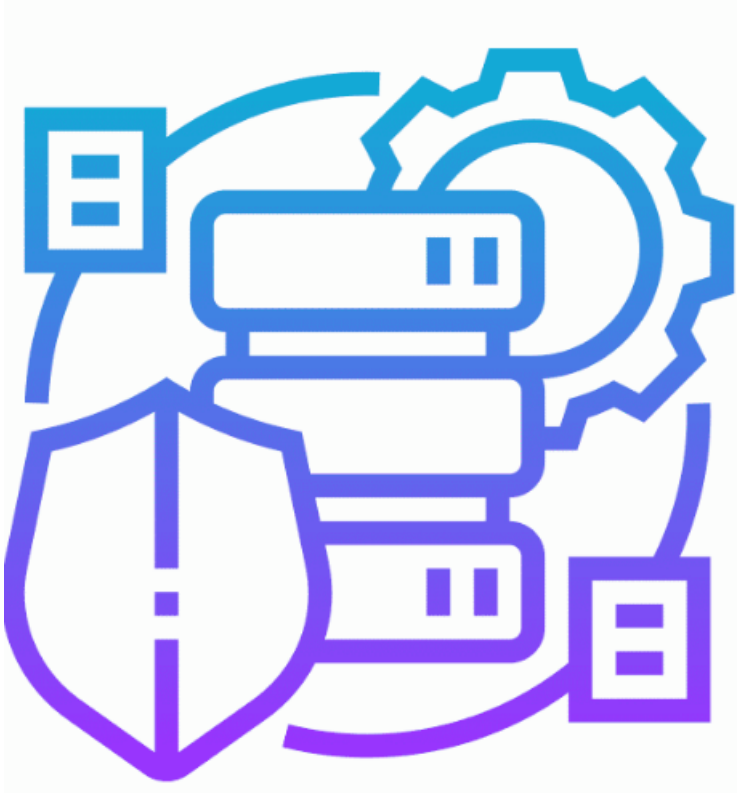
Чернівецький національний університет імені Юрія Федьковича
Факультет математики та інформатики

Мельник В. С., Мельник Г. В.

«Сучасні системи управління базами даних»

Навчальний посібник



Чернівці, 2025

Друкується за ухвалою редакційно-видавничої ради
Чернівецького національного університету
імені Юрія Федьковича

С-753 Сучасні системи управління базами даних: *Навчальний посібник / Укл. Мельник В. С., Мельник Г. В.* – Чернівці: ЧНУ, 2025. – 108 с.

Навчальний посібник містить теоретичний матеріал та методичні рекомендації для виконання лабораторних та практичних завдань з курсу Сучасні системи управління базами даних».

Для студентів факультету математики та інформатики

Зміст

Передмова

У сучасному цифровому світі, що стрімко розвивається, ефективна робота з даними стала невід'ємною складовою практично будь-якої галузі. Обсяг, складність і динаміка даних зростають, і це вимагає від фахівців глибокого розуміння принципів зберігання, обробки та керування даними. Саме тому вивчення дисципліни **«Сучасні системи управління базами даних»** є ключовим елементом підготовки майбутніх прикладних математиків, інженерів та аналітиків.

Цей навчальний посібник охоплює як **теоретичні аспекти побудови та функціонування реляційних і нереляційних систем керування базами даних**, так і **практичні навички**, що необхідні для проєктування, реалізації та супроводу баз даних у реальних додатках.

Структура посібника побудована відповідно до логіки вивчення дисципліни:

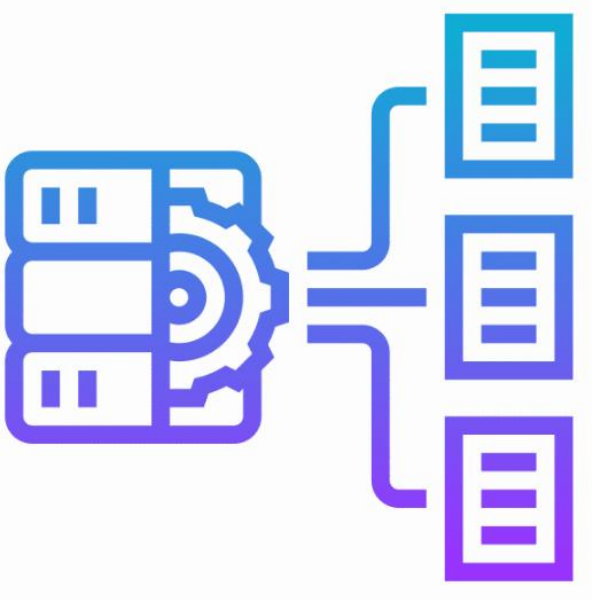
- від основ синтаксису SQL і побудови запитів;
- через формування складних реляційних структур, роботу з транзакціями та тригерами;
- до ознайомлення з нереляційними моделями даних та використання MongoDB.

