — основні особливості.

Протягом останнього десятиліття науковці та онлайн-продавці часто ставили під сумнів універсальний характер технології обробки даних. Такий підхід призвів до розробки нової альтернативної системи баз даних, відомої як NoSQL, що розшифровується як «не тільки SQL». NoSQL описує використання

веб-розробниками нереляційних баз даних [4]. У 1998 році термін NoSQL був вперше використаний, і конференція з нереляційних баз даних у Сан-Франциско привернула до нього більше уваги. Рисунок 1.1 описано ключові аспекти NoSQL баз даних.

Таблиця 1.1

Теорія CAP.

Consistency (Узгодженість даних)	**Availability (Доступність)**	**Partition Tolerance (Стійкість до розподілення)**
• Узгодженість означає, що дані в базі даних залишаються узгодженими після виконання операції.	• Доступність означає, що система працюватиме без простоїв (гарантується 100% часу безперебійної роботи сервісу).	• Стійкість до розподілення означає, що система продовжує функціонувати, навіть якщо зв'язок між серверами є ненадійним.
• Наприклад, після операції оновлення всі клієнти бачать одні й ті самі дані.	• Кожен вузол (якщо він не вийшов з ладу) завжди виконує запит.	• Сервери можуть бути розділені на кілька груп, які не можуть спілкуватися одна з одною.

Нижче наведено переваги NoSQL баз даних:

- Обсяг: Дані в стані спокою — терабайти до ексабайтів існуючих даних для обробки.
- Швидкість: Дані в русі — потокові дані, час відповіді від мілісекунд до секунд.
- Мінливість: дані в багатьох формах — структуровані, неструктуровані, текстові тощо.
- Достовірність: дані під сумнівом — невизначеність через затримку, обман, неоднозначності тощо.
- Не побудовано на таблицях і не використовує SQL для маніпулювання даними.
- Схема включає ключ-значення, документ, стовпцеву схему, схему графа тощо.
- Альтернатива традиційним реляційним базам даних.
- База даних для обробки неструктурованих, невпорядкованих та

непередбачуваних даних.

- Корисно для роботи з великими наборами розподілених даних.

Теоретично, неможливо виконати всі три вимоги. Тому CAP забезпечує основну потребу для розподіленої системи, щоб вона відповідала двом із трьох елементів. Отже, всі сучасні NoSQL бази даних підтримують різні комбінації C, A та P теорії CAP, як описано на рисунку 1.2.

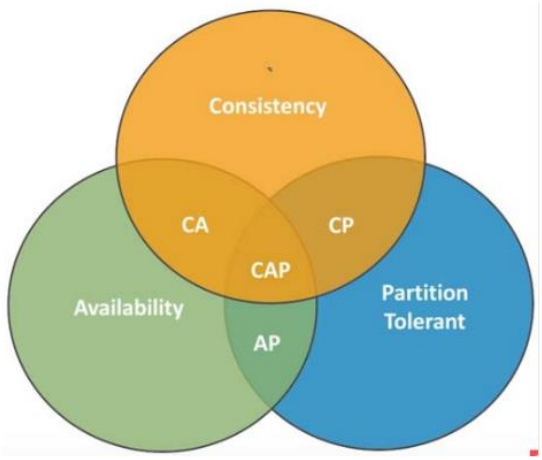

Рисунок 1.2 - Комбінації CAP.

NoSQL-бази даних відрізняються від реляційних баз даних і мають власні моделі та архітектури. Під час зберігання NoSQL-баз даних у хмарі необхідно бути обережним через різноманітність цих баз даних та необхідність забезпечення сумісності, портативності та заходів безпеки. Хмарні постачальники послуг (CSP) пропонують масштабованість , доступність та конфіденційність, але важливо шифрувати конфіденційні дані користувачів перед наданням доступу CSP. Це може бути складно через різноманітний характер NoSQL-баз даних. База даних як послуга (DBaaS) [4] – це платформа tloud, яка перетворює традиційні архітектури на хмарні.

Основний внесок цієї SLR arc IHC наступний:

- Цей односторонній документ пов'язаний з оцінкою архітектури баз даних SQL та NoSQL, можливостями масштабування та аналізом продуктивності, зокрема Oracle RDBMS та NoSQL Document Database (MongoDB). Крім того, досліджується переміщення даних між різними базами даних на кількох хмарних

платформах.

У цій роботі визначено прогалини в дослідженнях пов'язаних архітектур та їх причини.

Стан проблеми

Поширення великих даних призвело до потреби в масштабованих системах, які можуть ефективно обробляти величезні обсяги даних. Реляційні бази даних, засновані на SQL, можуть певною мірою керувати структурованими, напівструктурованими та неструктурованими даними, але мають обмеження щодо масштабованості. З іншого боку, NoSQL бази даних, які відповідають властивості BASE та можуть розширювати свою ємність сховища горизонтально, краще підходять для керування величезними обсягами даних та адаптації до змін типу та структури даних.

Існує чотири типи NoSQL баз даних — бази даних типу «ключ-значення», документи, стовпці та графи — кожна з яких має свої особливості та застосування. Наприклад, графова база даних NoSQL зберігає інформацію у вузлах, а не в таблицях, і зберігає асоціації (з'єднання) між вузлами, що вимагає постійного часу виконання. Це дає їй перевагу над базами даних SQL під час обробки високо взаємопов'язаних даних [5].

MongoDB є прикладом NoSQL бази даних, яка ефективно керує величезними обсягами даних завдяки своїм чудовим характеристикам масштабованості. Однак перетворення з OLTP- баз даних до MongoDB може спричинити проблеми, такі як унікальні індекси, складені ключі, невідповідність даних та дублювання даних через складність їхньої схеми.

Ще однією важливою проблемою при передачі даних у хмарне сховище є захист конфіденційної інформації користувачів. Поточні проекти постачальників хмарних послуг (CSP) розробляються без урахування питань сумісності та портативності, що ускладнює пропонування та створення єдиного хмарного рішення для моделей NoSQL [6].

Нарешті, у цьому SLR детально розглядаються найсучасніші методи та політики безпеки для NoSQL баз даних.

Метод

Цей SLR відповідає рекомендаціям PRISMA та має на меті синтезувати високоякісні первинні дослідження на основі доказів, пов'язаних з конкретними дослідницькими питаннями. SLR широко використовуються в дослідженнях у галузі програмної інженерії, а наше дослідження зосереджене на оцінці архітектури баз даних SQL та NoSQL та оцінці продуктивності, а також на переносимості даних та сумісності між кількома хмарними платформами. Характеристики, що відрізняють наше дослідження від існуючих, - це системний процес збору даних, вичерпний перелік охоплених досліджень, цілеспрямований обсяг дослідження та детальна класифікація та аналіз вибраних досліджень.

1.2 Моделі даних у сучасних інформаційних системах

Модель даних надає набір конструкцій для опису та структурування програм у базі даних. Її метою є забезпечення спільного обчислювально значущого середовища для використання розробниками систем та користувачами. Для розробників модель даних надає засоби представлення області застосування в термінах, які можна перевести в проект та реалізацію системи. Для користувачів вона надає опис структури системи, незалежно від конкретних елементів даних або деталей конкретної реалізації.

Слід чітко розрізняти моделі даних, на яких побудовані системи баз даних, наприклад, реляційну модель, та моделі даних, основна роль яких полягає в тому, щоб якомога точніше відображати значення областей застосування (так звані *семантичні моделі даних,* прикладом яких є моделювання «сутність-зв'язок») [7]. Можливо, семантична модель даних використовується для розробки застосунків для системи баз даних, спроектованої на основі іншої моделі: прототипним прикладом цього є використання моделі «сутність-зв'язок» для розробки застосунків реляційних баз даних. Три найважливіші підходи до моделювання даних наразі - це *записно-орієнтований, об'єктно-орієнтований* та *об'єктно-реляційний.*

Взаємодія людини з базою даних

Людям необхідно взаємодіяти з системами баз даних для виконання таких широких типів завдань:

1 *Визначення даних:* опис концептуальної та логічної організації бази даних, схема бази даних;

2 *Визначення сховища:* опис фізичної структури бази даних, наприклад, розташування файлів та методи індексування;

3 *Адміністрування бази даних:* щоденна експлуатація бази даних;

4 *Маніпулювання даними:* вставка, модифікація, пошук та видалення даних з бази даних.

Перші три з цих завдань, найімовірніше, виконуватимуть фахівці з баз даних, тоді як четверте буде потрібне різноманітним типам користувачів, які мають різноманітні навички та досвід, а також різні потреби щодо частоти та гнучкості доступу.

Інтерфейси користувача розроблені достатньо гнучкими, щоб впоратися з таким розмаїттям використання. Стандартні методи створення більш природних інтерфейсів для користувачів включають меню, форми та графіку. Природна мова була б відповідним засобом зв'язку між людиною та базою даних, але успішних інтерфейсів на основі природної мови ще не досягнуто. Для просторових даних, звичайно, дуже доречним є графічний інтерфейс користувача (GUI). Були розроблені спеціалізовані мови запитів для взаємодії з базою даних.

Управління базами даних

Програмна система, що керує базою даних, називається *системою керування базами даних* (СКБД) [8]. На рисунку 1.3 схематично показано місце деяких із цих компонентів в обробці інтерактивного запиту або прикладної програми, яка містить у своїй мові програмування загального призначення деякі команди доступу до бази даних. СКБД має компілятор запитів, який розбирає та аналізує запит і, якщо все правильно, генерує код виконання, який передається до процесора бази даних. У процесі компілятор може викликати оптимізатор запитів для оптимізації коду, щоб покращити продуктивність пошуку. Якби вираз мови

бази даних був вбудований у мову програмування загального призначення, таку як C++, то знадобився б попередній етап передкомпілювання. Щоб отримати необхідні дані з бази даних, необхідно виконати зіставлення між об'єктами високого рівня в операторі мови запитів та фізичним розташуванням даних на пристрої зберігання даних. Ці зіставлення виконуються за допомогою системного каталогу. Доступ до даних СКБД обробляється менеджером збережених даних, який звертається до операційної системи для керування фізичним доступом до пристроїв зберігання даних.

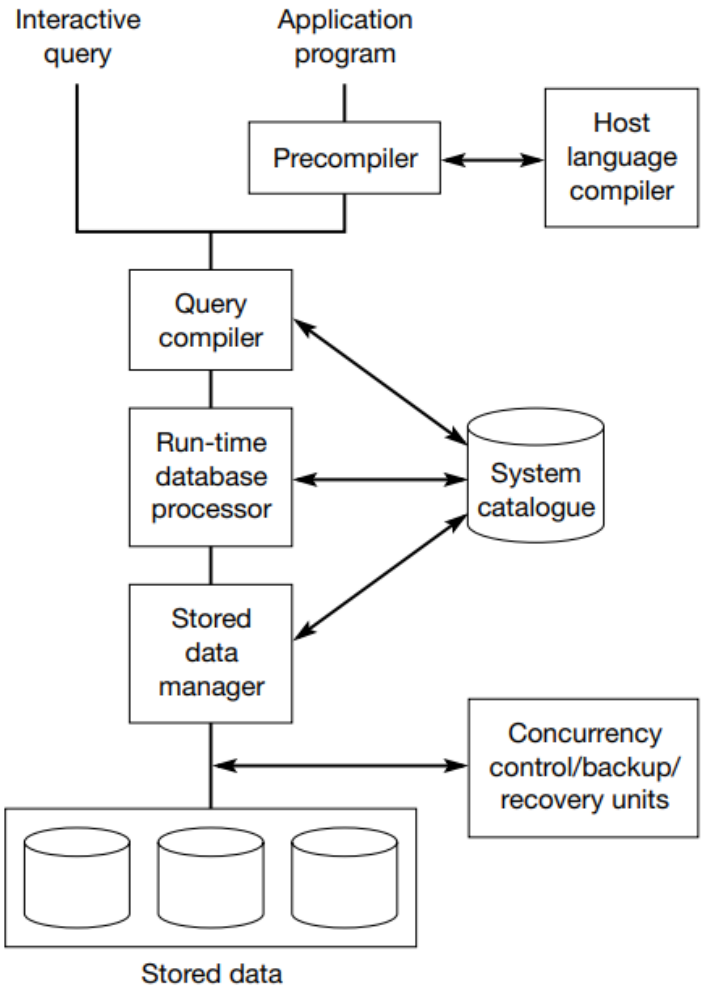

Рисунок 1.3 - Компоненти СУБД, що використовуються для обробки запитів користувачів.

Логічним атомом взаємодії з базою даних є транзакція , яку можна загалом класифікувати як *створення, зміна (оновлення)* та *видалення.* Транзакції або

виконуються повністю (фіксуються), або взагалі не виконуються (відкат до попереднього фіксування). Послідовність операцій, що містяться в транзакціях, зберігається в системному журналі, звідси й здатність СУБД здійснювати відкат. Коли досягнуто «фіксації», усі зміни з моменту останньої точки фіксації стають постійними в базі даних. Таким чином, транзакцію можна розглядати як одиницю відновлення. СУБД прагне підтримувати так звані ACID-властивості транзакцій: атомарність (все або нічого),

Узгодженість (бази даних), ізоляція (відсутність побічних ефектів та непередбачених наслідків для інших одночасних транзакцій) та довговічність (здатність вижити навіть після збою системи) [9].

1.3 Реляційна модель даних та її особливості

Модель на основі записів структурує базу даних як набір файлів записів фіксованого формату. Записи у файлі мають однаковий тип запису та містять фіксований набір полів (атрибутів). Ранні мережеві та ієрархічні системи баз даних, згадані раніше, відповідають моделі даних на основі записів. Однак вони виявилися занадто тісно пов'язаними з деталями фізичної реалізації, і їх значною мірою замінила реляційна модель.

Реляційна база даних — це колекція табличних відношень, кожне з яких має набір атрибутів. Дані у відношенні структуровані як набір рядків. Рядок, або *кортеж* , складається зі списку значень, по одному для кожного атрибута. Атрибут пов'язаний з доменом, з якого беруться його значення. Більшість сучасних систем вимагають, щоб значення були атомарними — наприклад, їх не можна розкласти на списки подальших значень — тому одна комірка у відношенні не може містити набір, список або масив значень [10]. Це обмежує можливості чисто реляційної моделі для ГІС.

Розрізняють *схему відношення,* яка не містить даних, але визначає структуру відношення (його атрибути, відповідні їм домени та будь-які обмеження на дані), та *відношення,* яке включає дані. Схема відношення зазвичай

оголошується під час налаштування бази даних, а потім залишається відносно незмінним протягом життєвого циклу системи.

Таблиця 1.1

Кортежі з відношення Країна.

Назва країни(Name)	**Населення, млн осіб(Population, millions)**	**Площа, тис. кв. миль(Land area, thous. sq. miles)**	**Столиця(Capital)**
Австрія *(Austria)*	8	32	Відень *(Vienna)*
Німеччина *(Germany)*	81	138	Берлін *(Berlin)*
Італія *(Italy)*	58	116	Рим *(Rome)*
Франція *(France)*	58	210	Париж *(Paris)*
Швейцарія *(Switzerland)*	7	16	Берн *(Bern)*

Однак відношення зазвичай часто змінюється, оскільки дані вставляються, змінюються та видаляються. *Схема бази даних* – це набір схем відношень, а *реляційна база даних* – це набір відношень, можливо, з деякими обмеженнями. Приклад схеми бази даних, що використовується в цьому розділі, складається з двох відношень: Країна та Місто, а також їхніх атрибутів, як показано:

Країна (Назва, Населення, Площа, Столиця) Місто (Назва, Країна, Населення).

У таблицях 1.1 та 1.2 показано частину прикладу бази даних відповідно до цієї схеми. Кожен рядок відношення в реляційній базі даних іноді називають кортежем.

Таблиця 1.2

Кортежі з відношення City.

Назва міста(Name)	**Країна(Country)**	**Населення, тис. осіб(Population, thousands)**
Відень *(Vienna)*	Австрія *(Austria)*	1 500
Берлін *(Berlin)*	Німеччина *(Germany)*	3 400
Гамбург *(Hamburg)*	Німеччина *(Germany)*	1 600
Рим *(Rome)*	Італія *(Italy)*	2 800
Мілан *(Milan)*	Італія *(Italy)*	1 400
Париж *(Paris)*	Франція *(France)*	2 100
Цюрих *(Zurich)*	Швейцарія *(Switzerland)*	300
Берн *(Bern)*	Швейцарія *(Switzerland)*	100

Примітивні операції, які може підтримувати реляційна база даних, – це традиційні операції над множинами об'єднання, перетину та різниці, а також характерні реляційні операції проєктування, обмеження, з'єднання та ділення.

Структура цих операцій та спосіб їх поєднання надаються реляційною алгеброю. Операції над множинами об'єднання, перетин та різниця працюють з відношеннями як з наборами кортежів. Операція проєктування застосовується до одного відношення та повертає нове відношення, яке має підмножину атрибутів оригіналу. Наприклад, у таблиці 1.3 показано країну.

Таблиця 1.3

Кортежі з відношення "Країна" після операції проекту.

Назва країни(Name)	**Населення, млн осіб(Population (millions))**
Австрія *(Austria)*	8
Німеччина *(Germany)*	81
Італія *(Italy)*	58
Франція *(France)*	58
Швейцарія *(Switzerland)*	7

Таблиця 1.4

Кортежі з відношення City після операції обмеження.

Назва міста(Name)	**Країна(Country)**	**Населення, тис. осіб(Population (thousands))**
Берлін *(Berlin)*	Німеччина *(Germany)*	3 400
Рим *(Rome)*	Італія *(Italy)*	2 800
Париж *(Paris)*	Франція *(France)*	2 100

Відношення, проектоване на його атрибути Name та Population [11]. Операція restrict діє на відношення, щоб повернути лише ті кортежі, які задовольняють задану умову. Наприклад, у таблиці 1.4 показано обмеження відношення City, яке витягує з відношення City ті кортежі, що містять міста з населенням понад два мільйони.

Таблиця 1.5

Кортежі з об'єднаних відносин «Країна» та «Місто».

Назва країни(Name)	Населення країни, млн(Country population (millions))	Площа, тис. кв. миль(Land area (thousand sq. miles))	Столиця(Capital)	Країна міста(Country city)	Населення міста, тис.(City population (thousands))
Австрія *(Austria)*	8	32	Відень *(Vienna)*	Австрія *(Austria)*	1 500
Німеччина *(Germany)*	81	138	Берлін *(Berlin)*	Німеччина *(Germany)*	3 400
Італія *(Italy)*	58	116	Рим *(Rome)*	Італія *(Italy)*	2 800
Франція *(France)*	58	210	Париж *(Paris)*	Франція *(France)*	2 100
Швейцарія *(Switzerland)*	7	16	Берн *(Bern)*	Швейцарія *(Switzerland)*	100

Операція join встановлює зв'язки між відношеннями, приймаючи два відношення як операнди, та повертає одне відношення. Відношення, показане в таблиці 1.5, є об'єднанням відношень Country та City, що зіставляє кортежі, коли вони мають однакові назви міст.

1.4 Висновок по розділу

У першому розділі досліджено теоретичні основи розроблення програмного забезпечення з використанням реляційних баз даних. Розглянуто основні принципи проєктування програмних систем і баз даних, проаналізовано сучасні моделі даних та їх роль в організації інформаційних ресурсів. Особливу увагу приділено реляційній моделі даних, її структурі, перевагам та особливостям використання в сучасних інформаційних системах. Проведений аналіз підтвердив актуальність застосування реляційних баз даних як надійного засобу зберігання та обробки даних, що створює основу для подальшого проєктування програмного забезпечення.

РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З РЕЛЯЦІЙНИМИ БАЗАМИ ДАНИХ

2.1 Взаємодія програмного забезпечення з реляційними базами даних за допомогою SQL

З самого початку існувала колекція спеціалізованих мов запитів для взаємодії з базами даних. Для реляційних баз даних структурована або стандартна мова запитів (SQL) є стандартом *де-факто* та *де-юре* . SQL може використовуватися самостійно якзасоби прямої взаємодії з базою даних або можуть бути вбудовані в мову програмування загального призначення. Найновішим стандартом SQL є SQL-92 (також званий SQL2: ISO 1992). Докладаються великі зусилля для переходу до SQL3 [12].