Матеріали посібника доповнено **лабораторними роботами**, прикладами запитів, питаннями для самоперевірки, короткими довідковими вставками та завданнями для самостійної роботи. Особлива увага приділена застосуванню мови SQL у середовищі **MS SQL Server**, а також використанню **моделі клієнт/сервер**, що відповідає практикам сучасного програмного забезпечення.

Автори сподіваються, що цей посібник стане для студентів не лише інструментом для засвоєння курсу, а й надійним помічником у подальшому професійному розвитку.

Укладачі:
к.е.н., доц. Г.В. Мельник, к.ф.-м.н., В.С.Мельник
Чернівці, 2025

Змістовий модуль 1: Засоби SQL-доступу до даних



Тема 1. Вступ до SQL. Основні поняття та терміни

Мета теми
Ознайомити студентів з історією виникнення мови SQL, її основними компонентами, категоріями операторів, а також із поняттям реляційної моделі й типовими типами даних.

1.1. Історія SQL

Походження: теоретичне підґрунтя
У 1970 році британський науковець **Едгар Френк Кодд**, співробітник IBM, опублікував статтю *“A Relational Model of Data for Large Shared Data Banks”* у журналі *Communications of the ACM*. У цій праці було представлено **реляційну модель** даних — математичний підхід до організації інформації у вигляді таблиць (реляцій), що опирався на теорію множин і логіку предикатів.

Від SEQUEL до SQL
- У середині 1970-х років у **лабораторії San Jose Research Laboratory IBM** почалася реалізація першої реляційної СУБД під назвою **System R**.
- Для взаємодії з цією системою розробили мову **SEQUEL** (*Structured English Query Language*), призначену для «англомовного» формулювання запитів до таблиць.
- Згодом назву скоротили до **SQL**, оскільки «SEQUEL» вже була торговою маркою авіакомпанії Hawker Siddeley.

Перші реалізації

- **1979** — компанія **Relational Software Inc.** (пізніше перейменована в **Oracle Corporation**) випустила першу комерційну СУБД, що підтримувала SQL: **Oracle v2**.
- **1981** — IBM представила свою версію: **SQL/DS**, а згодом і **DB2** (1983), що стали класикою корпоративного сегменту.

Стандартизація SQL
Масове впровадження SQL стимулювало потребу в стандартизації:
- **1986** — ANSI ухвалює перший стандарт SQL як **SQL-86**
- **1987** — ISO підтримує той самий стандарт
- Подальші ключові версії:\n
  - **SQL-89** — мінімальні уточнення
  - **SQL-92** — масштабне розширення (також званий SQL2)
  - **SQL:1999** — об'єктно-орієнтовані розширення, тригери, рекурсія
  - **SQL:2003** — XML-підтримка, Window Functions
  - **SQL:2008** — підтримка MERGE, розширення TRUNCATE
  - **SQL:2011** — підтримка temporal tables
  - **SQL:2016–2023** — JSON, polymorphic tables, нові типи валідації

Примітка: не всі реалізації підтримують повний стандарт, тому діалекти SQL різняться (наприклад, T-SQL у MS SQL Server, PL/pgSQL у PostgreSQL, PL/SQL в Oracle).

Цікаві факти
- Oracle назвала свою першу версію **v2** (а не v1), щоб створити ілюзію стабільного продукту.
- SQL не є чисто декларативною чи процедурною мовою — вона **гібридна**.
- Хоч SQL був розроблений у IBM, першою комерційною реалізацією став продукт конкурентів — Oracle.

1.2. Стандарти SQL
Існує декілька версій (стандартів) мови SQL, затверджених ANSI/ISO:
- **SQL-86 / SQL-89** — базові можливості (SELECT, INSERT, DELETE).
- **SQL-92** — основа для більшості сучасних реалізацій (JOIN, підзапити, транзакції).
- **SQL:1999 (SQL-99)** — введено тригери, об’єктно-орієнтовані можливості, рекурсивні запити.
- **SQL:2003** — підтримка XML, автоматичні послідовності (SEQUENCE).
- **SQL:2011, 2016, 2019** — додаткові функції JSON, аналізу вікон (window functions), time zones.

СУБД можуть реалізовувати різні частини стандарту, тому SQL в Oracle, MySQL, SQL Server має нюанси.

1.3. Категорії команд SQL
SQL поділяється на такі основні категорії:

Категорія	Назва	Призначення	Приклад
DDL	Data Definition Language	Визначення структури БД	CREATE TABLE, ALTER, DROP
DML	Data Manipulation Language	Робота з даними	SELECT, INSERT, UPDATE, DELETE
DCL	Data Control Language	Керування правами	GRANT, REVOKE

Категорія	Назва	Призначення	Приклад
TCL	Transaction Control Language	Керування транзакціями	COMMIT, ROLLBACK, SAVEPOINT

1.4. Типи даних у SQL

У мові SQL типи даних визначають, який тип значень може зберігатися в кожному стовпці таблиці. Вони критично важливі для забезпечення цілісності даних, правильного виконання запитів та ефективності зберігання.

SQL підтримує кілька категорій типів даних: числові, символьні (рядкові), типи для представлення дати й часу, логічні, а також спеціалізовані типи, які залежать від конкретної СУБД (наприклад, типи для зображень, JSON або географічних даних).

Числові типи

Числові типи використовуються для зберігання цілих чисел або чисел з плаваючою точкою.

Найпоширеніші:

- INT, INTEGER — стандартні типи для зберігання цілих чисел. Їх обсяг залежить від конкретної СУБД, але зазвичай охоплює діапазон від −2,147,483,648 до 2,147,483,647.
- SMALLINT — для менших цілих чисел. Займає менше місця, що важливо при роботі з великими таблицями.
- DECIMAL(p,s) або NUMERIC(p,s) — точні числа з фіксованою кількістю знаків після коми. Наприклад, DECIMAL(6,2) дозволяє зберігати число до 9999.99. Ці типи ідеально підходять для фінансових даних.
- FLOAT, REAL, DOUBLE PRECISION — числа з плаваючою точкою. Вони менш точні, ніж DECIMAL, але займають менше місця. Добре підходять для наукових розрахунків або приблизних значень.

Рядкові типи

Рядкові типи дозволяють зберігати текстову інформацію, наприклад імена, адреси, коментарі.

- CHAR(n) — рядок фіксованої довжини n. Якщо запис коротший, система доповнює його пробілами. Використовується для зберігання даних однакової довжини, наприклад, поштових індексів або кодів країн.
- VARCHAR(n) — рядок змінної довжини, де n — максимальна кількість символів. Цей тип зберігає лише фактичну довжину введеного рядка, тому є більш гнучким і економним у використанні пам’яті.
- TEXT — використовується для зберігання довгих текстів (описів, повідомлень тощо). Зазвичай не має чіткого обмеження довжини, але має обмеження на використання в індексах і обчислюваних полях у багатьох СУБД.

Типи дати й часу

Типи дати та часу дозволяють працювати з календарними і часовими значеннями, що особливо актуально в транзакційних та логістичних системах.

- DATE — представляє тільки дату (рік, місяць, день).
- TIME — лише час (години, хвилини, секунди).
- DATETIME або TIMESTAMP — поєднує дату і час. Часто використовується для фіксації моментів подій, створення записів, оновлень тощо.

У деяких СУБД TIMESTAMP додатково може мати функцію автоматичного оновлення значення при зміні запису.

Логічний тип

- BOOLEAN — зберігає логічне значення TRUE або FALSE.

- У багатьох реалізаціях SQL (наприклад, MySQL) цей тип фактично зберігається як TINYINT, де 0 — FALSE, а 1 — TRUE.

У зв'язку з відмінностями між СУБД, один і той самий тип може поводитися по-різному в PostgreSQL, MySQL, SQL Server, Oracle тощо. Наприклад, підтримка BOOLEAN у Oracle реалізується через NUMBER(1) або псевдологіку з CHAR(1).
Таким чином, вибір типів даних — це не лише питання точності, а й продуктивності, масштабованості та відповідності бізнес-вимогам.

1.5. Реляційна модель даних

Реляційна модель — це фундаментальна концепція, яка визначає, як організовуються і взаємодіють дані в сучасних базах даних. Її було запропоновано Едгаром Ф. Коддом у 1970 році, і вона стала основою для побудови більшості СУБД, які використовуються сьогодні.
У реляційній моделі інформація зберігається у вигляді таблиць, які ще називають реляціями. Кожна таблиця описує певну сутність предметної області — наприклад, клієнтів, замовлення або товари.
Кожна таблиця складається з:

- **Стовпців (атрибутів)** — кожен з яких визначає певну характеристику об'єкта, наприклад, ім’я клієнта, дату замовлення чи кількість товару.
- **Рядків (кортежів)** — кожен з яких відповідає одному запису, тобто окремому об’єкту або факту.

Унікальність кожного запису забезпечується за допомогою **ключа** — атрибуту або комбінації атрибутів, що однозначно ідентифікує кожний рядок у таблиці.
Первинний ключ (PRIMARY KEY) Це один або кілька атрибутів, значення яких унікальні для кожного запису. Наприклад, у таблиці Customers первинним ключем може бути CustomerID.
Вимоги до первинного ключа:

- не може містити NULL;
- повинен бути унікальним у межах таблиці;
- один на таблицю.