Визначення схеми за допомогою SQL

Компонент мови визначення даних SQL дозволяє створювати, змінювати та видаляти схеми відношень. Зазвичай схема відношень змінюється лише зрідка після того, як база даних працює. Схема відношень надає набір атрибутів, кожен з яких має пов'язаний з ним домен даних.

SQL дозволяє визначити домен за допомогою виразу CREATE DOMAIN.

Схема відношення створюється командою CREATE TABLE як набір атрибутів, кожен з яких пов'язаний з доменом, з додатковими властивостями, що стосуються ключів та обмежень цілісності. Наприклад, схема відношення City може бути створена командою:

```
CREATE TABLE    City
(Name           PlaceName,
Country         PlaceName,
Population      Population,
PRIMARY KEY     (Name)
```

Рисунок 2.1 – Створення таблиці

Цей оператор починається з надання схеми відношення (яку в SQL називають таблицею) імені City. Потім атрибути визначаються шляхом надання кожному імені та пов'язаного з ним домену (припускаючи, що ми вже створили домени PlaceName та Population). Первинний ключ, який служить для унікальної ідентифікації кортежу, далі задається як атрибут Name. Існують також команди SQL для зміни схеми відношення шляхом зміни атрибутів або обмежень цілісності та для видалення схеми відношення.

Маніпулювання даними за допомогою SQL

Після визначення схем та вставки даних у відношення, наступним кроком є отримання даних. Простий приклад отримання даних SQL, що призводить до відношення.

```
SELECT *
FROM City
WHERE Population > 2000000
```

Рисунок 2.2 – Отримання даних SQL

Речення SELECT вказує на атрибут, який потрібно отримати з відношення City (* вказує на всі атрибути), тоді як речення WHERE забезпечує умову обмеження. Реляційні об'єднання здійснюються шляхом дозволу більш ніж одного відношення (або навіть того самого відношення, яке викликається двічі з різними іменами) у реченні FROM . Наприклад, щоб знайти назви країн, столиці яких мають населення менше двох мільйонів осіб, використовуйте вираз:

```
SELECT Country.Name
FROM Country, City
WHERE Country.Capital = City. Name
AND City. Population < 2000000
```

Рисунок 2.3 – Складне отримання даних SQL

У цьому випадку перша частина речення WHERE забезпечує умову

об'єднання, вказуючи, що кортежі з двох таблиць мають об'єднуватися лише тоді, коли значення атрибутів Capital (Столиця) у Country (Країна) та Name (Назва) у City (Місто) рівні. Атрибути кваліфікуються шляхом додавання префікса перед назвою відношення у разі будь-якої неоднозначності.

Більшість функцій SQL було опущено в цьому дуже короткому огляді. Документація щодо стандарту SQL2 становить близько 600 сторінок. Читачеві рекомендується звернутися до роботи для отримання гарного огляду реляційної моделі та SQL2.

Реляційні технології для географічної інформації

Існує два по суті способи управління просторовими даними за допомогою реляційної технології: розміщення всіх даних (просторових і непросторових) у реляційній базі даних (інтегрований підхід) або відділення просторових даних від непросторових (гібридний підхід). Переваги використання інтегрованої архітектури значні, що дозволяє СУБД однаково обробляти всі дані, і таким чином не приписувати просторовим даним менш захищене існування поза базою даних, де цілісність, паралельність і безпека можуть не бути так суворо забезпечені. Теоретично, інтегрований підхід цілком можливий: наприклад,

Пропонується реляційну модель конфігурацій вузлів, дуг та полігонів. Однак на практиці чисто реляційна геопросторова модель досі не отримала широкого застосування через неприйнятну продуктивність. По суті, проблеми виникають через:

1 повільне отримання даних через необхідність множинних об'єднань просторових даних у відношеннях;

2 невідповідні індекси та методи доступу, які надаються переважно для одновимірних типів даних реляційними системами загального призначення;

3 відсутність виразних можливостей SQL для просторових запитів.

Перша проблема виникає через те, що просторові дані є фундаментально складними – полігони є послідовностями ланцюгів, які самі є послідовностями точок. Об'єктно-орієнтовані та розширені реляційні моделі набагато краще обробляють такі типи даних. Щодо другої проблеми, розширені реляційні моделі

забезпечують набагато більшу гнучкість в оголошенні індексів для різних типів даних. Щодо третьої проблеми, обмеження SQL були очевидними вже деякий час у низці галузей (наприклад, CAD/CAM, ГІС, мультимедійні бази даних, офісні інформаційні системи та текстові бази даних). SQL3, який зараз розробляється як стандарт, багатообіцяючий у цьому відношенні.

2.2 Об'єктно-орієнтовані моделі даних та їх інтеграція з реляційними базами даних

Основними компонентами об'єктно-орієнтованої моделі є її *об'єкти* або *сутності.* Модель «сутність-зв'язок-атрибут» (ERA) та об'єктно-орієнтовані (ОО) моделі – це два основні підходи до об'єктно-орієнтованого моделювання. Підхід ERA є основним інструментом моделювання для реляційних систем баз даних протягом близько 20 років. У підході ERA сутність – це конструкція семантичного моделювання даних і є чимось (наприклад, країною), що має незалежне та унікально ідентифіковане існування в області застосування [13].

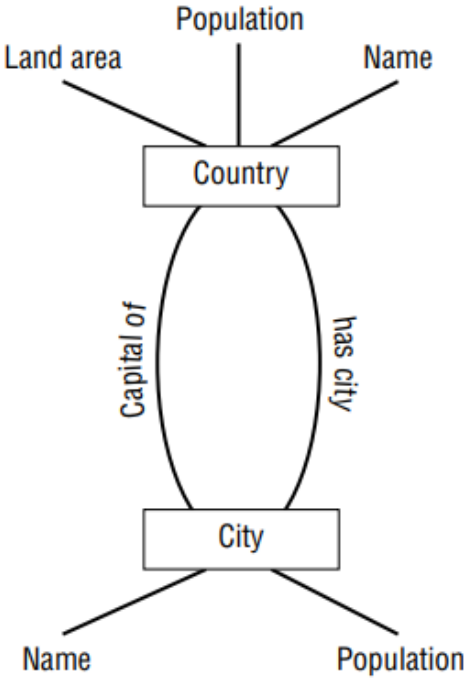

Рисунок 2.4 - Приклад діаграми ERA.

Сутності описуються за допомогою своїх атрибутів (наприклад, назва, межі та населення країни). Сутності мають явні зв'язки з іншими сутностями. Сутності групуються в типи сутностей, де сутності одного типу мають однакову структуру атрибутів та зв'язків. Структуру даних у базі даних можна візуально представити за допомогою діаграми ERA. На рисунку 2.4 показано діаграму ERA, що відображає структуру цих даних у прикладі схеми бази даних у рисунках 2.1 та 2.2. Типи сутностей представлені прямокутниками з відгалуженими атрибутами та сполучними ребрами, що показують зв'язки. Підхід ERA детально обговорюється, тому він не розглядається далі тут.

Об'єктно-орієнтований підхід

Для багатьох прикладних областей, включаючи ГІС, моделювання ERA виявилося занадто обмеженим і його замінює підхід ОО. Підхід ОО використовується як метод семантичного моделювання даних, так і як модель даних, що обробляються об'єктно-орієнтованим програмуванням та системами управління базами даних. З точки зору систем баз даних, модель ОО адаптує деякі конструкції об'єктно-орієнтованих мов програмування до систем баз даних. Фундаментальна ідея полягає в інкапсуляції, яка розміщує оболонку навколо ідентифікованої колекції даних та коду, який працює з ними для створення об'єкта. Стан об'єкта в будь-який момент часу визначається значенням елементів даних в його обгортці. Ці елементи даних називаються *змінними екземпляра,* а значення, що містяться в них, самі є об'єктами. Це важлива відмінність між об'єктами (у сенсі ОО) та сутностями (у сенсі ERA), які мають дворівневу структуру сутності та атрибута.

У заключному технічному звіті робочої групи з об'єктно-орієнтованих баз даних Американського національного інституту стандартів (ANSI 1991) об'єкт описується як щось, «що відіграє певну роль стосовно запиту на операцію. Запит викликає операцію, яка визначає певну послугу, що має бути виконана». Код, пов'язаний з колекцією даних в об'єкті, надає набір методів, які можна виконати над ним [14]. Окрім виконання методів над власними даними, об'єкт може як частину одного зі своїх методів надсилати повідомлення іншому об'єкту, що

призводить до для виконання методу у відповідь.

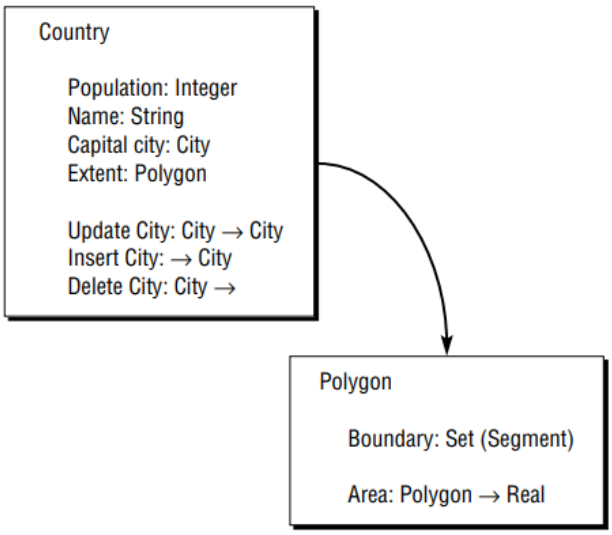

Рисунок 2.5 - Частина описів класів для Країни та Полігону.

Це високоактивне середовище є ще однією особливістю, яка відрізняє ООП від ERA, яке по суті є колекцією пасивних даних. Об'єкт має як стан, що є значеннями змінних екземпляра всередині нього, так і поведінку, що є потенціалом для дії на об'єкти (включаючи самого себе). Об'єкти з однаковими типами змінних екземпляра та методів називаються об'єктами одного класу об'єктів. На рисунку 2.5 показано деякі змінні екземпляра та методи, пов'язані з класами Country та Polygon, а також спосіб, у який клас Polygon посилається на змінну екземпляра класом Country. На рисунку 2.6 схематично показано об'єкт, що інкапсулює стан та методи, який отримує повідомлення від іншого об'єкта.

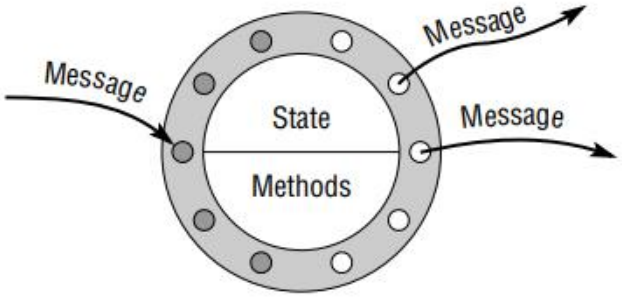

Рисунок 2.6 - Стан, методи та повідомлення об'єкта.

На рисунку 2.7 показано взаємодію кількох об'єктів у відповідь на повідомлення до одного з них.

Завдяки інкапсуляції внутрішня робота об'єкта прозора для користувачів та інших об'єктів, які можуть взаємодіяти з ним лише через набір попередньо визначених типів повідомлень, які об'єкт може зрозуміти та обробити. Візьмемо приклад з реального світу, мене зазвичай не хвилює стан мого автомобіля під капотом (внутрішній стан об'єкта класу Car) за умови, що коли я натискаю на педаль акселератора (надсилаю повідомлення), швидкість автомобіля збільшується (зміна внутрішнього стану призводить до зміни спостережуваних властивостей об'єкта) [15]. З точки зору, що знаходиться зовні об'єкта, зазвичай цікаві лише його спостережувані властивості.

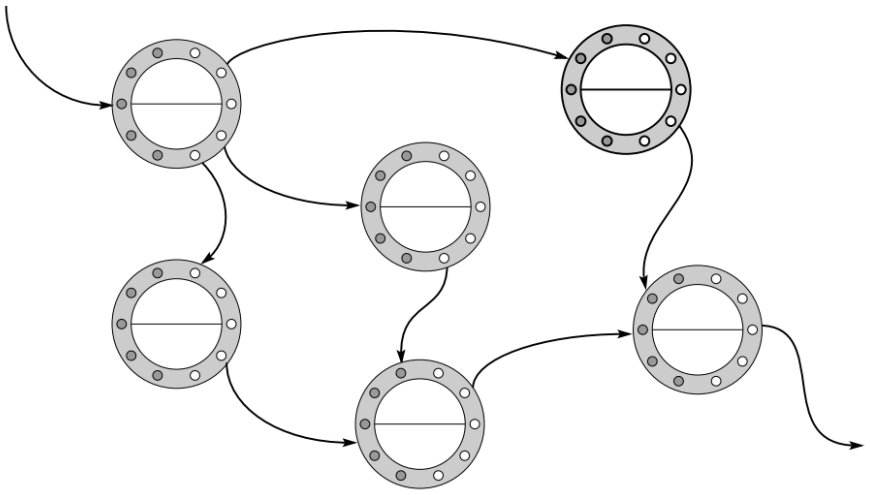

Рисунок 2.7 - Обмін повідомленнями між об’єктами.

Успадкування та композиція об'єктів

Успадкування є важливою конструкцією системного та семантичного моделювання, яка передбачає створення нового класу об'єктів шляхом модифікації існуючого класу. Успадкування в об'єктно-орієнтованому середовищі дозволяє успадкування методів. Таким чином, Triangle та Rectangle є підкласами Polygon [16]. Підкласи успадковують усі змінні екземпляра та методи від суперкласу, а також додають свої власні. У цьому прикладі Triangle та Rectangle можуть мати спеціалізовані методи, наприклад, алгоритм, що реалізує операцію

Area, може відрізнятися для Rectangle та Triangle, і кожен з них буде відрізнятися від алгоритму Area для Polygon. Це явище, коли оператор з однаковою назвою має різні реалізації в різних класах, називається *операторним поліморфізмом.* Приклад ієрархії успадкування класів просторових об'єктів наведено нижче (див. Рисунок 2.8).

Композиція об'єктів дозволяє моделювати об'єкти зі складними внутрішніми структурами. Існує кілька способів, за допомогою яких колекція об'єктів може бути об'єднана в новий об'єкт. *Агрегація* об'єднує колекцію класів об'єктів в агрегований клас. Наприклад, клас об'єктів "Власність" може бути агрегацією класів об'єктів "Земельна ділянка" та "Житло". «Агрегований об'єкт - це семантично розширений об'єкт, який розглядається як одиниця в багатьох операціях, хоча фізично складається з кількох менших об'єктів». *Асоціація* групує всі об'єкти з одного класу в асоційований клас. Наприклад, клас об'єктів "Райони" може бути асоціацією окремих класів об'єктів району.

Як ілюстрація деяких із цих конструкцій, на рисунку 2.6 показано клас об'єктів Country (як абстрактний клас об'єктів, представлений у вигляді трикутника) з трьома його змінними екземпляра Name (Назва), Population (Популяція) та Area (Площа). Змінні Name (Назва) та Population (Популяція) посилаються на друкований клас об'єктів Character String (представлений у вигляді овалу), а Area (Площа) посилається на абстрактний клас Polygon (Полигон). Клас Polygon має змінну екземпляра Boundary (Межа), яка посилається на асоціацію класу Segment (клас асоціації, показаний на рисунку у вигляді зірки та кола). Кожен сегмент має початкову та кінцеву точки, а кожна точка (Point) має позицію (Position), яка є агрегацією (показаною у вигляді хреста та кола) друкованих класів X-координати та Y-координати [17].

Об'єктно-орієнтовані системи управління базами даних

Забезпечення стійкості об'єктно-орієнтованого об'єкта

Об'єктно-орієнтовані мови програмування (OOPL), такі як C++ та Smalltalk, надають можливості для підтримки описаного вище підходу до об'єктно-орієнтованого програмування, включаючи створення, підтримку та

видалення об'єктів, класів об'єктів та ієрархій успадкування. Об'єктно-орієнтовані системи керування базами даних (OODBMS) доповнюють ці можливості функціональністю баз даних, зокрема здатністю підтримувати:

1 постійні об'єкти, класи об'єктів та ієрархії успадкування;

2 непроцедурні мови запитів для визначення класів об'єктів, маніпулювання об'єктами та їх пошуку;

3 ефективна обробка запитів, включаючи оптимізацію запитів та методи доступу;

4 відповідна обробка транзакцій (властивості ACID), підтримка паралельності, відновлення, цілісність та безпека.

Розробник ООПБМ-системи має два варіанти: розширити реляційну систему для обробки ООП або побудувати систему баз даних навколо мова програмування. Обидва варіанти були випробувані, про розширення об'єктів для реляційної технології досліджує перший. Щодо другого, об'єктно-орієнтовані функції вже підтримуватимуться OOPL, тому існує потреба додати збереження, обробку запитів та обробку транзакцій. Підхід до збереження полягає в додаванні нового класу Persistent Object та дозволі всім класам бази даних успадковуватися від цього класу [18]. Клас Persistent Object включатиме методи для:

1. створити новий постійний об'єкт;
2. видалити постійний об'єкт;
3. отримати стан постійного об'єкта;
4. забезпечити контроль паралельності;
5. змінити постійний об'єкт.

Ключовою перевагою ООСБД є підтримка, яку вона забезпечує для єдиного середовища програмування та роботи з базами даних. Однак більшість сучасних ООСБД по-різному обробляють постійні та непостійні дані. Фундаментальна відмінність між РБД та ООСБД полягає у виклику за значенням та виклику за посиланням [19].

У РБД зв'язки встановлюються шляхом зіставлення значень. У нашому прикладі, щоб отримати дані про населення столиці Німеччини, об'єднання між

країною та містом виконується за допомогою значення поля «Столиця» – Берлін. В ООБД (див. рисунок 2.7) з'єднання здійснюється за допомогою навігації за допомогою ідентифікаторів об'єктів (OID). Змінна екземпляра «Столиця» об'єкта «Країна» вказує на відповідний об'єкт «Місто».

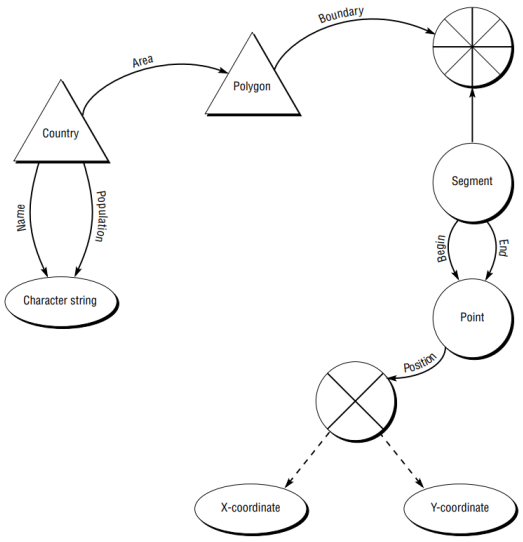

Рисунок 2.8 - Складні об'єкти

Стандартизація ООП-систем

Об'єктно-орієнтований підхід є складнішим за реляційну модель і ще не сформувався в набір універсально узгоджених конструкцій; навіть базові конструкції, такі як успадкування, отримали кілька різних інтерпретацій. Тим не менш, було проведено значну роботу для досягнення деяких спільних визначень.