Зовнішній ключ (FOREIGN KEY) Це атрибут або набір атрибутів, що посилаються на первинний ключ іншої таблиці. Вони встановлюють зв’язки між таблицями.
Наприклад, у таблиці Orders поле CustomerID може бути зовнішнім ключем, який посилається на CustomerID у таблиці Customers.
Завдяки зовнішнім ключам забезпечується цілісність зв’язків між даними — кожне замовлення пов’язане з існуючим клієнтом.
Цілісність даних (referential integrity) Це набір правил, що гарантує логічну узгодженість даних. Основний принцип: жодне посилання з однієї таблиці на іншу не повинно «вказувати в порожнечу». Тобто не можна, щоб у полі зовнішнього ключа містився ідентифікатор, якого немає у відповідній таблиці.
Цілісність підтримується механізмами СУБД автоматично, якщо ключі визначені правильно. При спробі вставити або видалити дані, які порушують зв’язки, СУБД згенерує помилку або (залежно від налаштувань) автоматично оновить/видалить пов’язані записи.
Таким чином, реляційна модель забезпечує логічну, математично обґрунтовану структуру для зберігання та обробки інформації, де таблиці — не просто сховища даних, а структуровані об’єкти з чіткими зв’язками, правилами та гарантіями.

Контрольні запитання для самоперевірки

1. Яка історія виникнення SQL?
2. Які стандарти SQL вам відомі? Які нововведення в SQL:1999?
3. Назвіть і поясніть призначення категорій операторів SQL.
4. Які типи даних найчастіше використовуються в SQL?
5. Що таке реляційна модель? Які її ключові особливості?

Приклад на повторення

```
-- Створення таблиці Employees
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    BirthDate DATE,
    Salary DECIMAL(10, 2)
);
```

Словник термінів до Теми 1

Термін	Визначення
SQL	Мова структурованих запитів (Structured Query Language) — стандартна мова для роботи з реляційними базами даних.
СУБД (DBMS)	Система управління базами даних — програмне забезпечення для створення, зберігання та керування базами даних.
Реляційна модель	Модель зберігання даних у вигляді таблиць (реляцій), де кожна таблиця має унікальний ключ.
Таблиця (table)	Основна структура зберігання даних у реляційній БД. Складається з рядків і стовпців.
Поле (field/column)	Стовпець у таблиці, який описує одну з характеристик запису (наприклад, ім’я, дата народження).
Рядок (row/record)	Один запис у таблиці, що містить дані про конкретний об'єкт.
Ключ (key)	Атрибут або набір атрибутів, що однозначно ідентифікує запис у таблиці.
Первинний ключ (Primary Key)	Унікальний ідентифікатор кожного запису в таблиці.
Зовнішній ключ (Foreign Key)	Поле, яке встановлює зв’язок між двома таблицями.
DDL	Data Definition Language — підмова SQL для створення, зміни і видалення об'єктів БД (таблиць, схем).
DML	Data Manipulation Language — підмова SQL для обробки даних (запити, вставка, оновлення, видалення).
DCL	Data Control Language — підмова SQL для керування доступом до об'єктів БД.

Термін	Визначення
TCL	Transaction Control Language — команди керування транзакціями (COMMIT, ROLLBACK).
NULL	Значення "невідомо" або "відсутнє значення". Не дорівнює нулю або порожньому рядку.

Тема 2. SELECT-запити до однієї таблиці

Мета теми
Опанувати базовий синтаксис SQL-запитів до однієї таблиці для вибірки, фільтрації, сортування та обчислення значень на основі даних.

Оператор SELECT є центральним елементом мови SQL. Саме за його допомогою користувачі отримують дані з бази у зручному табличному вигляді. Незалежно від розміру бази чи складності структури, запити на вибірку даних майже завжди починаються саме з SELECT.
Оператор SELECT дозволяє:

- визначити, які саме стовпці показати (SELECT column1, column2)
- відображати всі стовпці (SELECT *)
- застосовувати фільтри за допомогою WHERE
- сортувати результати через ORDER BY
- групувати дані з використанням GROUP BY
- використовувати агрегатні функції (наприклад, COUNT, AVG, MAX)
- здійснювати обчислення прямо в запиті (наприклад, SELECT price * quantity AS total_price)
- поєднувати дані з кількох таблиць через JOIN
- обмежувати кількість рядків (через LIMIT, TOP або FETCH FIRST, залежно від СУБД)

Важливо розуміти, що SELECT — декларативна конструкція. Це означає, що користувач описує, *які* дані він хоче отримати, а не *як* їх шукати. Реалізація ефективного пошуку — завдання СУБД.
У складніших випадках SELECT може включати вкладені запити (subqueries), об’єднання кількох результатів (UNION), умовні конструкції (CASE), або роботу з віконними функціями (OVER()).
Таким чином, SELECT — це не просто інструмент перегляду даних. Це універсальний механізм доступу до інформації, що поєднує логіку відбору, сортування, групування та аналітики в одному запиті. Навчитися ефективно працювати з ним — ключовий крок у використанні SQL.

2.1. Синтаксис SELECT

Базовий синтаксис SQL-запиту до однієї таблиці має вигляд:

```
SELECT [поля]
FROM [таблиця]
WHERE [умови]
ORDER BY [поля];
```

- SELECT — визначає, які поля буде виведено
- FROM — визначає таблицю, з якої здійснюється вибірка
- WHERE — (необов'язково) вказує умови фільтрації
- ORDER BY — (необов'язково) визначає порядок сортування

2.2. Оператори в WHERE

Умови порівняння:

```
=, <>, >, <, >=, <=
```

Приклади:

```
SELECT * FROM Employees WHERE Age > 40
```

Оператор BETWEEN:

```
SELECT * FROM Products WHERE Price BETWEEN 10 AND 50
```

Оператор IN:

```
SELECT * FROM Customers WHERE City IN ('Paris', 'London',
'Rome');
```

Оператор LIKE:

- _ — один символ
- % — будь-яка кількість символів

```
SELECT * FROM Employees WHERE FirstName LIKE 'A%';
```

Оператор IS NULL:

```
SELECT * FROM Orders WHERE ShippedDate IS NULL;
```

2.3. Обчислення в запитах

Можна виконувати обчислення прямо в запиті:

```
SELECT Name, Price, Price * 1.2 AS PriceWithTax FROM Products
```

2.4. Сортування результатів

```
ORDER BY [поле] [ASC | DESC]
```

Приклади:

```
SELECT * FROM Employees ORDER BY LastName;
SELECT * FROM Products ORDER BY Price DESC;
```

2.5. Вирази CASE

Вираз CASE дозволяє реалізувати умовну логіку в запиті:

```
SELECT Name,
       CASE
           WHEN Price < 50 THEN 'Cheap'
           WHEN Price BETWEEN 50 AND 100 THEN 'Medium'
           ELSE 'Expensive'
       END AS PriceCategory
FROM Products;
```

Контрольні запитання для самоперевірки

1. Який базовий синтаксис SELECT-запиту?
2. Чим відрізняються BETWEEN, IN та LIKE?
3. Як за допомогою SQL відсортувати дані за спаданням?
4. Наведіть приклад використання CASE у запиті.
5. Яке значення повертає IS NULL?

Приклади

```
-- Вивести імена співробітників із Лондона
SELECT FirstName, LastName FROM Employees WHERE City =
'London';

-- Імена співробітників, чиє ім’я починається на A
SELECT FirstName, LastName FROM Employees WHERE FirstName LIKE
'A%';

-- Імена й вік співробітників віком понад 55, впорядкувати за
прізвищем
SELECT FirstName, LastName, Age FROM Employees
WHERE Age > 55
ORDER BY LastName;
```

Словник термінів до Теми 2

Термін	Пояснення
SELECT	Оператор вибірки даних з таблиці
WHERE	Умова фільтрації результатів
ORDER BY	Оператор сортування результатів
LIKE	Оператор пошуку за шаблоном

Термін	Пояснення
IN	Оператор перевірки належності значення списку
BETWEEN	Оператор для перевірки входження в інтервал
IS NULL	Перевірка на відсутність значення
CASE	Умовний вираз (аналог if...else)
AS	Псевдонім для поля або виразу

Тема 3. Агрегування та групування даних

Мета теми

Опанувати механізми виконання узагальнюючих запитів з використанням агрегатних функцій та операторів GROUP BY і HAVING для аналітики даних.

3.1. Навіщо потрібна агрегація та групування?

У реляційних базах даних зберігається велика кількість детальних записів: кожен рядок — це окремий факт (транзакція, користувач, замовлення тощо). Проте часто нас цікавить не кожен окремий запис, а узагальнена інформація: скільки всього було продажів за місяць, середня зарплата по відділах, максимальна кількість замовлень клієнтом тощо.

У таких випадках застосовуються **агрегація** та **групування** — два тісно пов'язані механізми, які дозволяють виконувати обчислення на основі множини записів.

Агрегація — це обчислення узагальненої характеристики для набору значень. SQL надає набір **агрегатних функцій**, які застосовуються до одного або кількох стовпців:

Якщо потрібно виконати агрегацію не по всій таблиці, а **за категоріями**, використовується конструкція GROUP BY. Вона групує записи за значенням одного або кількох атрибутів, і тоді агрегатні функції застосовуються **до кожної групи окремо**.

Поєднання з фільтрацією

Існує два способи обмежити результати при агрегації:

- WHERE — фільтрує **рядки до агрегації** (наприклад, брати лише активних клієнтів).
- HAVING — фільтрує **групи після агрегації** (наприклад, показати тільки ті департаменти, де середня зарплата > 50000).

Практична користь

Групування й агрегація використовуються в аналітичній звітності, бізнес-інтелекті, фінансовій звітності, контролі якості, управлінні персоналом тощо. Вони дозволяють узагальнити інформацію, виявити тенденції, порівнювати ефективність, визначати аномалії.