Група управління об'єктами (OMG) – це консорціум постачальників апаратного та програмного забезпечення, заснований у 1990 році з метою сприяння стандартам для сумісності програм в рамках об'єктно-орієнтованого підходу. З цією метою вона визначила об'єктну модель OMG, багато концепцій якої обговорювалися вище. Важливим компонентом роботи OMG є Стандарт Common Object Request Broker Architecture (CORBA), який визначає мову визначення інтерфейсу для розподіленого доступу до об'єктів. Object Database Management Group (ODMG) – це консорціум постачальників ООСБД, заснований у 1990 році з метою розробки загальноузгодженого інтерфейсу ООСБД. ODMG

визначила мови визначення, маніпулювання та запитів об'єктів, що відповідають мовам визначення та маніпулювання даними в реляційних системах [20].

Розширення об'єктів для реляційної технології

Функціональність бази даних навколо мови об'єктно-орієнтованого програмування. Удосконалення включають складні, можливо, визначені користувачем типи даних, успадкування, агрегацію та ідентичність об'єктів. Рання робота в Каліфорнійському університеті в Берклі щодо включення нових типів даних у реляційні системи баз даних призвела до появи СУБД POSTGRES [21]. Паралельні розробки в дослідницьких лабораторіях IBM, призвели до проекту STARBURST. Ці розробки призвели до появи сучасних власних об'єктно-реляційних систем, а також до додавання об'єктних функцій до нових випусків широко використовуваних власних реляційних систем. Що стосується мов запитів, які підтримують об'єктно-орієнтовані розширення реляційної моделі, SQL3 зараз розробляється як міжнародний стандарт. SQL3 сумісний з SQL-92 угору та додає підтримку об'єктів, включаючи множинне успадкування та оператори. Метою об'єктно-реляційних систем є забезпечення широкого спектру об'єктно-орієнтованих функцій, які виявилися настільки корисними для семантичного моделювання даних та систем програмування, водночас забезпечуючи ефективну продуктивність, пов'язану з реляційною моделлю.

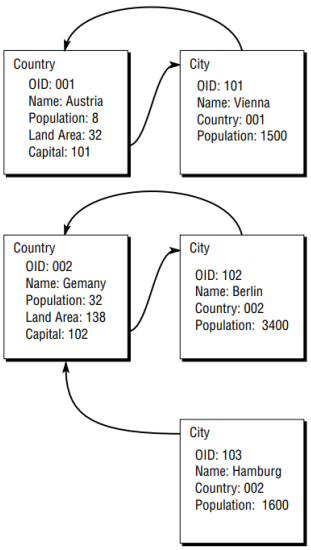

Рисунок 2.9 - Навігація за допомогою ідентифікаторів об'єктів.

Об'єкти структуровані реляційною моделлю як кортежі атомарних значень, таких як цілі числа, числа з плаваючою комою, логічні значення або рядки символів. Це забезпечує лише обмежені засоби для визначення складних типів даних [22]. Об'єктно-реляційні системи дозволяють використовувати неатомарні типи.

Поширеним розширенням реляційної моделі для забезпечення складних типів даних є дозвіл на *вкладені відношення.* У вкладеному відношенні значення атрибутів не обов'язково повинні бути атомарними, але самі можуть бути відношеннями.

Реляційні системи баз даних забезпечують хешування та індекси B-дерев для доступу до стандартних, наданих системою типів даних. Важливим розширенням, яке дозволяє об'єктно-реляційна система, є надання більш відповідних індексів для типів, визначених користувачем. Об'єктно-реляційні системи забезпечують визначення діапазону індексів, що відповідають гетерогенній колекції класів об'єктів [23].

2.3 Концептуальне та логічне проєктування реляційних баз даних

Моделі даних забезпечують структуроване та формальне описання даних і зв’язків між ними, необхідних для інформаційної системи. Коли дані потрібні для IT-проєктів, наприклад, інформація про співробітників, підрозділи та проєкти (як на рис. 2.10), можна визначити необхідні категорії даних та їхні взаємозв’язки. Визначення цих категорій даних, які називаються наборами сутностей (entity sets), та визначення наборів зв’язків (relationship sets) на цьому етапі виконується без урахування типу системи керування базами даних (SQL чи NoSQL), яка згодом використовуватиметься для введення, зберігання та підтримки даних. Це робиться для того, щоб дані та зв’язки між ними залишалися стабільними з точки зору користувачів протягом усього життєвого циклу розробки та розширення інформаційних систем [24].

Для створення бази даних, що описує певну ділянку реального світу,

необхідно пройти три етапи: аналіз даних, проєктування концептуальної моделі даних (у даному випадку - моделі «сутність-зв’язок») та перетворення її в реляційну або нереляційну схему бази даних.

Метою аналізу даних (див. крок 1 на рис. 2.10) є спільне з користувачем виявлення даних, необхідних для інформаційної системи, а також визначення їхніх взаємозв’язків, включаючи кількісну структуру. Це критично важливо для раннього визначення меж системи. Аналіз вимог виконується ітеративно на основі інтерв’ю, аналізу потреб, анкет, збирання форм тощо. Він містить щонайменше вербальний опис завдання з чітко сформульованими цілями та перелік релевантних відомостей (див. приклад на рис. 2.10). Письмовий опис зв’язків між даними може бути доповнено графічними ілюстраціями або узагальнювальним прикладом.

Важливо, щоб аналіз даних викладав факти, необхідні для подальшої розробки бази даних, мовою користувачів.

На кроці 2 (рис. 2.10) відбувається побудова моделі «сутність-зв’язок» (entity-relationship model), яка включає як необхідні набори сутностей, так і відповідні набори зв’язків. У нашій моделі набори сутностей зображаються прямокутниками, а набори зв’язків - ромбами.

На основі аналізу даних з кроку 1 основними наборами сутностей є DEPARTMENT (ПІДРОЗДІЛ), EMPLOYEE (СПІВРОБІТНИК) та PROJECT (ПРОЄКТ). Для фіксації того, в яких підрозділах працюють співробітники та в яких проєктах вони беруть участь, створюються набори зв’язків SUBORDINATE (ПІДЛЕГЛИЙ) та INVOLVED (ЗАЛУЧЕНИЙ), які графічно з’єднуються з відповідними наборами сутностей.

Таким чином, модель «сутність-зв’язок» дозволяє структурувати та графічно представити факти, зібрані під час аналізу даних [25]. Однак слід зазначити, що виявлення наборів сутностей і зв’язків, а також визначення відповідних атрибутів не завжди є простим і однозначним процесом. Цей етап проєктування вимагає значного досвіду та практики від архітектора даних.

Далі модель «сутність-зв’язок» перетворюється в реляційну схему бази

даних (рис. 2.10, 3a) або в граф-орієнтовану схему бази даних (рис. 2.10, 3b).

Схема бази даних - це формальний опис об’єктів бази даних за допомогою таблиць або вузлів і ребер. Оскільки реляційні системи керування базами даних дозволяють використовувати лише таблиці як об’єкти, як набори сутностей, так і набори зв’язків мають бути виражені у вигляді таблиць. Тому на кроці 3a (рис. 2.10) для наборів сутностей DEPARTMENT, EMPLOYEE та PROJECT створюється по одній таблиці сутностей.

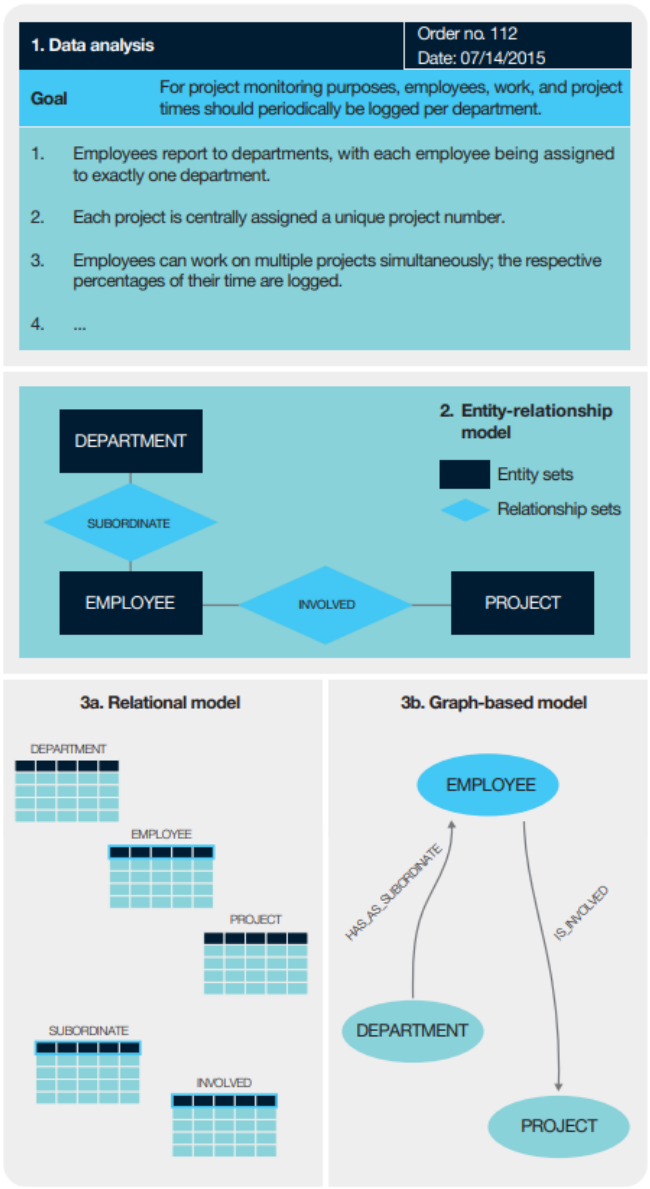

Рисунок 2.10 - Три етапи, необхідні для моделювання даних

Для представлення зв’язків у вигляді таблиць також потрібно визначити

окремі таблиці для кожного набору зв’язків. У нашому прикладі це таблиці SUBORDINATE та INVOLVED [26]. Такі таблиці наборів зв’язків завжди містять ключі задіяних наборів сутностей як зовнішні ключі (foreign keys), а також можуть містити додаткові атрибути самого зв’язку.

На кроці 3b (рис. 2.10) показано еквівалентну графову базу даних. Кожен набір сутностей відповідає вузлу в графі, тому маємо вузли DEPARTMENT, EMPLOYEE та PROJECT. Набори зв’язків SUBORDINATE та INVOLVED з моделі «сутність-зв’язок» перетворюються на ребра графа. Набір зв’язків SUBORDINATE стає спрямованим ребром від вузла DEPARTMENT до вузла EMPLOYEE і називається HAS_AS_SUBORDINATE. Аналогічно, спрямоване ребро з назвою IS_INVOLVED проводиться від вузла EMPLOYEE до вузла PROJECT.

Це лише загальний огляд процесу аналізу даних, розробки моделі «сутність-зв’язок» та визначення реляційної або графової схеми бази даних. Головний висновок полягає в тому, що проєктування бази даних має базуватися на моделі «сутність-зв’язок». Це дозволяє збирати та обговорювати фактори моделювання даних разом з користувачами незалежно від конкретної системи баз даних. Лише на наступному етапі проєктування визначається та деталізується найбільш підходяща схема бази даних. Для реляційних і граф-орієнтованих баз даних існують чітко визначені правила відображення.

Модель «сутність-зв’язок»

Сутність - це конкретний об’єкт реального світу або нашої уяви, який є відмінним від усіх інших. Це може бути особа, предмет, абстрактне поняття або подія. Сутності одного типу об’єднуються в набори сутностей і додатково характеризуються атрибутами. Атрибути - це категорії властивостей сутності та/або набору сутностей, наприклад, розмір, назва, вага тощо.

Для кожного набору сутностей визначається ключ ідентифікації — один атрибут або певна комбінація атрибутів, яка є унікальною. Крім унікальності, він також повинен відповідати критерію мінімальної комбінації атрибутів для ключів ідентифікації [27].

На рис. 2.11 окремий співробітник представлений як сутність зі своїми конкретними атрибутами. Якщо в рамках внутрішнього моніторингу проєктів потрібно отримати список усіх співробітників з їхніми прізвищами та адресними даними, створюється набір сутностей EMPLOYEE (СПІВРОБІТНИК). Штучний номер співробітника разом з атрибутами «Прізвище», «Вулиця» та «Місто» дозволяє однозначно ідентифікувати окремих співробітників (сутностей) у складі штату (набору сутностей).

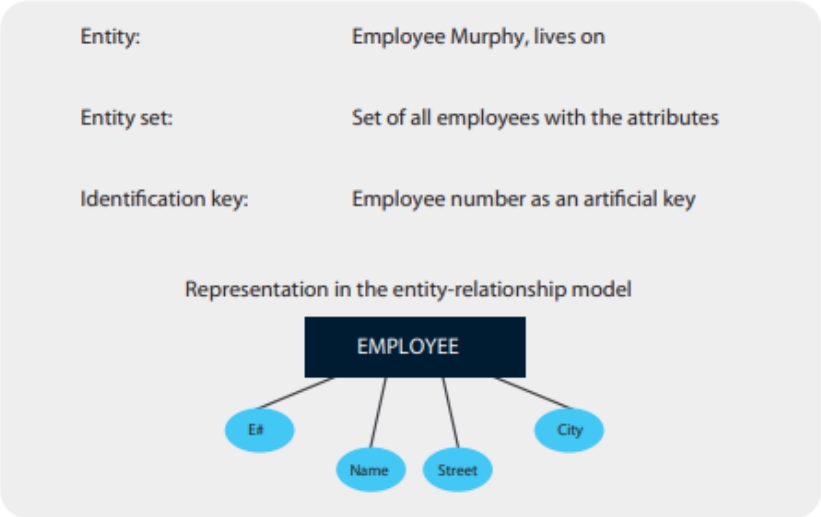

Рисунок 2.11 - Набір сутностей EMPLOYEE

Окрім самих наборів сутностей, важливими є також зв’язки між ними, які можуть утворювати власні набори. Подібно до наборів сутностей, набори зв’язків також можуть характеризуватися атрибутами.

Рисунок 2.12 ілюструє висловлювання «Співробітниця Мерфі виконує 70% своєї роботи в проєкті P17» як конкретний приклад зв’язку між співробітником і проєктом. Відповідний набір зв’язків INVOLVED (ЗАЛУЧЕНИЙ) призначений для обліку всіх випадків участі співробітників у проєктах. Він містить складений ключ, утворений з зовнішніх ключів «номер співробітника» та «номер проєкту». Ця комбінація атрибутів забезпечує унікальну ідентифікацію кожної участі співробітника в проєкті. Крім складеного ключа, набір зв’язків отримує власний атрибут «Percentage» (Відсоток), який вказує частку робочого часу, яку

співробітники виділяють на кожен проєкт, у якому вони беруть участь [28].

Загалом, зв'язки можна розглядати як асоціації в двох напрямках: Набір зв'язків INVOLVED з точки зору набору сутностей EMPLOYEE можна інтерпретувати як «один співробітник може брати участь у кількох проєктах»; з точки зору набору сутностей PROJECT - «один проєкт може виконуватися кількома співробітниками».

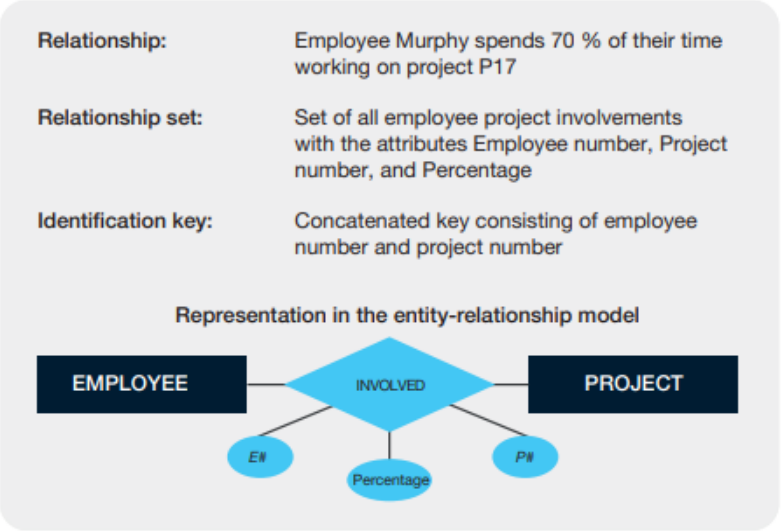

Рисунок 2.12 - Зв'язок INVOLVED між співробітниками та проєктами

Асоціація набору сутностей ES_1 з іншим набором сутностей ES_2 - це значення зв'язку в цьому напрямку. Наприклад, зв'язок DEPARTMENT_HEAD (КЕРІВНИК ПІДРОЗДІЛУ) на рис. 2.4 має дві асоціації: з одного боку, кожен підрозділ має одного співробітника в ролі керівника; з іншого боку, деякі співробітники можуть виконувати роль керівника певного підрозділу.

Кожна асоціація від набору сутностей ES_1 до набору сутностей ES_2 може бути охарактеризована типом асоціації. Тип асоціації від ES_1 до ES_2 показує, скільки сутностей пов'язаного набору ES_2 може бути призначено одній конкретній сутності з ES_1 [29].

Основні типи асоціацій: унікальний, умовний, множинний та множинно-умовний.

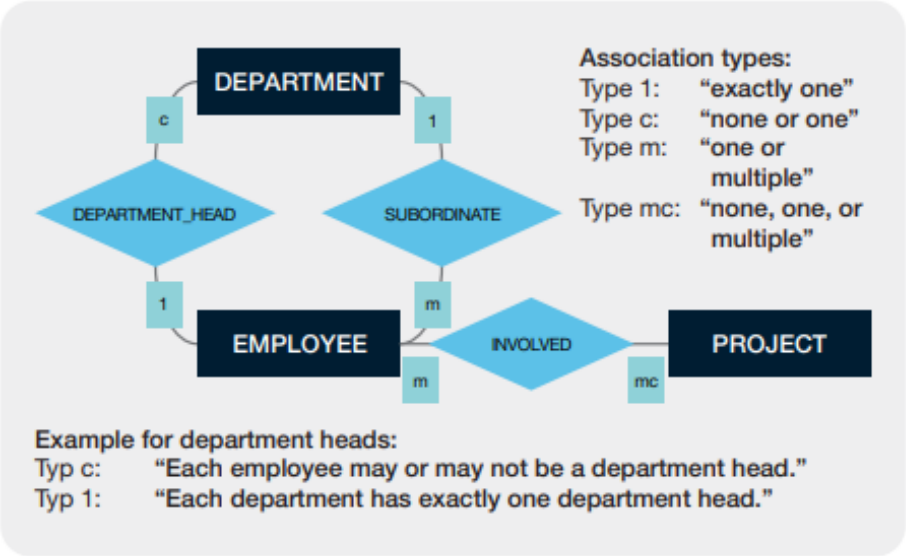

Рисунок 2.13 - Модель «сутність-зв'язок» з типами асоціацій

u Унікальна асоціація (тип 1) У унікальній (тип 1) асоціації кожній сутності з набору ES_1 відповідає рівно одна сутність з набору ES_2. Наприклад, аналіз даних показав, що кожен співробітник підпорядкований рівно одному підрозділу (матричне управління не допускається) [30]. Тому зв'язок SUBORDINATE від співробітників до підрозділів на рис. 2.13 є унікальною асоціацією типу 1.

u Умовна асоціація (тип c) Асоціація типу c означає, що кожній сутності з набору ES_1 відповідає нуль або одна (тобто максимум одна) сутність з набору ES_2. Зв'язок є необов'язковим, тому тип асоціації — умовний. Прикладом умовної асоціації є зв'язок DEPARTMENT_HEAD (рис. 2.13), оскільки не кожен співробітник може бути керівником підрозділу.

u Множинна асоціація (тип m) У множинній (тип m) асоціації кожній сутності з набору ES_1 відповідає одна або кілька сутностей з набору ES_2. Цей тип асоціації часто називають комплексним, оскільки одна сутність з ES_1 може бути пов'язана з довільною кількістю сутностей з ES_2. Приклад множинної асоціації на рис. 2.13 - зв'язок INVOLVED від проєктів до співробітників: кожен проєкт може залучати кількох співробітників, але повинен мати щонайменше одного.

u Множинно-умовна асоціація (тип mc) Кожній сутності з набору ES_1 відповідає нуль, одна або кілька сутностей з набору ES_2. Множинно-умовні асоціації відрізняються від множинних тим, що не кожна сутність з ES_1

обов’язково повинна мати зв’язок із сутностями ES_2. За аналогією їх також називають умовно-комплексними. Приклад - той самий зв’язок INVOLVED на рис. 2.13, але з точки зору співробітників: не кожен співробітник обов’язково бере участь у проєктах, але деякі співробітники беруть участь у кількох проєктах.