Наприклад:

- скільки замовлень зробив кожен клієнт;
- середній чек по містах;
- максимальна кількість товарів в одному замовленні;
- департаменти з найвищим рівнем продуктивності.

Таким чином, *агрегація та групування є ключовими інструментами перетворення "сирих" даних у змістовну аналітику.*

3.2. Агрегатні функції в SQL

Агрегатні функції виконують обчислення над **групами рядків** (або над усією таблицею, якщо GROUP BY не використовується):

Функція	Опис
COUNT(*)	Підраховує кількість рядків
COUNT(поле)	Підраховує кількість непорожніх значень у полі
SUM(поле)	Обчислює суму значень
AVG(поле)	Обчислює середнє значення
MIN(поле)	Повертає мінімальне значення
MAX(поле)	Повертає максимальне значення

Приклад:

```
SELECT COUNT(*) FROM Employees;
SELECT AVG(Salary) FROM Employees;
```

3.3. Групування результатів: GROUP BY

GROUP BY дозволяє об'єднати записи у **групи** за певною ознакою та застосувати до кожної групи агрегатну функцію.

Приклад:

```
SELECT City, COUNT(*) AS NumEmployees
FROM Employees
GROUP BY City;
```

Це поверне кількість працівників у кожному місті.

Інший приклад:

```
SELECT CategoryID, AVG(UnitPrice) AS AvgPrice
FROM Products
GROUP BY CategoryID;
```

3.4. Умови до груп: HAVING

HAVING використовується для **фільтрації результатів після групування**, подібно до WHERE, але застосовується до **агрегатів**.

Приклад:

```
SELECT City, COUNT(*) AS NumCustomers
FROM Customers
GROUP BY City
HAVING COUNT(*) > 3;
```

Повертає лише ті міста, в яких більше трьох клієнтів.

Важливо: не можна використовувати WHERE COUNT(*) > 3, бо WHERE працює **до** агрегування, а HAVING — **після**.

3.5. Порядок виконання SQL-запиту з групуванням

1. FROM — вибір таблиці
2. WHERE — фільтрація рядків
3. GROUP BY — групування рядків
4. **Агрегатні функції** — обчислення над групами
5. HAVING — фільтрація груп
6. SELECT — вибірка фінальних значень
7. ORDER BY — сортування результатів

3.6. Приклади

Порахувати кількість замовлень по кожному клієнту:

```
SELECT CustomerID, COUNT(*) AS NumOrders
FROM Orders
GROUP BY CustomerID;
```

Порахувати середню ціну по категоріях товарів:

```
SELECT CategoryID, AVG(UnitPrice) AS AvgPrice
FROM Products
GROUP BY CategoryID
HAVING AVG(UnitPrice) > 50;
```

Вивести країни, де було зроблено більше 100 замовлень:

```
SELECT Country, COUNT(*) AS NumOrders
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY Country
HAVING COUNT(*) > 100;
```

Контрольні запитання для самоперевірки

1. Які є агрегатні функції в SQL?
2. Для чого використовується GROUP BY?
3. Чим відрізняється HAVING від WHERE?
4. Чи можна використати ORDER BY разом із GROUP BY?
5. Який порядок виконання SQL-запиту з агрегацією?

Словник термінів до Теми 3

Термін	Пояснення
AGGREGATE FUNCTION	Функція, яка обчислює значення на основі групи рядків
GROUP BY	Оператор групування даних за значеннями одного чи кількох полів
HAVING	Умова для фільтрації вже агрегованих результатів
COUNT(*)	Кількість усіх рядків
SUM()	Сума всіх значень у полі
AVG()	Середнє значення
MIN() / MAX()	Мінімальне / максимальне значення у групі
AS	Псевдонім для колонки або виразу
ORDER BY	Оператор сортування після агрегування

Практичні завдання

Примітка: *використовуйте навчальну базу даних Northwind або її аналог із таблицями Employees, Orders, Products, Customers, Categories, OrderDetails.*

Рівень 1: Базова агрегація

1. **Порахуйте загальну кількість співробітників.**

```
SELECT COUNT(*) FROM Employees;
```

2. **Знайдіть середній вік співробітників.**

Використовуйте поле BirthDate, обчисліть вік за допомогою DATEDIFF або YEAR(CURRENT_DATE) - YEAR(BirthDate) (залежно від СУБД).

3. **Виведіть максимальну і мінімальну зарплату серед співробітників.**

```
SELECT MAX(Salary), MIN(Salary) FROM Employees;
```

Рівень 2: Групування та агрегація

4. **Порахуйте кількість клієнтів у кожному місті.**

```
SELECT City, COUNT(*) AS NumCustomers
FROM Customers
GROUP BY City;
```

5. **Знайдіть середню вартість товарів у кожній категорії.**

```
SELECT CategoryID, AVG(UnitPrice) AS AvgPrice
FROM Products
GROUP BY CategoryID;
```

6. **Порахуйте кількість замовлень, виконаних кожним співробітником.**

```
SELECT EmployeeID, COUNT(*) AS NumOrders
FROM Orders
GROUP BY EmployeeID;
```

7. **Знайдіть країни, з яких є більше ніж 5 клієнтів.**

```
SELECT Country, COUNT(*) AS NumClients
FROM Customers
GROUP BY Country
HAVING COUNT(*) > 5;
```

Рівень 3: Умовне групування з HAVING

8. **Виведіть продукти, де середня ціна перевищує 40.**

```
SELECT CategoryID, AVG(UnitPrice) AS AvgPrice
FROM Products
GROUP BY CategoryID
HAVING AVG(UnitPrice) > 40;
```

9. **Покажіть співробітників, які обробили більше 100 замовлень.**

```
SELECT EmployeeID, COUNT(*) AS NumOrders
FROM Orders
GROUP BY EmployeeID
HAVING COUNT(*) > 100;
```

10. **Порахуйте кількість замовлень за кожним роком.**

Передбачає використання функції YEAR(OrderDate):

```
SELECT YEAR(OrderDate) AS OrderYear, COUNT(*) AS NumOrders
FROM Orders
GROUP BY YEAR(OrderDate);
```

Завдання ускладнені

11. **Знайдіть категорію товарів, у якій загальна сума замовлень найбільша.**

Потрібне приєднання Products, OrderDetails, агрегація:

```
SELECT p.CategoryID, SUM(od.Quantity * od.UnitPrice) AS
TotalSales
FROM OrderDetails od
JOIN Products p ON od.ProductID = p.ProductID
GROUP BY p.CategoryID
ORDER BY TotalSales DESC
LIMIT 1;
```

12. **Знайдіть клієнтів, які зробили замовлення на суму понад 5000.**

Потребує приєднання Orders, OrderDetails і Customers, з GROUP BY CustomerID + HAVING.

Індивідуальне завдання (на вибір)

13. Напишіть власний запит, що:

- групує дані за полем вашого вибору;
- містить принаймні одну агрегатну функцію;
- використовує HAVING для фільтрації результатів.

Тести для самоконтролю

1. Яка функція підраховує кількість усіх записів у таблиці?
   - SUM(*)
   - COUNT(*)
   - TOTAL()
   - NUMBER()

2. Який оператор використовується для групування даних?
   - ORDER BY
   - GROUP
   - GROUP BY
   - WHERE GROUP

3. Яка функція дозволяє знайти середнє значення у колонці?

- SUM()
- AVG()
- COUNT()
- MEAN()

4. Яке ключове слово дозволяє фільтрувати групи після агрегації?
   - WHERE
   - GROUP
   - FILTER
   - HAVING

5. Який запит покаже кількість замовлень у кожного клієнта?
   - SELECT CustomerID FROM Orders WHERE COUNT(*) > 0;
   - SELECT COUNT(CustomerID) FROM Orders GROUP BY CustomerID;
   - SELECT CustomerID, COUNT(*) FROM Orders GROUP BY CustomerID;
   - SELECT * FROM Orders GROUP CustomerID;

6. Яка з наведених функцій є агрегатною?
   - IF()
   - CASE
   - MAX()
   - ROUND()

7. Що поверне запит: SELECT COUNT(Salary) FROM Employees?
   - Загальну кількість співробітників
   - Суму зарплат
   - Кількість зарплат, відмінних від NULL
   - Максимальну зарплату

8. Чим відрізняються WHERE та HAVING?
   - Обидва фільтрують до агрегування
   - HAVING використовується лише з JOIN
   - WHERE — після GROUP BY, HAVING — до
   - WHERE — до GROUP BY, HAVING — після

9. Який запит покаже середню ціну товарів у кожній категорії, але тільки тих, де середня ціна > 100?

10. Що виведе запит SELECT Country, COUNT(*) FROM Customers GROUP BY Country HAVING COUNT(*) > 5?
   - Клієнтів, що проживають у країнах
   - Список усіх країн
   - Лише ті країни, де більше 5 клієнтів
   - Помилку

Ключі до тестів
1. Яка функція підраховує кількість усіх записів у таблиці?
Відповідь: **`COUNT(*)`**
2. Який оператор використовується для групування даних?
Відповідь: **`GROUP BY`**