Типи асоціацій надають інформацію про кардинальність зв’язку. Як ми бачили, кожен зв’язок має два типи асоціацій. Тому кардинальність зв’язку між наборами сутностей ES_1 та ES_2 - це пара типів асоціацій у форматі:

Кардинальність = (тип асоціації від ES_1 до ES_2, тип асоціації від ES_2 до ES_1)

Наприклад, пара (mc, m) між EMPLOYEE та PROJECT означає, що зв’язок INVOLVED є (множинно-умовним, множинним).

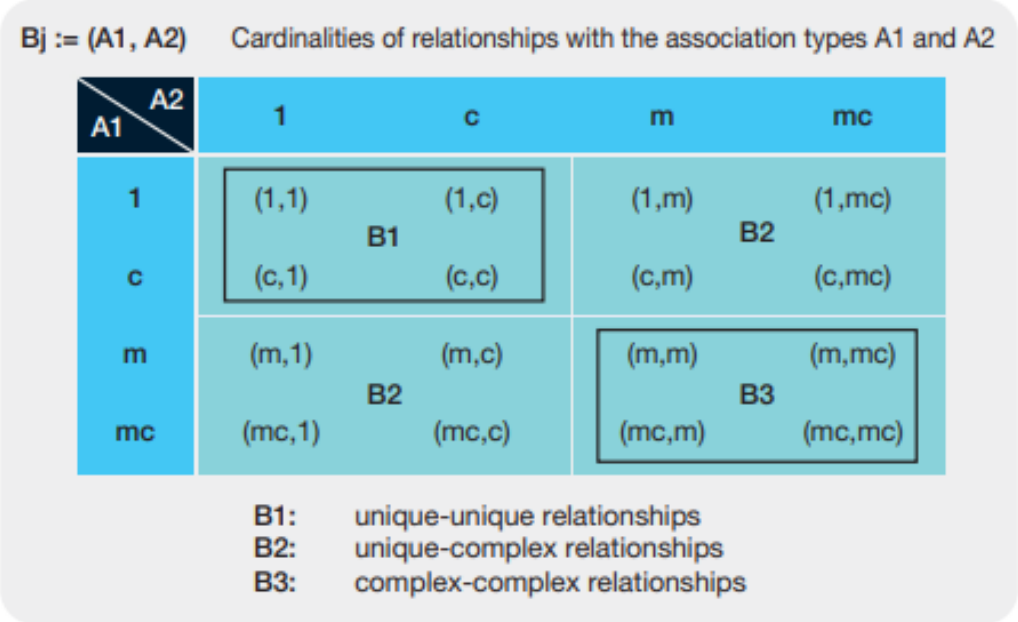

Рисунок 2.14 - Огляд можливих кардинальностей відносин

Рисунок 2.14 показує всі 16 можливих комбінацій типів асоціацій. Перший квадрант містить чотири варіанти унікально-унікальних зв’язків (випадок B1). Унікально-комплексні зв’язки (ієрархічні, випадок B2) мають вісім комбінацій. Комплексно-комплексні або мережеві зв’язки (випадок B3) включають чотири варіанти: (m,m), (m,mc), (mc,m) та (mc,mc) [31].

Замість типів асоціацій можна задавати мінімальні та максимальні пороги,

якщо це зручніше. Наприклад, замість множинної асоціації від проєктів до співробітників можна вказати діапазон (МІН, МАКС) := (3,8). Нижній поріг означає, що в проєкті має бути задіяно щонайменше три співробітники, а верхній обмежує кількість учасників вісьмома.

Реалізація в реляційній моделі

Вивчення реляційної моделі дало початок новій теорії баз даних, яка точно описує формальні аспекти. Однією з головних галузей цієї теорії є нормальні форми, які використовуються для виявлення та аналізу залежностей всередині таблиць з метою уникнення надлишкової інформації та пов'язаних з нею аномалій.

Про надлишковість атрибутів Атрибут у таблиці вважається надлишковим, якщо окремі його значення можна вилучити без втрати інформації. Наприклад, таблиця DEPARTMENT_EMPLOYEE містить номер співробітника, прізвище, вулицю та місто для кожного співробітника, а також номер підрозділу та назву підрозділу. Для всіх співробітників підрозділу A6 у таблиці на рис. 2.9 повторюється назва «Finances». Якщо припустити, що кожен підрозділ складається з кількох співробітників, подібні повтори відбуватимуться для всіх підрозділів [32]. Можна сказати, що атрибут DepartmentName є надлишковим, оскільки одне й те саме значення повторюється в таблиці багато разів.

Краще зберігати назву, що відповідає кожному номеру підрозділу, в окремій таблиці для подальшого використання, а не дублювати її для кожного співробітника.

Таблиці з надлишковою інформацією можуть призводити до аномалій бази даних, які бувають трьох видів:

- Аномалія вставки (Insertion Anomaly): Якщо з організаційних причин потрібно створити новий підрозділ A9 з назвою «Marketing» у таблиці DEPARTMENT_EMPLOYEE (рис. 2.15), але до нього ще не призначено жодного співробітника, додати такий запис неможливо. Це аномалія вставки — не можна додати новий рядок без унікального номера співробітника.

DEPARTMENT_EMPLOYEE

E#	Name	Street	City	D#	DepartmentName
E19	Stewart	E Main Street	Stow	D6	Accounting
E1	Murphy	Morris Road	Kent	D3	IT
E7	Howard	Lorain Avenue	Cleveland	D5	HR
E4	Bell	S Water Street	Kent	D6	Accounting

Рисунок 2.15 - Таблиця з надлишковістю та схильністю до аномалій

- Аномалія видалення (Deletion Anomaly): Виникає, коли видалення деяких даних призводить до ненавмисної втрати інших даних. Наприклад, якщо видалити всіх співробітників з таблиці DEPARTMENT_EMPLOYEE, ми також втратимо номери та назви підрозділів.
- Аномалія оновлення (Update/Modification Anomaly): Якщо назву підрозділу А3 змінити з «IT» на «Data Processing», доведеться редагувати запис для кожного співробітника цього підрозділу окремо. Тобто через зміну лише одного факту таблицю потрібно коригувати в багатьох місцях. Це і є аномалія оновлення [33].

Наступні параграфи розглядають нормальні форми, які допомагають уникнути надлишковості та аномалій. Рисунок 2.16 дає огляд різних нормальних форм та їх визначень.

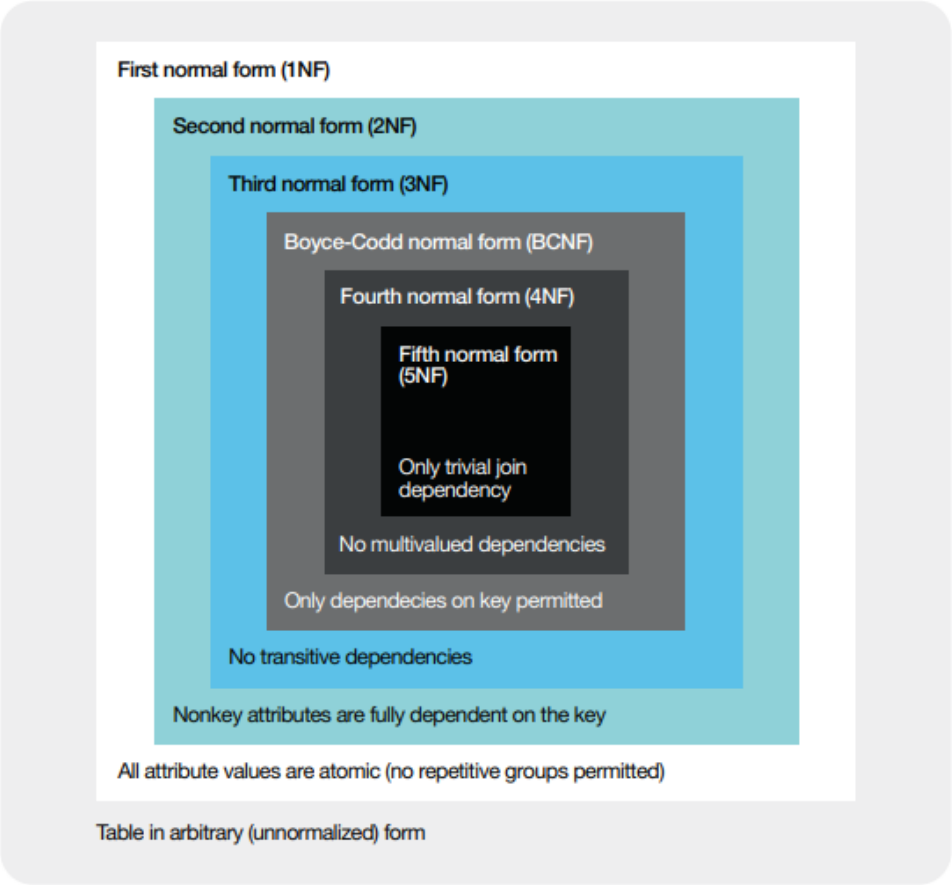

Рисунок 2.16 - Огляд нормальних форм та їх визначень

Нормальні форми поступово обмежують допустиму структуру таблиць. Наприклад, таблиця або вся схема бази даних у третій нормальній формі повинна задовольняти всі вимоги першої та другої нормальних форм, а також не повинна містити транзитивних залежностей між неключовими атрибутами.

Важливо зазначити, що не всі нормальні форми однаково практичні. Зазвичай використовують лише перші три нормальні форми, оскільки багатозначні та з'єднувальні залежності рідко зустрічаються на практиці. Тому ми лише коротко розглянемо четверту та п'яту нормальні форми [34].

Розуміння нормальних форм допомагає правильно застосовувати правила відображення моделі «сутність-зв'язок» у реляційну модель. Насправді, при правильно побудованій моделі «сутність-зв'язок» та послідовному застосуванні правил відображення нормальні форми зазвичай виконуються автоматично. Таким чином, після створення ER-моделі та її відображення в реляційну схему часто можна обійтися без детальної перевірки нормальних форм на кожному

кроці.

Функціональні залежності

Перша нормальна форма (1НФ) Таблиця перебуває в першій нормальній формі, якщо домени всіх її атрибутів є атомарними. Це означає, що кожний атрибут отримує значення зі структурованого (неструктурованого набору) домену, і в ньому не повинно бути множин, списків чи повторюваних груп.

Таблиця PROJECT_PARTICIPANT на рис. 2.17 ще не нормалізована, оскільки кожен рядок співробітника містить кілька номерів проєктів, у яких він бере участь. Для переведення таблиці в 1НФ потрібно створити окремий рядок для кожної участі в проєкті. При цьому ключ таблиці розширюється - для унікальної ідентифікації кожного рядка тепер потрібні як номер співробітника, так і номер проекту [35].

Парадоксально, але після переходу до першої нормальної форми в таблиці залишається багато надлишковості - у прикладі на рис. 2.17 прізвища та адреси співробітників повторюються для кожної участі в проєкті. Тут вступає в дію друга нормальна форма.

Друга нормальна форма (2НФ) Таблиця перебуває в другій нормальній формі, якщо вона задовольняє вимоги першої нормальної форми, і кожен неключовий атрибут повністю функціонально залежить від усього ключа.

Атрибут B функціонально залежить від атрибута A (позначається A → B), якщо кожному значенню A відповідає рівно одне значення B. Для складених ключів ця залежність повинна бути повною функціональною залежністю.

Таблиця PROJECT_PARTICIPANT у 1НФ (рис. 2.17) має складений ключ (E#, P#). Атрибути Name та City функціонально залежать лише від частини ключа (E#), а не від усього ключа (E#, P#). Це порушує вимогу 2НФ.

Для усунення цієї проблеми таблицю потрібно розділити. Атрибути, що залежать лише від частини ключа, переносяться в окрему таблицю разом з цією частиною ключа. У результаті отримуємо таблиці EMPLOYEE та PROJECT_INVOLVEMENT, які задовольняють як 1НФ, так і 2НФ [36].

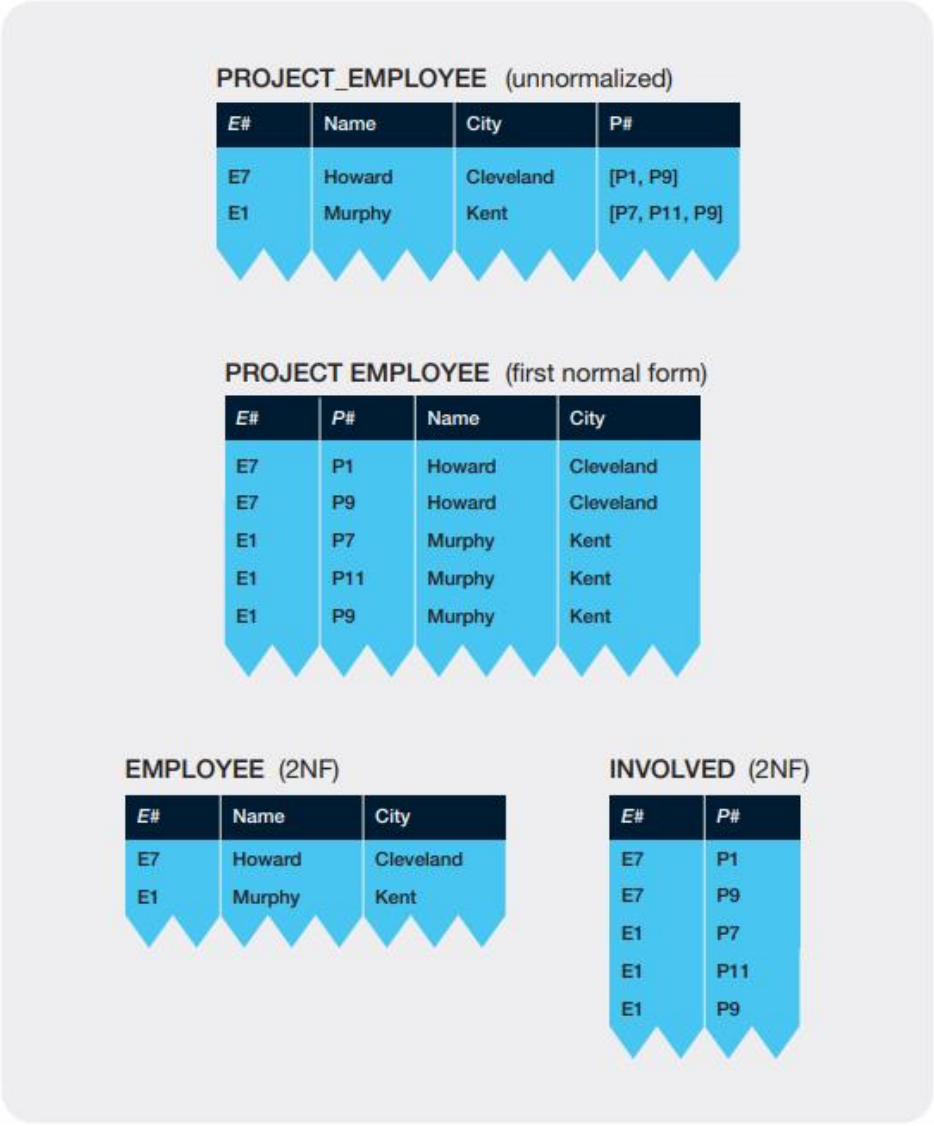

Рисунок 2.17 - Таблиці в першій та другій нормальних формах

Транзитивні залежності

Повернемося до таблиці DEPARTMENT_EMPLOYEE (рис. 2.18). Вона вже перебуває в 1НФ і 2НФ, але атрибут DepartmentName все ще є надлишковим. Цю проблему вирішує третя нормальна форма.

Третя нормальна форма (3НФ) Таблиця перебуває в третій нормальній формі, якщо вона задовольняє вимоги другої нормальної форми, і жоден неключовий атрибут не перебуває в транзитивній залежності від ключового атрибута.

Транзитивна залежність виникає, коли атрибут C функціонально залежить від A не напряму, а через інший атрибут B (A → B → C), причому A не залежить

від B.

У таблиці DEPARTMENT_EMPLOYEE атрибут DepartmentName транзитивно залежить від E# через D#. Для усунення цієї залежності надлишковий атрибут DepartmentName виноситься в окрему таблицю DEPARTMENT, а в таблиці EMPLOYEE залишається лише номер підрозділу (D#) як зовнішній ключ зі роллю SUBORDINATE [37].

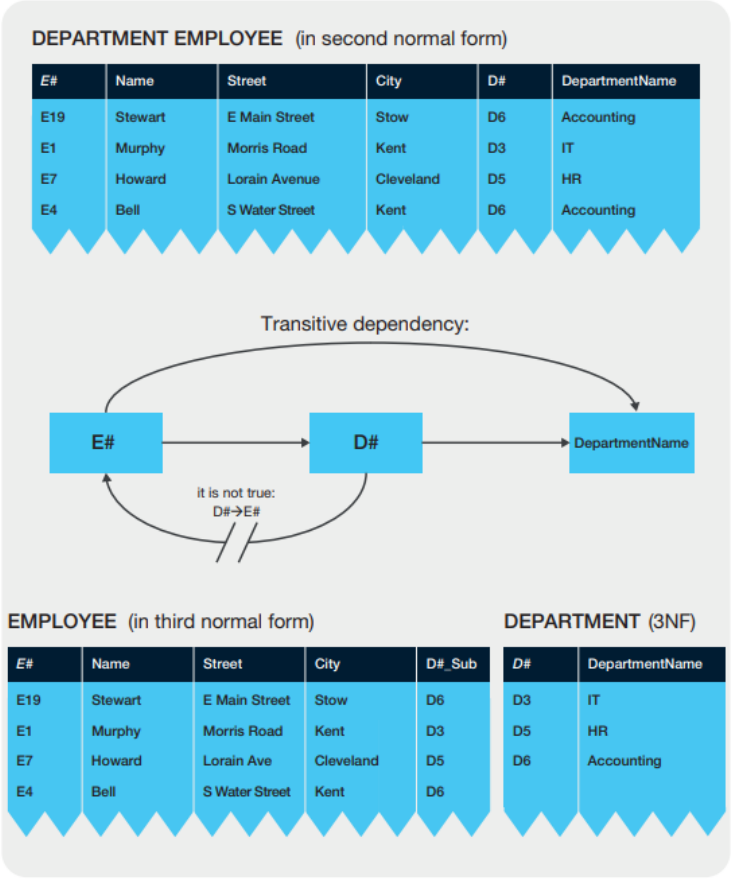

Рисунок 2.18 - Транзитивна залежність та третя нормальна форма

Залежності та нормальні форми (продовження)

Багатозначні залежності

Друга та третя нормальні форми дозволяють усунути надлишковість серед неключових атрибутів. Однак перевірка на надлишковість не повинна на цьому зупинятись, оскільки надлишковість може також стосуватися складених ключів. Для таких випадків була запропонована модифікація третьої нормальної форми, яка отримала назву нормальна форма Бойса-Кодда (Boyce-Codd Normal Form,

BCNF). Вона застосовується, коли в одній таблиці існує кілька перекриваючих кандидатних ключів. Такі таблиці, навіть перебуваючи в 3НФ, можуть порушувати вимоги BCNF. У цьому випадку таблицю необхідно розділити на основі кандидатних ключів. Детальнішу інформацію можна знайти в рекомендованій літературі.

Інша нормальна форма виникає при вивченні багатозначних залежностей (multivalued dependencies) між окремими ключовими атрибутами. Хоча вони відіграють незначну роль на практиці, рис. 2.19 наводить простий приклад для ілюстрації проблеми.

Початкова таблиця METHOD не нормалізована, оскільки поруч із кожним методом може бути вказано кілька авторів і кілька термінів. Наприклад, метод структограм містить авторів Nassi і Shneiderman, а також кілька значень для атрибута Term (sequence, iteration, branching) [38].

Після перетворення не нормалізованої таблиці в першу нормальну форму (шляхом розбиття множин) ми отримуємо таблицю, яка складається лише з ключових атрибутів. Вона перебуває не тільки в 1НФ, але й у 2НФ та 3НФ. Однак, попри це, у таблиці все ще присутня надлишкова інформація. Для кожного автора методу структограм повторюються три терміни, а для кожного терміну - обидва автори. Маємо справу з парними багатозначними залежностями, які потрібно усунути.

Визначення багатозначної залежності: Атрибут C є багатозначно залежним від атрибута A (позначається A →→ C), якщо для будь-якої комбінації конкретного значення A з довільним значенням B отримується однаковий набір значень C.

У прикладі на рис. 2.19 атрибут Term багатозначно залежить від Method (Method →→ Term), а атрибут Author також багатозначно залежить від Method (Method →→ Author).

Четверта нормальна форма (4НФ) Таблиця перебуває в четвертій нормальній формі, якщо вона не містить двох і більше істинних та незалежних багатозначних залежностей.

Таблицю METHOD потрібно розділити на дві підтаблиці: METHODOLOGIST (метод - автор) та METHODOLOGY (метод - термін). Обидві таблиці вільні від надлишковості та відповідають 4НФ.

З'єднувальні залежності (Join Dependencies)

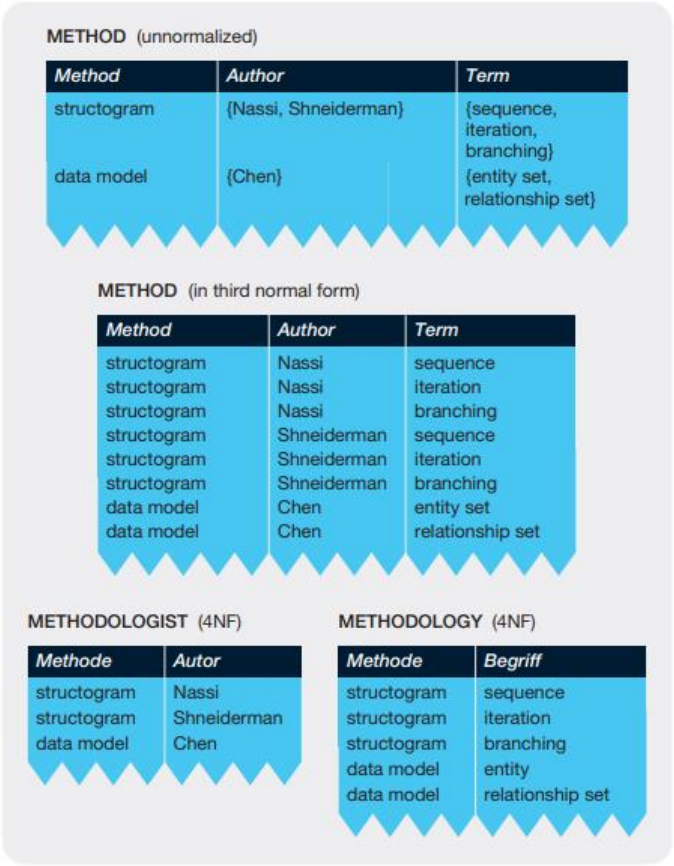

Рисунок 2.19 - Таблиця з багатозначними залежностями

При розділенні таблиць на підтаблиці можлива втрата інформації. Щоб цього уникнути, існують критерії безвтратного розділення таблиць.

Рисунок 2.20 демонструє неправильне розділення таблиці PURCHASE на підтаблиці BUYER та WINE. Після з’єднання цих двох таблиць за спільним атрибутом Variety отримуємо неправильну таблицю, яка містить записи, яких не було в оригіналі (помилкові записи виділено жирним).

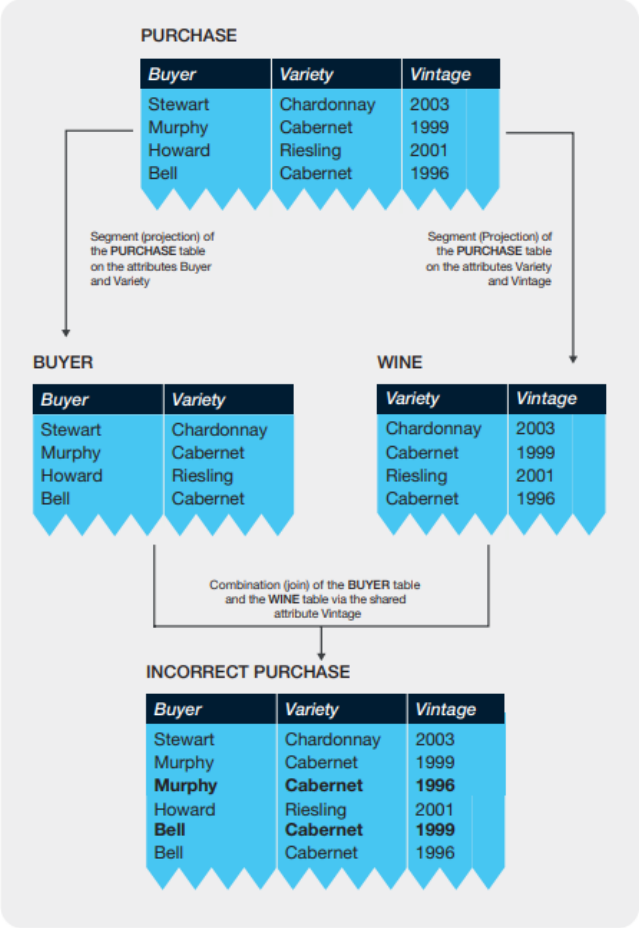

Рисунок 2.20 - Неправильне розділення таблиці PURCHASE

П'ята нормальна форма (5НФ) Таблиця перебуває в п’ятій нормальній формі (також називається проєкційно-з’єднувальною нормальною формою, PJNF), якщо її можна довільно розділити за допомогою оператора проєкції, а потім відновити в оригінальному вигляді за допомогою оператора з’єднання (join) без втрати або спотворення інформації [39]. Ця властивість називається безвтратною залежністю від з’єднання (lossless join dependency).

У прикладі на рис. 2.21 таблиця, яка перебуває в 4НФ, не задовольняє умову безвтратного з’єднання. Для приведення її до 5НФ її розділяють на три підтаблиці: BUYER, WINE та PREFERENCE. З’єднання цих трьох таблиць повністю відновлює оригінальну таблицю PURCHASE.

Це обговорення закономірно викликає питання: чи можна використовувати синтетичний підхід до проєктування бази даних замість правил нормалізації та розділення (аналітичний підхід)? Існують алгоритми, які

дозволяють об’єднувати підтаблиці в таблиці, що відповідають третій або вищій нормальній формі.

Рисунок 2.21 - Таблиці в п’ятій нормальній формі

Такі правила синтезу дають можливість проєктувати схеми баз даних на основі набору залежностей. Таким чином, проєктування може бути як зверху-вниз (аналітичне), так і знизу-вгору (синтетичне). На жаль, дуже мало CASE-засобів підтримують обидва методи, тому архітектори баз даних повинні вручну перевіряти правильність своїх проєктів.

Висновок по розділу

У другому розділі розглянуто методи та інструменти розробки програмного забезпечення з використанням реляційних баз даних. Досліджено механізми взаємодії програм із базами даних за допомогою мови SQL, а також підходи до інтеграції об’єктно-орієнтованих моделей із реляційними структурами даних. Виконано аналіз процесів концептуального та логічного проєктування баз даних, що дозволяє забезпечити цілісність, узгодженість і ефективність зберігання інформації. Отримані результати стали підґрунтям для реалізації програмного рішення та вибору оптимальної структури бази даних.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Трансляція та оптимізація реляційних запитів у програмних системах

Інтерфейси користувача реляційних систем баз даних є орієнтованими на множини, оскільки користувачам надаються цілі таблиці або представлення (views). Під час використання мови реляційних запитів та маніпулювання даними система баз даних повинна транслювати та оптимізувати відповідні команди. Важливо, щоб ні обчислення, ні оптимізація дерева запиту не вимагали дій з боку користувача.

Дерево запиту

Дерева запитів графічно візуалізують реляційні запити за допомогою еквівалентних виразів реляційної алгебри. Листками дерева запиту є таблиці, що використовуються в запиті; корінь та внутрішні вузли містять алгебраїчні оператори.

На рисунку 3.1 зображено дерево запиту з використанням SQL та раніше введених таблиць EMPLOYEE (СПІВРОБІТНИК) і DEPARTMENT (ВІДДІЛ). Запит повертає список міст, у яких проживають співробітники відділу IT:

SQL

```
SELECT City
FROM EMPLOYEE, DEPARTMENT
WHERE Sub=D# AND Department_Name=’IT’
```

Цей запит також можна виразити алгебраїчно за допомогою послідовності операторів:

SQL

```
TABLE := π_City (σ_Department_Name=IT (EMPLOYEE ⋈_{Sub=D#} DEPARTMENT))
```

Цей вираз спочатку виконує з’єднання (join) таблиць EMPLOYEE та DEPARTMENT за спільним номером відділу. Далі з отриманого проміжного результату вибираються співробітники, які працюють у відділі з назвою «IT», і нарешті за допомогою проекції повертаються потрібні міста.

Рисунок 3.1 показує цей вираз алгебраїчних операторів у вигляді відповідного дерева запиту.

Інтерпретація дерева запиту:

Листкові вузли — це дві таблиці EMPLOYEE і DEPARTMENT, що використовуються в запиті. Спочатку вони об’єднуються в одному внутрішньому вузлі (оператор з’єднання — join), потім у другому внутрішньому вузлі (оператор селекції) залишаються лише записи з назвою відділу «IT». Кореневий вузол представляє операцію проекції, яка формує результуючу таблицю з потрібними містами [40].

Корінь та внутрішні вузли дерев запитів посилаються на один або два піддерева. Якщо оператор, що формує вузол, працює з однією таблицею, його називають унарним оператором; якщо він впливає на дві таблиці — бінарним оператором.

Унарними операторами, які можуть обробляти лише одну таблицю, є оператори проекції та селекції (див. рис. 3.2 та 3.3). Бінарними операторами, що приймають дві таблиці як операнди, є об’єднання множин, перетин множин, різниця множин, декартовий добуток, з’єднання (join) та ділення.

Створення дерева запиту є першим кроком у трансляції та виконанні реляційного запиту до бази даних. Таблиці та атрибути, вказані користувачем, повинні бути присутніми в системних таблицях перед подальшою обробкою. Тому дерево запиту використовується для перевірки як синтаксису запиту, так і прав доступу користувача.

Додаткові заходи безпеки, такі як захист даних залежно від значень, можуть оцінюватися лише під час виконання. Другим кроком після контролю доступу та цілісності є вибір та оптимізація шляхів доступу; третім кроком є генерація коду або інтерпретоване виконання запиту.

При генерації коду модуль доступу зберігається в бібліотеці модулів для подальшого використання. Альтернативно, інтерпретатор може взяти на себе динамічне керування для виконання команди.

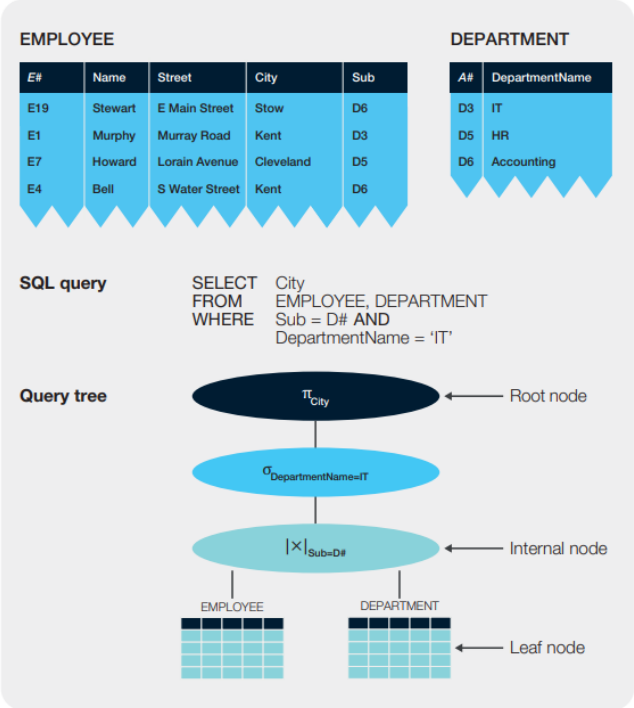

Рисунок 3.1 - Дерево запиту для кваліфікованого запиту до двох таблиць

Оптимізація шляхом алгебраїчних перетворень

Як було показано в розділі 3, окремі оператори реляційної алгебри також можна комбінувати. Якщо такі комбіновані вирази генерують однаковий результат, незважаючи на різний порядок операторів, їх називають еквівалентними виразами. Еквівалентні вирази дозволяють оптимізувати запити до бази даних за допомогою алгебраїчних перетворень без зміни кінцевого результату. Таким чином, зменшуючи обчислювальні витрати, вони становлять важливу частину компонента оптимізації реляційної системи баз даних.

Значний вплив порядку операторів на обчислювальні витрати можна проілюструвати на прикладі запиту з попереднього розділу. Вираз:

SQL

TABLE := π_City (σ_Department_Name=IT (EMPLOYEE ⋈_{Sub=D#} DEPARTMENT))

може бути замінений на такий еквівалентний вираз (див. рис. 3.2):

SQL

TABLE := π_City (π_{Sub,City} (EMPLOYEE) ⋈_{Sub=D#} π_{D#} (σ_Department_Name=IT (DEPARTMENT)))

У цьому випадку спочатку виконується селекція (σ_Department_Name=IT) на таблиці DEPARTMENT, оскільки для запиту актуальний лише відділ IT. Далі виконуються дві операції проекції: одна (π_{Sub,City}) на таблиці EMPLOYEE, інша (π_{D#}) на проміжній таблиці з відділом IT, отриманою на першому кроці. І лише після цього виконується операція з'єднання (⋈_{Sub=D#}) за номером відділу, а наприкінці — остаточна проекція (π_City) для отримання міст.

Хоча кінцевий результат залишається тим самим, обчислювальні витрати в цьому випадку значно нижчі.

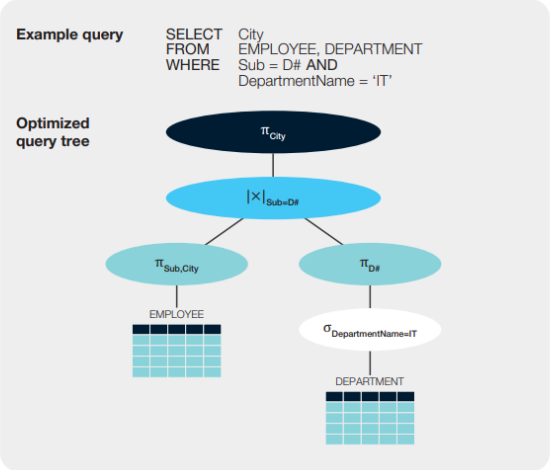

Рисунок 3.2 - Алгебраїчно оптимізоване дерево запиту

Загалом рекомендується розміщувати оператори проекції та селекції в дереві запиту якомога ближче до листків, щоб отримувати лише невеликі проміжні результати перед обчисленням трудомістких і тому дорогих операторів з'єднання.

Успішне перетворення дерева запиту за такою стратегією називається алгебраїчною оптимізацією. При цьому застосовуються такі принципи:

- Кілька селекцій на одній таблиці можна об'єднати в одну, щоб

перевіряти умову селекції лише один раз.

- Селекції слід виконувати якомога раніше, щоб проміжні результати були якомога меншими. Для цього оператори селекції потрібно розміщувати якомога ближче до листків (тобто до вихідних таблиць).
- Проекції також слід виконувати якомога раніше, але ніколи перед селекціями. Операції проекції зменшують кількість стовпців і часто також кількість кортежів.
- Оператори з'єднання (join) слід обчислювати ближче до кореня дерева запиту, оскільки вони вимагають значних обчислювальних витрат.

Окрім алгебраїчної оптимізації, суттєвого приросту продуктивності обробки реляційних запитів можна досягти також завдяки використанню ефективних структур зберігання та доступу. Наприклад, система баз даних може покращувати оператори селекції та з'єднання залежно від розміру задіяних таблиць, порядку сортування, індексних структур тощо.

При цьому важливим є ефективна модель оцінки вартості доступу для вибору між кількома можливими послідовностями обробки. Для цього необхідні формули вартості, які дозволяють розрахувати обчислювальні витрати різних запитів до бази даних, таких як:

- послідовний пошук у таблиці,
- пошук через індексні структури,
- сортування таблиць або підтаблиць,
- використання індексів за атрибутами з'єднання,
- обчислення екві-з'єднань (equi-joins) між кількома таблицями.

Ці формули вартості враховують кількість звернень до фізичних сторінок і створюють зважену оцінку операцій введення/виведення, а також використання процесора (CPU). Залежно від конфігурації комп'ютера формула може сильно залежати від часу доступу до зовнішніх носіїв, кеш-пам'яті, оперативної пам'яті, а також від внутрішньої обчислювальної потужності.

Обчислення операторів з'єднання

Реляційна система баз даних повинна надавати різні алгоритми, які можуть

виконувати операції реляційної алгебри та реляційного обчислення. Вибірка кортежів з кількох таблиць значно дорожча, ніж вибірка з однієї таблиці. У наступному розділі розглядаються різні стратегії з'єднання, хоча звичайні користувачі майже не зможуть впливати на варіанти обчислення.

Реалізація операції з'єднання двох таблиць полягає в тому, щоб порівняти кожен кортеж однієї таблиці з усіма кортежами іншої таблиці щодо умови з'єднання і, у разі відповідності, вставити обидва кортежі в результуючу таблицю як комбінований кортеж.

Щодо обчислення екві-з'єднань (equi-joins), існують дві основні стратегії з'єднання: вкладене з'єднання та сортувально-зливальне з'єднання.

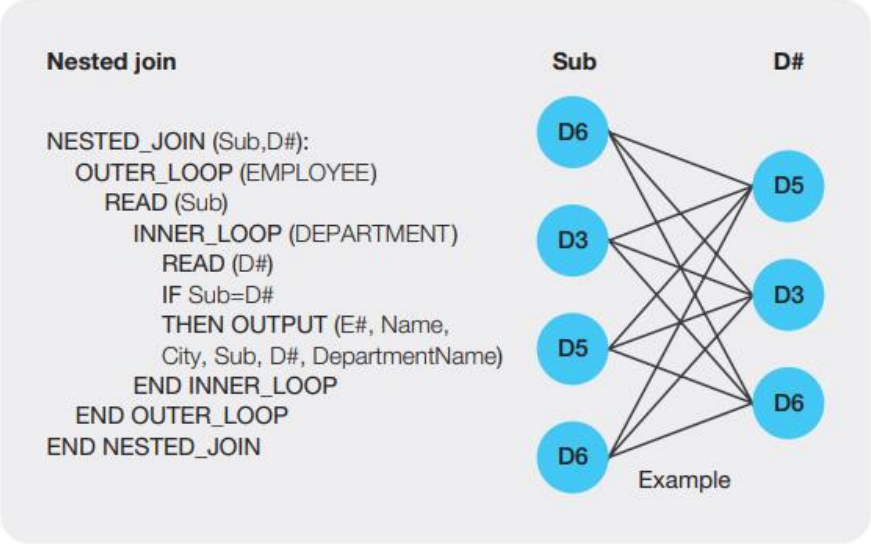

Рисунок 3.3 - Обчислення з'єднання методом вкладення

Вкладене з'єднання (Nested Join)

Для вкладеного з'єднання між таблицею R з атрибутом A та таблицею S з атрибутом B кожен кортеж у R порівнюється з кожним кортежем у S для перевірки виконання умови з'єднання R.A = S.B. Якщо таблиця R містить n кортежів, а таблиця S — m кортежів, це вимагає n × m порівнянь.

Алгоритм вкладеного з'єднання обчислює декартовий добуток і одночасно перевіряє виконання умови з'єднання. Оскільки всі кортежі таблиці R у зовнішньому циклі порівнюються з усіма кортежами таблиці S у внутрішньому

циклі, витрати мають квадратичну складність.

Витрати можна зменшити, якщо для атрибута A та/або атрибута B існує індекс. Рисунок 3.3 ілюструє сильно спрощений алгоритм вкладеного з'єднання інформації про співробітників і відділи на основі раніше введених таблиць.

У алгоритмі чітко видно OUTER_LOOP (зовнішній цикл) та INNER_LOOP (внутрішній цикл), які показують порівняння всіх кортежів таблиці EMPLOYEE з усіма кортежами таблиці DEPARTMENT.

Для операції з'єднання на рис. 3.3 існує індекс для атрибута D#, оскільки він є первинним ключем таблиці DEPARTMENT. Система баз даних використовує індексну структуру для номера відділу, завдяки чому у внутрішньому циклі не переглядається вся таблиця DEPARTMENT кортеж за кортежем, а відбувається прямий доступ до потрібних кортежів через індекс.

Ідеально, якщо для атрибута Sub (підпорядкування) таблиці EMPLOYEE також існує індекс, який система баз даних може використовувати для оптимізації. Цей приклад ілюструє важливість правильного вибору індексних структур адміністраторами баз даних.

Сортувально-зливальне з'єднання (Sort-Merge Join)

Більш ефективний алгоритм, ніж вкладене з'єднання, доступний у тому випадку, коли кортежі таблиць R і S вже фізично відсортовані у висхідному або низхідному порядку за атрибутами A і B умови з'єднання відповідно. Для цього може знадобитися внутрішнє сортування перед власне операцією з'єднання, щоб привести обидві таблиці до потрібного порядку.

Обчислення з'єднання тоді зводиться лише до проходження таблиць у порядку сортування за значеннями атрибутів умови з'єднання та одночасного порівняння значень A і B.

Сортувально-зливальне з'єднання вимагає, щоб таблиці R і S з умовою з'єднання R.A = S.B були відсортовані за значеннями атрибутів A (для R) та B (для S). Алгоритм обчислює з'єднання шляхом порівнянь у порядку сортування. Якщо атрибути A і B є унікальними (наприклад, первинний і зовнішній ключ), обчислювальні витрати є лінійними.

Рисунок 3.5 показує базовий алгоритм сортувально-зливального з’єднання. Спочатку обидві таблиці сортуються за атрибутами, що використовуються в умові з’єднання, потім алгоритм проходить їх у порядку сортування і виконує порівняння R.A = S.B.

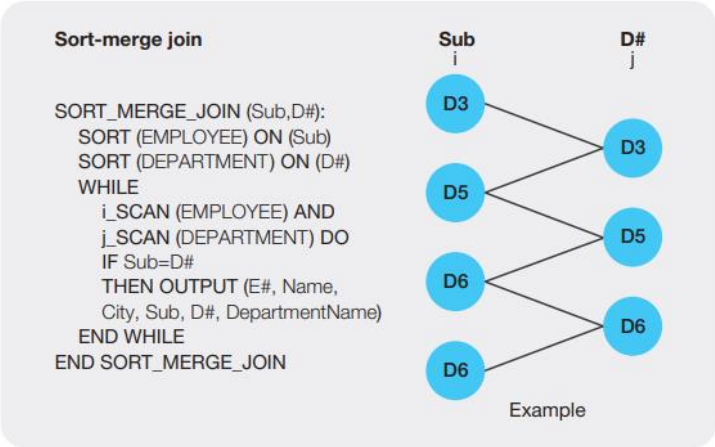

Рисунок 3.5 - Проходження таблиць у порядку сортування

Загалом атрибути A і B можуть мати складнішу залежність, тобто значення атрибутів можуть повторюватися в обох стовпцях R.A і S.B. У такому випадку для певного кортежу з R може існувати кілька сумісних кортежів з S і навпаки. Для таких значень атрибутів відповідні кортежі з R і S потрібно комбінувати через вкладене часткове з’єднання.

У запиті до таблиць EMPLOYEE і DEPARTMENT сортувально-зливальне з’єднання має лінійну залежність від кількості кортежів, оскільки D# є ключовим атрибутом. Алгоритму достатньо пройти обидві таблиці лише один раз для обчислення з’єднання.

Системи баз даних загалом не можуть вибрати відповідну стратегію з’єднання (або будь-яку іншу стратегію доступу) заздалегідь. На відміну від алгебраїчної оптимізації, це рішення залежить від поточного стану вмісту бази даних. Тому вкрай важливо регулярно оновлювати статистичну інформацію в системних таблицях - або автоматично з певними інтервалами, або вручну спеціалістами з баз даних.

Паралельна обробка з використанням MapReduce

Аналіз великих обсягів даних вимагає розподілу завдань з використанням паралелізму для отримання результатів за прийнятний час. Метод MapReduce можна застосовувати як у комп'ютерних мережах, так і на мейнфреймах; у цьому розділі розглядається перший варіант - розподілений.

У розподіленій комп'ютерній мережі, яка часто складається з недорогих горизонтально масштабованих компонентів, обчислювальні процеси розподіляються значно легше, ніж набори даних. Саме тому метод MapReduce набув широкого поширення для веб-пошуку та аналітичних завдань. Він використовує паралельну обробку для створення та сортування простих вибірок даних перед виведенням результатів:

- Фаза Map: Підзавдання розподіляються між різними вузлами комп'ютерної мережі для використання паралелізму. На окремих вузлах на основі запиту витягуються прості пари ключ-значення, які потім сортуються (наприклад, за допомогою хешування) і виводяться як проміжні результати.
- Фаза Reduce: На цьому етапі згадані проміжні результати консолідуються для кожного ключа або діапазону ключів і виводяться як остаточний результат у вигляді списку ключів з відповідними агрегованими значеннями.

Рисунок 3.6 показує простий приклад роботи процедури MapReduce. Потрібно знайти в документах або веб-сайтах терміни: algorithm, database, NoSQL, key, SQL та distribution. Результатом має стати частота появи кожного терміна.

Фаза Map складається з двох паралельних функцій відображення M1 і M2. M1 генерує список пар ключ-значення, де ключами є шукані терміни, а значеннями - їх частота. M2 одночасно виконує подібний пошук на іншому вузлі комп'ютерної мережі з іншими документами або веб-сайтами. Попередні результати потім сортуються за алфавітом за допомогою алгоритму хешування. У верхній частині критерієм сортування є перші літери ключів (шукані терміни) від A до N, у нижній - літери від O до Z.

Фаза Reduce на рис. 3.6 об'єднує проміжні результати. Функція R1

підсумовує частоти для термінів, що починаються з A до N; R2 виконує те саме для термінів від O до Z. Результатом є два списки, відсортовані за частотою термінів: один зі значеннями NoSQL (4), database (3), algorithm (1), а другий - SQL (3), distribution (2), key (1). Остаточний результат об'єднує ці два списки та сортує їх за частотою.

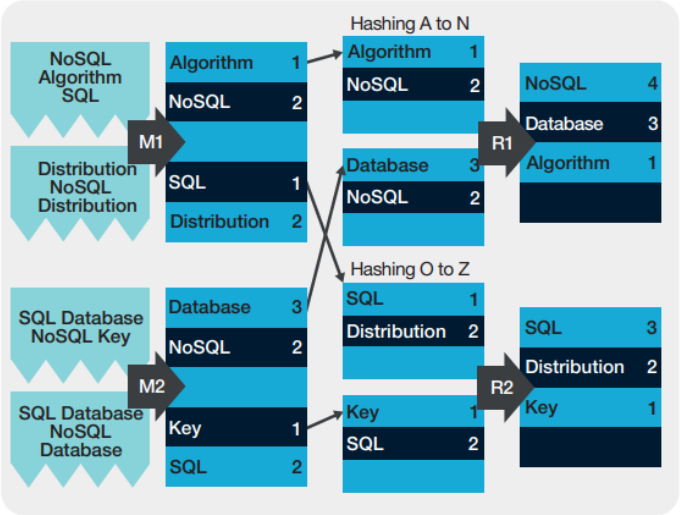

Рисунок 3.6 - Визначення частоти пошукових термінів за допомогою MapReduce

Метод MapReduce базується на функціональних мовах програмування, таких як LISP (LISt Processing). Функція map() обчислює модифікований список для всіх елементів вихідного списку у вигляді проміжного результату. Функція reduce() агрегує окремі результати та зводить їх до вихідного значення.

MapReduce було вдосконалено та запатентовано розробниками Google для роботи з величезними обсягами напівструктурованих і неструктурованих даних. Однак ця функція також доступна в багатьох інструментах з відкритим кодом. Процедура відіграє важливу роль у базах даних NoSQL, де різні виробники використовують цей підхід для отримання записів з бази даних.

Завдяки використанню паралелізму метод MapReduce корисний не лише для аналізу даних, а й для розподілу навантаження, передачі даних, розподіленого пошуку, категоризації та моніторингу.

Шарова архітектура

Важливим правилом архітектури систем баз даних вважається те, що майбутні зміни або розширення повинні бути локально обмеженими. Подібно до реалізації операційних систем чи інших програмних компонентів, у реляційних і нереляційних системах баз даних вводяться повністю незалежні системні шари, які взаємодіють через чітко визначені інтерфейси.

Рисунок 3.7 дає загальний огляд п’яти шарів системної архітектури на основі реляційної технології баз даних. У наступному розділі показано, як ці шари відповідають основним функціям, описаним у розділі 4 та попередніх розділах глави 5.

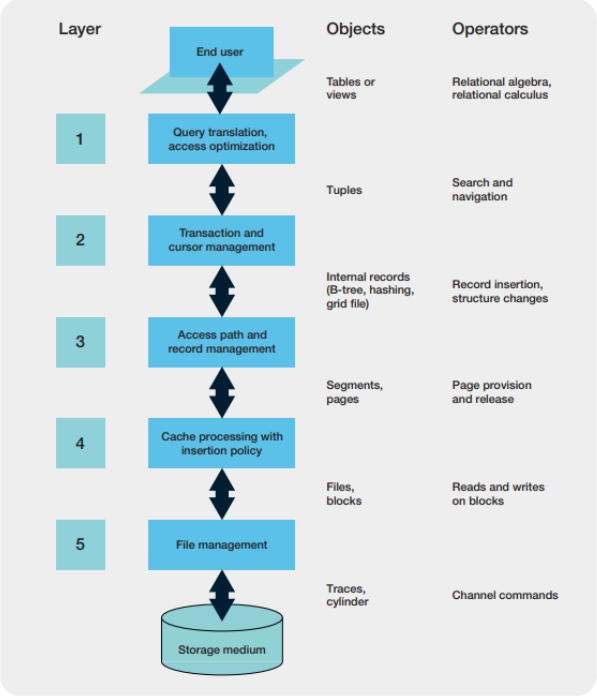

Рисунок 3.7 - П’ятишарова модель для реляційних систем баз даних

Шар 1: Інтерфейс, орієнтований на множини Перший шар використовується для опису структур даних, надання множинних операцій, визначення умов доступу та перевірки обмежень цілісності. Під час ранньої трансляції та генерації модуля доступу або під час виконання необхідно перевіряти синтаксис, розв’язувати імена та вибирати шляхи доступу. Вибір шляхів доступу відкриває значний простір для оптимізації.

Шар 2: Інтерфейс, орієнтований на записи Другий шар перетворює логічні записи та шляхи доступу у фізичні структури. Концепція курсору (cursor)

дозволяє здійснювати навігацію або обробку записів відповідно до фізичного порядку зберігання, позиціонувати конкретні записи в таблиці або надавати записи, відсортовані за значенням.

Для забезпечення узгодженості бази даних і запобігання взаємним блокуванням (deadlocks) між різними запитами користувачів необхідно використовувати управління транзакціями [21].

Шар 3: Структури зберігання та доступу Третій шар реалізує фізичні записи та шляхи доступу на рівні сторінок. Кількість форматів сторінок обмежена, але окрім деревоподібних структур і методів хешування в майбутньому повинні підтримуватися багатовимірні структури даних. Ці загальні структури зберігання призначені для ефективного доступу до зовнішніх носіїв інформації.

Для подальшої оптимізації управління записами та шляхами доступу також можна використовувати фізичне кластеризування та багатовимірні шляхи доступу.

Шар 4: Розподіл сторінок З метою підвищення ефективності та підтримки процедур відновлення четвертий шар поділяє лінійний адресний простір на сегменти з однаковими межами сторінок.

Система управління файлами надає сторінки в кеш за запитом. З іншого боку, сторінки можуть вставлятися в кеш або замінюватися в ньому відповідно до політики вставки та заміни. Існує не лише пряме призначення сторінок блокам, а й непряме призначення, наприклад, методи кешування, які дозволяють атомарно вставляти кілька сторінок у кеш бази даних.

Шар 5: Розподіл пам’яті П’ятий шар реалізує структури розподілу пам’яті та забезпечує блочно-орієнтоване управління файлами для вищележачого шару. Властивості апаратного забезпечення залишаються прихованими від файлових та блочно-орієнтованих операцій.

Управління файлами зазвичай підтримує динамічно зростаючі файли з визначуваними розмірами блоків. В ідеалі також має бути можливість кластеризувати блоки та виконувати введення/виведення кількох блоків за одну операцію.

3.2 Розроблення та реалізація тестового програмного додатка

Для цілей цієї роботи було створено веб-застосунок для соціальних мереж. Застосунок у певний момент може використовувати одну з трьох баз даних – PostgreSQL, MongoDB або Cassandra, залежно від конфігурації. Він надає такі функціональні можливості, як надсилання постів, позначення постів хештегами, додавання коментарів до постів, підписки на інших користувачів, перегляд часової шкали, яка містить пости користувачів, на яких ви підписані, відсортовані за датою у порядку спадання, та перегляд усіх постів, позначених певним хештегом. Однією з вимог також була реалізація пагінації під час отримання повідомлень. Важливо, що пагінація здійснювалася безпосередньо в базі даних, а не в застосунку. Нам вдалося досягти цієї мети для всіх вибраних баз даних.

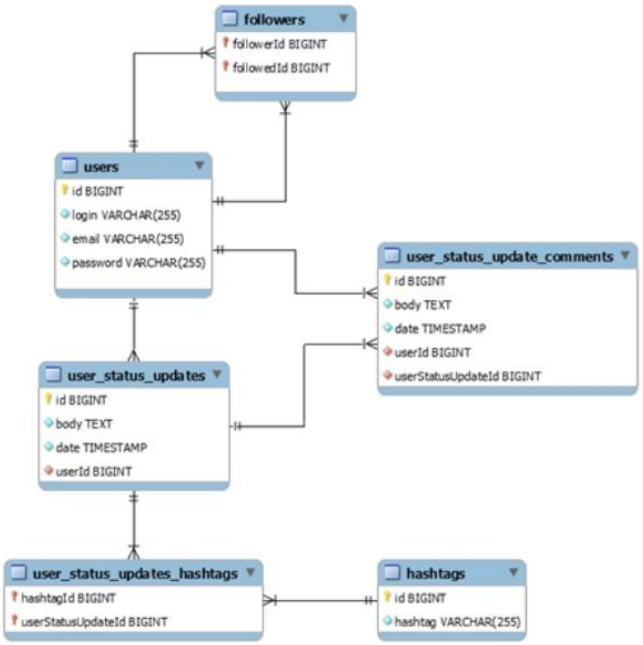

Рисунок 3.8 - Модель даних реляційної бази даних

Додаток було написано на Java 8 та JavaScript. Були використані наступні фреймворки та бібліотеки:

— Spring Boot [15] дозволяє дуже просто створювати веб-застосунки на Java. Вся застосунок являє собою один JAR-файл із вбудованим Tomcat, його можна запускати як стандартну консольну програму Java.

— AngularJS [16] - це JavaScript-фреймворк, що забезпечує такі функції, як автоматичне зв'язування даних між виглядом та моделлю та впровадження залежностей.

— Spring JDBC [17] – спрощує використання драйвера JDBC завдяки автоматичному відкриттю та закриттю з’єднань, наборів результатів та операторів, обробці винятків SQLException, обробці транзакцій, ітерації по наборах результатів.

Не використовувався інструмент ORM (об'єктно-реляційне відображення) (наприклад, Hibernate), оскільки це могло вплинути на продуктивність програми.

Реалізація SQL

Додаток було протестовано за допомогою PostgreSQL. Для доступу до даних SQL використовувалася бібліотека Spring JDBC. Рис. 1 містить модель даних для реляційної бази даних.

Одним із найскладніших запитів, що використовувалися в застосунку, був запит, який вибирає часову шкалу користувача. У випадку бази даних SQL він має таку структуру:

```
SELECT user.login login, update.id id, update.date date, update.body body
FROM user_status_updates update
JOIN users user ON user.id = update.userId
JOIN followers f ON f.followedId = user.id
WHERE f.followerId = ?
ORDER BY update.date DESC
LIMIT 20 OFFSET (CURRENT_PAGE - 1) * 20
```

Рисунок 3.9 - Запит вибирає часову шкалу користувача

Реалізація MongoDB

Для доступу до даних MongoDB було використано офіційний драйвер Java. Що стосується реляційної бази даних, то на рис. 3.10 наведено модель даних для бази даних MongoDB. Вона містить три колекції документів (документи **з** коментарями вкладені в документи status_updates). Вкладення даних призводить до меншої кількості об'єктів даних, ніж у реляційній базі даних.

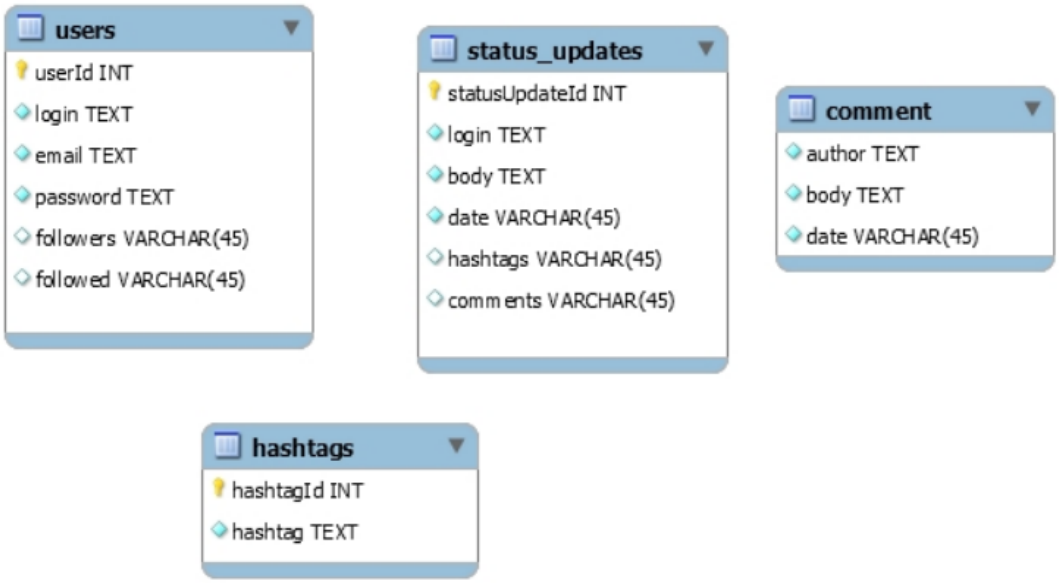

Рисунок 3.10 - Модель даних MongoDB

Запит, який повертає часову шкалу користувача, виглядає так:

```
db.status_updates.
find({"login": {"$in": ["?","?"]}}).
sort({date: -1}).
skip((CURRENT_PAGE - 1) * 20).
limit(20);
```

Рисунок 3.11 – Запит повертає часову шкалу користувача

Де замість знаків питання ми ставимо логіни користувачів, на яких ви підписані.

Впровадження Cassandra

Драйвер DataStax використовувався для доступу до даних Cassandra. Як і для баз даних SQL та MongoDB. Рис. 3 містить схему моделі даних. Жовті ключі позначають ключі розділів, а червоні – ключі кластеризації. Стрілки вказують напрямок сортування для стовпця, визначеного під час створення таблиці. Ця модель даних базується на моделі, запропонованій Брауном [13].

Рисунок 3.12 - Модель даних Cassandra

У випадку з Cassandra вибір часової шкали користувача є складнішим. Для першої сторінки запит даних виглядає так:

```
SELECT statusUpdateLogin, statusUpdateId,
toTimestamp(statusUpdateId) as date, body
FROM user_status_update_timeline  WHERE timelineLogin = ?
```

Рисунок 3.13 - Cassandra вибір часової шкали користувача

Для кожної наступної сторінки нам довелося додавати ще одну умову в речення WHERE:

```
SELECT statusUpdateLogin, statusUpdateId,
toTimestamp(statusUpdateId) as date, body
FROM user_status_update_timeline
WHERE timelineLogin = ? and statusUpdateId < ?
```

Рисунок 3.14 - Cassandra вибір часової шкали користувача з додаванням ще одної умови

Там, де замість знаків питання ми ставимо ідентифікатор останнього оновлення статусу з попередньої сторінки, наприклад, для другої сторінки це буде ідентифікатор останнього оновлення статусу з першої сторінки (20-те оновлення

статусу).

Порівняння моделей

Моделі даних для порівнюваних баз даних абсолютно різні. Модель даних SQL була розроблена таким чином, щоб уникнути надмірності та використовувати зв'язки між даними. Тому в запитах є багато об'єднань, що може бути дуже неефективним для великих наборів даних.

Модель даних для MongoDB є найпростішою. Завдяки використанню таких функцій, як вкладені документи та масиви, вона складається з трьох колекцій документів.

Модель даних для Cassandra є найскладнішою. Вона була розроблена відповідно до документа DataStax [14], де — одна таблиця на шаблон запиту, щоб уникнути читання з кількох розділів. Тому в цій моделі даних є багато надлишковості. Наприклад, один пост зберігається 1 + number_of_followers раз - один раз у таблиці user_status_updates **та** number_of_followers раз у таблиці user_status_update_timeline . Це дозволяє отримати часову шкалу користувача, запитуючи лише один розділ. Таблиця, що зберігає хештеги, також є складнішою, ніж в інших базах даних. Вона містить три стовпці — стовпець префікса містить перші два символи хештегу, решта — решту, а стовпець хештегу містить весь хештег. Мова запитів Cassandra (CQL) не підтримує оператор like**,** відомий з SQL, і така структура дозволяє виконувати операцію повнотекстового пошуку в Cassandra.

Тести продуктивності

Усі тести продуктивності проводилися на ПК з такими характеристиками:

— Процесор Intel Core i5-4460 3,2 ГГц,

— 16 ГБ оперативної пам'яті DDR3,

— Western Digital Blue 1 ТБ SATA 3 7200 об/хв,

— Домашня версія Windows 10.

Тест використовує такі бази даних:

— PostgreSQL 9.5 для Windows x64,

— MongoDB 3.2 для Windows x64,

— Апач Кассандра 3.7.

Для максимальної надійності перед кожним тестом проводилася дефрагментація. Як інструмент для підтримки тестів було обрано JMeter. Для MongoDB та Cassandra параметри запису були налаштовані таким чином, що запис був успішним лише після збереження даних на фізичному диску. Щоб максимізувати швидкість зчитування даних у базах даних, індекси були визначені для стовпців, що використовуються в умовах запиту.

Моделювання трафіку користувачів

Першим типом тестів були ті, що імітували використання застосунку 100 користувачами одночасно. Кожен тест тривав 5 хвилин. План тестування був наступним:

— увійти до програми,

— переглянути перші чотири сторінки повідомлень, надісланих поточним користувачем,

— переглянути перші чотири сторінки публікацій із хронології поточного користувача,

— надішліть новий пост, позначений двома хештегами.

Кожну базу даних було протестовано на чотирьох різних наборах даних. Кожен набір даних містив різну кількість користувачів та публікацій: 1000 користувачів та 1 мільйон публікацій, 5000 користувачів та 5 мільйонів публікацій, 10 000 користувачів та 10 мільйонів публікацій, 15 000 та 15 мільйонів публікацій. Для кожного набору даних кожен користувач підписався на 100 інших користувачів.

На рис. 3.15 міститься інформація про кількість тестових циклів, виконаних протягом 5-хвилинного тесту. Для невеликого набору даних PostgreSQL є найшвидшою базою даних, але її ефективність різко падає зі збільшенням обсягу даних. Для найбільших наборів даних кількість виконаних тестових циклів у кілька разів менша, ніж для інших баз даних. У табл. 1 показано, що найповільнішою операцією PostgreSQL є читання постів з часової шкали. MongoDB є найефективнішою для великих наборів даних. Її продуктивність дещо

знизилася лише для найбільшого набору даних. Cassandra зафіксувала значне падіння продуктивності лише для 15 000 користувачів.

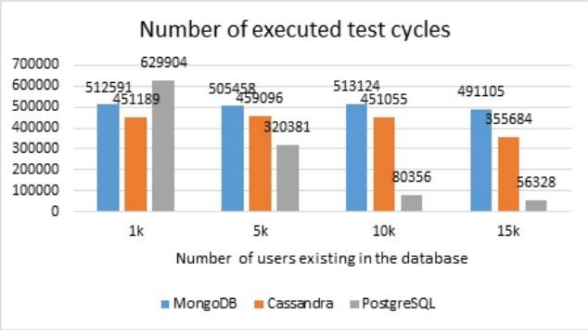

Рисунок 3.15 - Кількість циклів, виконаних протягом 5-хвилинного тесту

Таблиця 3.1

Середній час виконання окремих операцій

Середній час виконання [мс](Average execution time [ms])	**PostgreSQL**	**MongoDB**	**Cassandra**
Кількість користувачів [*10³] *(Number of users [*10^3])*	**1 | 5 | 10 | 15**	**1 | 5 | 10 | 15**	**1 | 5 | 10 | 15**
Вхід у систему *(Log in)*	30 | 49 | 36 | 24	4 | 3 | 4 | 2	29 | 27 | 26 | 31
Читання однієї сторінки публікацій *(Read one page of posts)*	36 | 66 | 47 | 24	3 | 2 | 3 | 2	24 | 23 | 23 | 30
Читання однієї сторінки стрічки *(Read one page of timeline)*	43 | 99 | 828 | 1236	6 | 5 | 6 | 5	24 | 24 | 25 | 37
Надіслати нову публікацію *(Send new post)*	122 | 214 | 196 | 264	535 | 552 | 537 | 570	421 | 414 | 424 | 520

Вставка даних

Ще однією досліджуваною операцією була вставка даних. Один тест полягав у вставці 1000 записів до таблиці/колекції, яка зберігає публікації. Результати наведено на рис. 3.16. На ньому показано, що MongoDB є найповільнішою при додаванні даних. Це результат використання журнальованого запису, який призводить до повернення статусу успішного завершення в базі даних лише після збереження даних на фізичному диску.

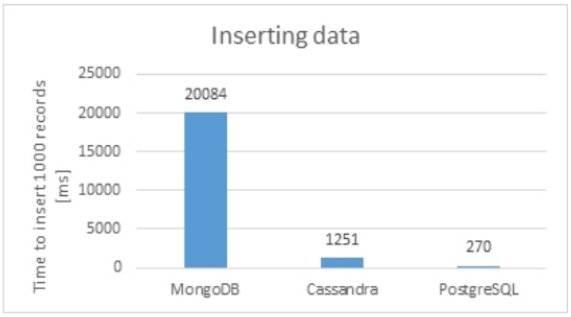

Рисунок 3.16 - Результати вставки даних

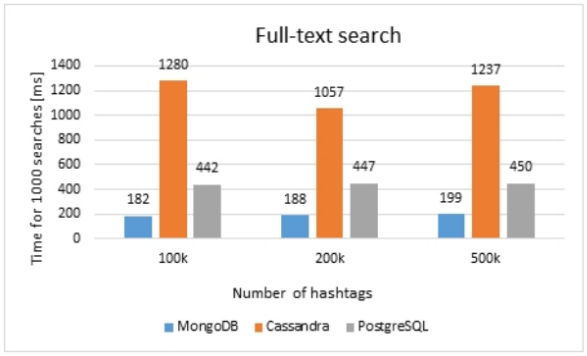

Рисунок 3.17 - Результати повнотекстового пошуку

Повнотекстовий пошук

Останнім тестом був пошук хештегів, що починаються із заданого шаблону. Для кожної бази даних було проведено три тести, кожен для різної кількості хештегів. На рис. 6 показано, скільки часу потрібно для виконання 1000 повнотекстових пошуків. Виявляється, що найповільнішою є Cassandra, якій потрібно в кілька разів більше часу для виконання цього завдання, ніж іншим базам даних. MongoDB приблизно вдвічі швидша за PostgreSQL.

3.3 Аналіз результатів роботи та оцінка ефективності програмного забезпечення

Дискусія SQL проти NoSQL не стосується реляційних та нереляційних баз даних, незважаючи на назви. Аналізуються та порівнюються моделі транзакцій обох баз даних. Під час виконання програми база даних виконує серію операцій,

відомих як «транзакції ». Усі транзакції в базі даних SQL залежать від властивостей ACID. У цьому випадку ACID посилається на принципи атомізму, узгодженості, ізоляції та довговічності. Однак розробники баз даних NoSQL дійшли висновку, що властивість ACID є марною перешкодою для захисту величезних даних. Отже, на початку 2000-х років професор Ерік Брюер запропонував нову теорію. Принципи CAP - це узгодженість, доступність та толерантність до розділів. Теорема стверджує, що всі ці якості можуть бути отримані одночасно розробниками, які працюють у розподіленому середовищі. Як компроміс для гарантованої узгодженості та доступності, розробники можуть реалізувати базу даних, толерантну до розділів, використовуючи модель CA. Якщо розробник бази даних надає більше значення доступності та толерантності до розділів, ніж узгодженості, то не можна сказати, що база даних базується на AP. Щоб досягти узгодженості та толерантності до розділів без шкоди для доступності, розгортається база даних на основі CP. Таблиця У таблиці 3.2 описано основні функції, що підтримуються обома типами баз даних.

Таблиця 3.2

Функції СУБД.

Характеристика	**RDBMS (Реляційні)**	**NoSQL (Нереляційні)**
Дані	S *(Структуровані)*	SUSm *(Напів-/Неструктуровані)*
Схема	Фіксована	Динамічна
Масштабованість	Вертикальна	Горизонтальна
Відповідність	ACID	BASE
Архітектура	Централізована	Розподілена
Узгодженість	Сувора (Strict)	У кінцевому підсумку (Eventual)
Мова запитів	SQL	OO API, SQL-подібна
Продуктивність	Повільна	Швидка
Найкраще для	BFT *(Фінансові транзакції)*	LSWA, SD *(Масштабні вебдодатки)*

Стандартизація класифікації СУБД призвела до появи кількох підкатегорій. Основою будь-якої системи управління базами даних є модель даних, на якій вона побудована. Багато сучасних комерційних СУБД значною

мірою спираються на реляційну модель даних. Об'єктна модель даних отримала мало поширення, хоча вона використовується в невеликій кількості комерційних систем. Багато застарілих програм продовжують працювати на системах баз даних на основі ієрархічних та мережевих моделей даних. Ми можемо класифікувати бізнес-модель NoSQL як одну з наступних, як визначено в таблиці 3.3.

Системи баз даних NoSQL отримали різноманітні рекомендації, що можна пояснити тим фактом, що структура реалізації не включає традиційний SQL. Кілька різних типів баз даних NoSQL використовують унікальний набір процедур запитів. Розробники програмного забезпечення часто відповідають за належне проектування виконання запитів, а не покладаються на декларативну мову запитів, яка поєднує питання та забезпечує швидкі плани виконання запитів. Це пояснюється тим, що декларативна мова запитів може поєднувати питання. Користувачі системи відповідають за виконання додаткових завдань, таких як перевірка та інтерпретація зібраної інформації. Крім того, відповідальність за забезпечення узгодженості даних, реплікації даних та доступності під час одночасних змін у спільних та реплікованих базах даних все більше переходить до рук розробників. Поєднання масивних систем даних з базами даних NoSQL може мати кілька різних наслідків для проектування та архітектури програмного забезпечення.

Брюер сформулював набір вимог до розподілених баз даних, використовуючи свій метод CAP [7]. Ці вимоги слугують базовими. У випадку, коли є мережевий розділ (P: у кластері, коли інформація випадковим чином втрачається між вузлами), узгодженість (C: представлення однакових даних усім користувачам) повинна мати пріоритет над доступністю (доступністю для користувачів) (A: підтвердження має бути отримане кожним клієнтом як успіх або невдача). Якщо розділу немає (P), фреймворк сприятиме затримці.

(L) над узгодженістю (C), як показано в PARCELS Абаді; проте, коли є розділення (P), фреймворк наголошуватиме на доступності (A) над узгодженістю (C).

Сьогодні недостатньо покладатися виключно на структуровані дані,

оскільки неструктурована природа астрономічних даних вимагає швидкої аналітики даних та методів вилучення інформації. Можливості баз даних SQL обмежені в їхній здатності ефективно обробляти різні форми великих даних. Можливості баз даних NoSQL дозволяють ефективно обробляти масивні набори даних. Бази даних NoSQL перевершують у сферах розширення ємності сховища, гнучкості схеми, масштабованості та доступу в режимі реального часу. Бази даних NoSQL дотримуються атрибута BASE. Замість того, щоб надавати пріоритет узгодженості та безпеці даних, бази даних NoSQL роблять акцент на покращенні ефективності читання/запису даних. Додаток відповідає за забезпечення узгодженості даних. З цієї причини це чудовий варіант для обробки великих наборів даних. Крім того, бази даних NoSQL не застосовують обмеження на рівні даних, стовпців або таблиць, як це використовується в структурованих базах даних.

Таблиця 3.3

Класифікація бази даних.

Тип / Концепція	**Опис та характеристики**	**Приклади систем**
CA *(Consistency & Availability)*	**Проблеми зі стійкістю до розділення:** Цей дизайн бази даних надає пріоритет узгодженості даних (consistency) та доступності (accessibility) за допомогою підходу реплікації. Частини бази даних не дбають про стійкість до розділення (partition tolerance). Якщо вузли розділяються, дані розсинхронізуються.	Vertica, Greenplum, реляційні системи керування базами даних.
CP *(Consistency & Partition Tolerance)*	**Проблеми з узгодженістю та стійкістю до розділення, які необхідно вирішувати:** Головна мета такої системи керування базами даних — забезпечити цілісність даних, які вона зберігає. Однак висока доступність більше не підтримується. Дані зберігаються на різних вузлах, і коли один вузол виходить з ладу, це призводить до того, що дані, які забезпечують узгодженість між ними.	Hypertable, BigTable, HBase.
AP *(Availability & Partition Tolerance)*	**Найвищим пріоритетом є забезпечення доступності даних та стійкості до розділення:** Якщо між вузлами стається збій зв'язку, це не впливає на стан окремого вузла. Після того, як розділення усунено, дані ресинхронізуються, але узгодженість не гарантується. Цих принципів дотримуються такі бази даних, як Riak, та KAI.	Riak, CouchDB, KAI.
Базова доступність *(Basically Available)*	Коли один елемент бази даних виходить з ладу, але інші продовжують працювати, ця секція бази даних вважається «базово доступною». У разі відмови вузла робота.	*Не вказано в тексті*

Продовження таблиці 3.3

	продовжуватиметься шляхом реплікації даних між рештою вузлів	
Гнучкий стан *(Soft State)*	У гнучкому стані дані можуть змінюватися з часом залежно від таких факторів, як рівень активності користувача. Корисність такої інформації також може погіршуватися після закінчення певного періоду. Тому, щоб інформація залишалася корисною в системі, необхідно або оновлювати, або повторно отримувати дані.	*Не вказано в тексті*
Узгодженість у кінцевому підсумку *(Eventual Consistency)*	Відповідно до концепції узгодженості в кінцевому підсумку, після кожної модифікації дані не стають миттєво узгодженими по всій системі, але вони стануть узгодженими з часом. Стверджується, що в осяжному майбутньому дані залишатимуться точними.	*Не вказано в тексті*

У цьому дослідженні проаналізовано приблизно 140 попередніх досліджень, які порівнювали корисність, продуктивність та надійність баз даних SQL з базами даних NoSQ . У цьому дослідженні розглядалися величезні обсяги даних. Згідно з нашими дослідженнями, бази даних NoSQL пропонують більшу можливість розширення [23]. Бази даних SQL краще підходять для транзакційних систем і використовують більше ресурсів для забезпечення цілісності та узгодженості даних, тоді як бази даних NoSQL краще оснащені для обробки величезних та різноманітних наборів даних і використовують менше ресурсів для забезпечення цілісності та узгодженості даних. З іншого боку, бази даних NoSQL [24] відрізняються тим, що вони надають більшого пріоритету доступності даних. Результати нашого тестування показують, що реляційні бази даних не можуть бути ефективно замінені базами даних NoSQL. Оскільки обидві бази даних мають свої переваги та недоліки, база даних, яку організація вирішить використовувати, залежатиме від вимог, унікальних для цього бізнесу. Для прикладу, бази даних NoSQL, які дозволяють використовувати програмний модуль MapReduce, краще підходять для паралельних обчислень [16], коли вони реалізовані в кластерному середовищі.

Бази даних NoSQL побудовані на основі більш гнучкої та динамічної схеми, на відміну від реляційних баз даних, які сильно залежать від попередньо визначеної структури даних, що називається «схемою» (таблична форма) [18]. Наприклад, для відстеження даних студентів слід використовувати поля

StdRegNo, StdName та StdAddress. Під час роботи з реляційними базами даних перше , що потрібно зробити, це побудувати схему, яка задовольняє всі необхідні вимоги домену та цілісності.

Після цього можна буде зберегти відповідні дані студентів та дотримуватися необхідних обмежень. Розглядаючи можливість розширення обсягу поточної бази даних шляхом додавання двох нових стовпців, слід врахувати це. Ті, хто буде відповідати за внесення змін до існуючої схеми, також будуть відповідати за перенесення даних з попередньої схеми до нової. Під час роботи з великими наборами даних процедура може стати непрактично повільною та затягнутою. Той факт, що бази даних NoSQL не оновлюються автоматично щоразу, коли до схеми вносяться нові зміни, є найважливішою проблемою для гнучкої розробки програмного забезпечення. [25]. Під час роботи з базами даних NoSQL не обов'язково мати заздалегідь визначену схему для внесення будь-яких змін. Таблиця А4 Додатка А містить список кількох баз даних, до яких було здійснено доступ під час обраного дослідження.

Нормалізація даних для контролю викидів, реляційних схем (атрибутів), обмежень домену, обмежень перевірок , обмежень унікальності та обмежень Not NULL, серед іншого, сприяє безпечній інтеграції даних у реляційних базах даних [37] на відміну від баз даних NoSQL [24]. Реляційні бази даних забезпечують добре налагоджені системи безпеки та автентифікації для користувачів. Бази даних SQL перевершують бази даних NoSQL як за продуктивністю, так і за довговічністю. SQL є стандартним інтерфейсом для всіх реляційних баз даних; однак бази даних NoSQL не використовують SQL. Натомість вони використовують інший інтерфейс.

Бази даних SQL та NoSQL використовують широкий спектр моделей [40]. Крім того, існує різниця в базовій моделі даних, що використовується кожним типом бази даних NoSQL [17]. Враховуючи поширеність кількох постачальників послуг зв'язку (CSP), важко забезпечити перенесення даних. Швидке розширення комерційних баз даних створює ще одну труднощі для хмарного сховища. Натомість, AWS DBaaS [1] пропонує дещо обмежені можливості розширення

сховища для хмарних баз даних Azure. Категорії продуктів, архітектури, типи баз даних та мови наведено в таблиці. 3.4 .

Впровадження складних систем розподілу необхідне для досягнення значного рівня як доступності, так і масштабованості. Крім того, шардування та розподіл відбуваються на базових рівнях архітектури програмного забезпечення, які називаються рівнем додатків, рівнем кешування та рівнем серверного сховища відповідно. Однак досягнення високої масштабованості може бути складним завданням через атомарну абстракцію, яку представляє типовий фреймворк, що використовує SQL як золотий стандарт непроцедурної мови.

Таблиця 3.4

Категорія продукту бази даних, тип архітектури та мови.

Назва бази даних	**Категорія бази даних**	**Архітектура бази даних**	**Тип бази даних**	**Мова написання**
MongoDB	NoSQL - Документоорієнтована	Розподілена мультимодельна	Відкритий код (Open Source)	C++, Go, JavaScript, Python
MySQL	SQL (Реляційна)	Класична / Локальна	Відкритий код (Open Source)	C, C++
Oracle	SQL (Реляційна)	Класична / Локальна	Закритий код (Пропрієтарна)	Асемблер, C, C++
SQL Server	SQL (Реляційна)	Класична / Локальна	Закритий код (Пропрієтарна)	C, C++
Neo4j	NoSQL - Графова	Класична / Локальна	Відкритий код (Open Source)	Java
Couchbase	NoSQL - Документоорієнтована	Розподілена мультимодельна	Відкритий код (Open Source)	C++, Erlang, C, Go
CouchDB	NoSQL - Документоорієнтована	Розподілена мультимодельна	Відкритий код (Open Source)	Erlang, JavaScript, C, C++
Cassandra	NoSQL - Стовпчикова	Розподілена мультимодельна	Відкритий код (Open Source)	Java
Rethink	NoSQL - Документоорієнтована	Розподілена мультимодельна	Відкритий код (Open Source)	C++, Python, Java, JavaScript, Bash
MariaDB	SQL (Реляційна)	Класична / Локальна	Відкритий код (Open Source)	C, C++, Perl, Bash
RavenDB	NoSQL - Документоорієнтована	Класична / Локальна	Відкритий код (Open Source)	C#
BigTable	NoSQL - Стовпчикова	Класична / Локальна	Закритий код (Пропрієтарна)	C++ (core), Java, Python, Go.

Продовження таблиці 3.4

DynamoDB	NoSQL - Ключ/Значення	Класична / Локальна	Закритий код (Пропрієтарна)	Java
SimpleDB	NoSQL - Ключ/Значення	Розподілена база даних	Закритий код (Пропрієтарна)	Erlang
PostgreSQL	SQL (Реляційна)	Класична / Локальна	Відкритий код (Open Source)	C
PostGIS	SQL (Реляційна)	Класична / Локальна	Відкритий код (Open Source)	C
Hbase	NoSQL - Стовпчикова	Розподілена мультимодельна	Відкритий код (Open Source)	Java
GraphDB	NoSQL - Графова	Розподілена база даних	Не вказано	Java
Sybase	SQL (Реляційна)	Класична / Локальна	Закритий код (Пропрієтарна)	SQL
Redis	NoSQL - Ключ/Значення	Класична / Локальна	Відкритий код (Open Source)	ANSI C
HyperTable	NoSQL - Стовпчикова	Класична / Локальна	Відкритий код (Open Source)	C++
JanusGraph	NoSQL - Графова	Класична / Локальна	Відкритий код (Open Source)	Java
VoltDB	СКБД в оперативній пам'яті	Класична / Локальна	Відкритий код (Open Source)	Java, C++
Infogrid	NoSQL - Графова	Класична / Локальна	Відкритий код (Open Source)	Java
H2	SQL (Реляційна)	Класична / Локальна	Відкритий код (Open Source)	Java
ArangoDB	NoSQL БД	Класична / Локальна	Відкритий код (Open Source)	C++, JavaScrip
Azure SQL	SQL (Реляційна)	Класична / Локальна	Закритий код (Пропрієтарна)	C, C++
Azure DocumentDB	NoSQL - Документоорієнтована	Класична / Локальна	Відкритий код (Open Source)	SQL (Core) API

Крім того, програмне забезпечення має бути інтелектуальним, щоб вирішувати проблеми, що виникають з репліками даних, та невідповідності, спричинені колізіями між змінами в репліках, що відбуваються одночасно. Що стосується критеріїв якості, таких як масштабованість, узгодженість, продуктивність та довговічність, кожен різновид NoSQL- бази даних має свій унікальний набір недоліків та обмежень. Крім того, архітектор принципово повинен проводити дослідження характеристик номінальної бази даних, щоб відповідати вимогам, що висуваються постачальником, під час вибору відповідної бази даних. Прогалини кожного різновиду NoSQL-баз даних описані в таблиці 3.5

Термін «великі дані» часто використовується для позначення даних, отриманих з великомасштабних сенсорних мереж або інформації, згенерованої користувачами із соціальних мереж [38]. Однак, оскільки багато людей не мають необхідних навичок, вони не знають, як обробляти дані для досягнення бажаного результату для свого бізнесу. Це пояснюється тим, що значна кількість технологій, методів та дисциплін, що стосуються великих даних, є відносно новими розробками. Реляційні сховища даних, з іншого боку, не здатні обробляти такий великий набір даних, що вимагає створення нових можливостей системи зберігання. Дані, що зберігаються в базі даних NoSQL, не потребують централізованої схеми. Завдяки своїй більшій масштабованості, доступності даних, відмовостійкості та швидкій обробці величезних обсягів неструктурованих даних, бази даних NoSQL підходять для обробки великих даних. Існує багато питань, які ще не вирішені через відмінності між реляційними базами даних та базами даних NoSQL. У роботі робиться висновок, що NoSQL не є придатною альтернативою реляційним базам даних. Крім того, NoSQL є чудовим варіантом для гетерогенних великих даних. Кожна з цих галузей має численні незаповнені прогалини в дослідженнях. Простота, масштабованість та продуктивність проектів баз даних NoSQL є областями зі значним простором для дослідження.

На відміну від суворої структури даних схеми реляційних баз даних, бази даних NoSQL використовують модель даних типу «ключ/значення» у вигляді плоского файлу (табличну форму). Бази даних NoSQL, завдяки своїй горизонтальній масштабованості, добре підходять для обробки величезної кількості та різноманітності даних, що використовуються наразі. Однак це не стосується баз даних SQL, які можуть масштабуватися вертикально. При порівнянні NoSQL з реляційними базами даних масштабованість та продуктивність є двома найважливішими факторами. Також залишається без відповіді дослідницьке питання про те, як інтегрувати функції [19] реляційних баз даних з функціями нереляційних баз даних. Для баз даних NoSQL також бракує досліджень щодо проблеми проектування простих схем.

Таблиця 3.5

Прогалини між NoSQL базами даних.

Тип бази даних	Опис та особливості	Сценарії застосування / Обмеження
Сховища «ключ-значення» (Key-value stores)	Оптимальний вибір, якщо додатку потрібно просто зберігати та отримувати елементи даних, які є прозорими для СКБД і можуть бути ідентифіковані за унікальним ключем.	**Обмеження:** Програмне забезпечення зазнає збою, якщо спробує виконати запит до БД на основі значення будь-якого іншого атрибута, окрім ключа. Крім того, неможливо оновити або отримати лише одне поле з документа у такому сховищі.
Документоорієнтовані БД (Document databases)	Чудовий варіант, коли додатки потребують детального (гранулярного) контролю над тим, які саме записи отримувати, які поля всередині запису змінювати та які поля вилучати на основі критеріїв, відмінних від первинного ключа.	**Перевага:** Документоорієнтовані сховища даних забезпечують значно більшу гнучкість виконання запитів порівняно зі сховищами «ключ-значення».
Стовпчикові БД (Column-family stores)	Ефективне рішення для випадків, коли додаткам необхідно зберігати дані з сотнями або тисячами полів, але у більшості запитів потрібен доступ лише до певної підмножини цих полів.	**Особливість:** Такі репозиторії даних чудово підходять для обробки та зберігання надвеликих масивів даних (Massive data sets).
Графові бази даних (Graph databases)	Ідеально підходять для сценаріїв використання, що включають зберігання та аналіз даних про сутності зі складними взаємозв'язками між собою.	**Особливість:** У графових базах даних однакова важливість надається як самим сутностям (вузлам), так і зв'язкам між ними.

Розробка гнучких фреймворків для міграції даних з реляційної бази даних до NoSQL-сховища даних – це ще одна галузь досліджень, яка отримала підвищений інтерес в останні роки. На карту поставлено багато факторів щодо успіху передачі даних з SQL до NoSQL. Це має вирішальне значення, оскільки будь-якій компанії необхідно отримувати корисну інформацію з власних даних, таку як використання системних ресурсів та оцінки ефективності роботи співробітників.

Крім того, NoSQL є кращим варіантом порівняно з реляційними базами даних у хмарі через розподілену природу хмарної платформи. Щоб зменшити

зусилля користувачів хмарних служб під час переміщення даних між багатьма хмарними платформами та контролювати проблеми взаємодії та переносимості даних, сучасний рівень розвитку вимагає уніфікованих фреймворків API [13].

Прогнозування та виникнення СУБД для конкретної СУБД

Наша система даних базується на таблиці . Спочатку ми створили пару даних (x, y). Потім ми використали формулу комбінації, щоб створити пару (x, y) для рядка з більш ніж двома іменами баз даних, таким чином побудувавши 301 пару. Ми використали кодувальник міток для кодування x та y у числа з рядка для попередньої електронної обробки та використали гауссівський наївний баєсівський алгоритм для навчання наших закодованих даних. Після того, як вони були підготовлені, ми отримали унікальне ім'я бази даних з x. Ми закодували ці унікальні імена, а потім протестували їх на окремих закодованих даних. Потім ми отримали ймовірність усіх класів, що відповідають окремим закодованим даним, і представили результат як унікальне закодоване x, що відповідає решті різних ймовірностей бази даних. Таким чином, розмір таблиці дорівнює n на n, де n - це унікальна база даних.

Аналіз зібраних даних: На зібраних даних ми навчили гауссову наївну баєсівську модель. Тим не менш, головна проблема полягає в тому, що вона передбачала MongoDB для всіх імен баз даних, окрім себе. MongoDB у наборі міток (y) багато порівняно з іншими. Показують, що відсоток MongoDB містить понад 22% усіх даних. Таким чином, модель має упередженість щодо даних. Щоб вирішити цю проблему дисбалансу даних, ми згенерували дані. Аналіз зібраних та згенерованих даних: на основі аналізу, , ми дійшли висновку, що бази даних "MongoDB, Cassandra, MySQL та HBase" зробили значний внесок у зібраний набір даних. Таким чином, ми застосували стратегію ігнорування цих імен баз даних під час створення комбінації інших імен баз даних. Відсоток віку цих імен баз даних потім можна зменшити, а ім'я іншої бази даних можна збільшити, що призведе до збалансування набору даних. Набір даних видається збалансованим, а результати відрізняються від тих, що ми отримали із зібраного набору даних.

Метою цієї роботи було порівняння реляційних та нереляційних баз даних.

Для цієї роботи було створено веб-застосунок для соціальних мереж. Застосунок використовувався для дослідження продуктивності вибраних баз даних.

Усі бази даних мають зручний інтерфейс для Java. Реалізація певних функцій, таких як пагінація та повнотекстовий пошук, є складнішою для Cassandra через те, що мова запитів не така багата, як інтерфейс доступу до даних SQL або MongoDB.

Для вибраних моделей даних результати показують переваги продуктивності нереляційних баз даних над реляційними. Для достатньо великих наборів кількість операцій, що виконуються реляційною базою даних, у кілька разів менша. MongoDB була найшвидшою базою даних у контексті читання. Тільки у випадку запису даних SQL був найшвидшим.

Статус реляційних баз даних на ринку не перебуває під загрозою, і важко уявити, що це скоро зміниться. NoSQL бази даних наразі все ще є новим і маловідомим рішенням. Однак подальший розвиток Інтернету та мобільних пристроїв змусить розробників програмного забезпечення все частіше використовувати NoSQL бази даних.

Наші плани на майбутнє охоплюють аналіз продуктивності прикладних NoSQL баз даних у BigData. Ця галузь швидко розвивається, і нещодавні дослідження показують, що ця тенденція є багатообіцяючою.

3.4 Висновок по розділу

У третьому розділі реалізовано програмне забезпечення з використанням реляційної бази даних та досліджено його ефективність. Розглянуто особливості трансляції й оптимізації реляційних запитів, що впливають на швидкодію та продуктивність системи. Розроблено тестовий програмний додаток, який демонструє практичне використання реляційної бази даних для зберігання та обробки інформації. Проведений аналіз результатів підтвердив коректність роботи системи, ефективність обраних методів проєктування та доцільність використання реляційних баз даних під час створення сучасного ПЗ.

ВИСНОВКИ

У ході виконання дипломної роботи було досліджено теоретичні та практичні аспекти розроблення програмного забезпечення з використанням реляційних баз даних. У роботі проаналізовано основні моделі даних, принципи побудови реляційних баз даних та сучасні підходи до проєктування програмних систем, що забезпечують ефективне зберігання й обробку інформації. Особливу увагу приділено питанням взаємодії програмного забезпечення з базами даних за допомогою SQL, а також методам оптимізації запитів для підвищення продуктивності систем.

На початковому етапі було проведено аналіз предметної області та визначено основні вимоги до програмного забезпечення. Розгляд моделей даних дозволив оцінити переваги реляційного підходу щодо забезпечення цілісності, узгодженості та структурованості інформації. Вивчення реляційної моделі даних стало основою для подальшого проєктування структури бази даних і реалізації програмного рішення.

У процесі роботи було досліджено методи взаємодії з реляційними базами даних, розглянуто можливості використання SQL для виконання операцій вибірки, додавання, оновлення та видалення даних. Також проаналізовано об'єктно-орієнтовані моделі даних та їх інтеграцію з реляційними базами даних. На основі концептуального та логічного проєктування було створено структуру бази даних, яка забезпечує ефективне зберігання та обробку інформації.

Практична частина роботи включала реалізацію тестового програмного додатка з використанням реляційної бази даних. У процесі реалізації було досліджено механізми трансляції та оптимізації реляційних запитів, що дозволило підвищити швидкодію системи та зменшити навантаження на базу даних. Проведене тестування підтвердило коректність функціонування програмного забезпечення та його відповідність поставленим вимогам.

Отримані результати свідчать про ефективність використання реляційних баз даних у сучасних програмних системах. Розроблене програмне забезпечення забезпечує надійне зберігання даних, підтримує їх цілісність та дозволяє

ефективно виконувати операції обробки інформації. Перевагами запропонованого рішення є структурованість даних, можливість масштабування та використання стандартних засобів керування базами даних.

У подальшому програмний продукт може бути вдосконалений шляхом розширення функціональних можливостей, впровадження додаткових механізмів безпеки, використання технологій розподіленого зберігання даних та застосування сучасних підходів до оптимізації продуктивності. Це дозволить підвищити ефективність роботи системи та розширити сферу її практичного застосування.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Date, C. J. (2019). An Introduction to Database Systems (8th ed.). Addison-Wesley.
2. Elmasri, R., & Navathe, S. B. (2021). Fundamentals of Database Systems (7th ed.). Pearson Education.
3. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). Database System Concepts (7th ed.). McGraw-Hill Education.
4. Coronel, C., & Morris, S. (2022). Database Systems: Design, Implementation, and Management (14th ed.). Cengage Learning.
5. Connolly, T., & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation and Management (6th ed.). Pearson.
6. Harrington, J. L. (2016). Relational Database Design and Implementation (4th ed.). Morgan Kaufmann.
7. Garcia-Molina, H., Ullman, J. D., & Widom, J. (2019). Database Systems: The Complete Book (2nd ed.). Pearson.
8. Ramakrishnan, R., & Gehrke, J. (2021). Database Management Systems (3rd ed.). McGraw-Hill.
9. Kleppmann, M. (2017). Designing Data-Intensive Applications. O'Reilly Media.
10. Fowler, M. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley.
11. Martin, R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson.
12. Sommerville, I. (2020). Software Engineering (10th ed.). Pearson Education.
13. Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner's Approach (9th ed.). McGraw-Hill.

14. Ambler, S. W. (2012). Agile Database Techniques: Effective Strategies for the Agile Software Developer. Wiley.

15. Celko, J. (2015). SQL for Smarties: Advanced SQL Programming (5th ed.). Morgan Kaufmann.

16. Beaulieu, A. (2020). Learning SQL (3rd ed.). O'Reilly Media.

17. Viescas, J. L., Hernandez, M. J., & Getz, D. (2018). SQL Queries for Mere Mortals (4th ed.). Addison-Wesley.

18. PostgreSQL Global Development Group. (2025). PostgreSQL Documentation. https://www.postgresql.org/docs/

19. Oracle Corporation. (2025). Oracle Database Documentation. https://docs.oracle.com/en/database/

20. Microsoft. (2025). SQL Server Documentation. https://learn.microsoft.com/en-us/sql/

21. MySQL Documentation Team. (2025). MySQL Reference Manual. https://dev.mysql.com/doc/

22. SQLite Documentation. (2025). https://www.sqlite.org/docs.html

23. IBM. (2025). Db2 Documentation. https://www.ibm.com/docs/en/db2

24. Microsoft. (2025). Entity Framework Core Documentation. https://learn.microsoft.com/en-us/ef/core/

25. Hibernate ORM Documentation. (2025). https://hibernate.org/orm/documentation/

26. Oracle. (2025). Java Persistence API Documentation. https://docs.oracle.com/javaee/

27. ISO/IEC 12207:2017. Systems and Software Engineering — Software Life Cycle Processes. International Organization for Standardization.

28. ISO/IEC 25010:2023. Systems and Software Quality Requirements and Evaluation (SQuaRE). International Organization for Standardization.

29. ISO/IEC 29148:2018. Systems and Software Engineering — Requirements Engineering. International Organization for Standardization.

30. OWASP Foundation. (2025). SQL Injection Prevention Cheat Sheet. https://cheatsheetseries.owasp.org/

31. Redgate Software. (2024). Database Design Best Practices. https://www.red-gate.com

32. Vertabelo Academy. (2024). Database Modeling and Design Fundamentals. https://academy.vertabelo.com

33. Oracle Academy. (2024). Database Design and Programming with SQL. https://academy.oracle.com

34. Hernandez, M. J. (2013). Database Design for Mere Mortals (3rd ed.). Addison-Wesley.

35. Teorey, T. J., Lightstone, S., Nadeau, T., & Jagadish, H. V. (2011). Database Modeling and Design (5th ed.). Morgan Kaufmann.

36. Kimball, R., & Ross, M. (2013). The Data Warehouse Toolkit (3rd ed.). Wiley.

37. Hellerstein, J. M., Stonebraker, M., & Hamilton, J. (2007). Architecture of a Database System. Foundations and Trends in Databases, 1(2), 141–259.

38. Chaudhuri, S. (1998). An Overview of Query Optimization in Relational Systems. Proceedings of PODS, 34–43.

39. Özsu, M. T., & Valduriez, P. (2020). Principles of Distributed Database Systems (4th ed.). Springer.

40. Shasha, D., & Bonnet, P. (2019). Database Tuning: Principles, Experiments, and Troubleshooting Techniques. Morgan Kaufmann.

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: "Розроблення програмного забезпечення з використанням реляційних баз даних "

Обсяг пояснювальної записки: 85 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____