3. Яка функція дозволяє знайти середнє значення у колонці?
Відповідь: **AVG()**
4. Яке ключове слово дозволяє фільтрувати групи після агрегації?
Відповідь: **HAVING**
5. Який запит покаже кількість замовлень у кожного клієнта?
Відповідь:
```
SELECT CustomerID, COUNT(*) FROM Orders GROUP BY CustomerID;
```
6. Яка з наведених функцій є агрегатною?
Відповідь: **MAX()**
7. Що поверне запит: SELECT COUNT(Salary) FROM Employees?
Відповідь: Кількість співробітників, у яких зарплата не є NULL
8. Чим відрізняються WHERE та HAVING?
Відповідь: WHERE — фільтрує рядки до агрегування, HAVING — після
9. Який запит покаже середню ціну товарів у кожній категорії, але тільки тих, де середня ціна > 100?
Відповідь:
```
SELECT CategoryID, AVG(UnitPrice)
FROM Products
GROUP BY CategoryID
HAVING AVG(UnitPrice) > 100;
```
10. Що виведе запит SELECT Country, COUNT(*) FROM Customers GROUP BY Country HAVING COUNT(*) > 5?
Відповідь: Країни, у яких більше 5 клієнтів

Тема 4. З’єднання таблиць (JOIN-запити)

Мета теми
Опанувати синтаксис і логіку виконання SQL-запитів, що об'єднують дані з кількох таблиць. Розуміти відмінності між типами з'єднань і знати, коли і яке з'єднання слід застосовувати.

4.1. Навіщо потрібні з’єднання таблиць?

У реляційній моделі бази даних інформація зберігається **нормалізовано**, тобто розділена на кілька таблиць, кожна з яких описує окрему сутність: наприклад, клієнти, замовлення, товари. Такий підхід дозволяє уникнути дублювання даних, спростити оновлення й забезпечити цілісність.
Проте в реальному житті майже ніколи не буває достатньо даних з однієї таблиці. Наприклад:

- Щоб побачити ім’я клієнта разом із його замовленням — потрібно звернутися до двох таблиць: customers і orders.
- Щоб показати товар, який входить до конкретного замовлення — потрібна інформація з orders, order_items і products.

Тут на допомогу приходять **з’єднання таблиць (JOINs)**.
З’єднання дозволяють **об'єднати записи з кількох таблиць** на основі логічного зв’язку — зазвичай через зовнішній ключ.

Приклад потреби в з’єднанні
Припустимо, у нас є дві таблиці:

- customers(customer_id, name)

- orders(order_id, customer_id, order_date)

Щоб отримати список замовлень із іменами клієнтів, недостатньо просто вибрати з orders. Потрібно поєднати orders.customer_id з customers.customer_id.

```
SELECT orders.order_id, customers.name, orders.order_date
FROM orders
JOIN customers ON orders.customer_id = customers.customer_id;
```

Навіщо це важливо

1. **Відображення зв'язків між сутностями**
2. З'єднання дозволяють об'єднати логічно пов'язані дані (наприклад, який клієнт замовив який товар).
3. **Уникнення дублювання**
4. Замість того, щоб зберігати ім'я клієнта в кожному рядку замовлення, ми зберігаємо лише ідентифікатор і при потребі з'єднуємо таблиці.
5. **Аналітика та звітність**
6. Наприклад, якщо потрібно порахувати суму замовлень по кожному клієнту, необхідно поєднати таблиці customers, orders і, можливо, order_items.
7. **Масштабованість**
8. При грамотній нормалізації і використанні з'єднань можна працювати з великими обсягами даних без дублювання, підтримуючи гнучкість і логічну структуру.
9. **Гнучкість запитів**
10. JOIN дозволяє комбінувати довільні поля з кількох таблиць, створюючи потрібний вигляд даних без змін самої структури бази.

Типовий підхід у SQL

- З'єднання виконується через JOIN, вказуючи умову через ON.
- Найчастіше з'єднання базується на співпадінні первинного та зовнішнього ключа.
- З'єднані таблиці можна використовувати як єдину віртуальну таблицю — фільтрувати, сортувати, групувати тощо.

Таким чином, **з'єднання таблиць — ключовий механізм роботи з нормалізованими базами даних**, який дозволяє отримати цілісну картину з розподілених фрагментів інформації.

4.2. Види з'єднань у SQL

INNER JOIN

Повертає тільки ті записи, для яких є відповідність у **обох** таблицях.

```
SELECT e.FirstName, o.OrderID
FROM Employees e
INNER JOIN Orders o ON e.EmployeeID = o.EmployeeID;
```

Виведе лише ті замовлення, які мають призначеного співробітника.

LEFT JOIN (LEFT OUTER JOIN)

Повертає **всі записи з лівої таблиці**, навіть якщо немає відповідності у правій.

```
SELECT c.CustomerID, o.OrderID
```

```
FROM Customers c
LEFT JOIN Orders o ON c.CustomerID = o.CustomerID;
```

*Показує всіх клієнтів, включно з тими, які **не робили жодного замовлення** (OrderID буде NULL).*

RIGHT JOIN (RIGHT OUTER JOIN)

Аналог LEFT JOIN, але навпаки — повертає всі записи з **правої таблиці**.

Не всі СУБД підтримують RIGHT JOIN (наприклад, SQLite — ні).

FULL JOIN (FULL OUTER JOIN)

Повертає всі записи з обох таблиць: якщо немає відповідності — підставляє NULL.

```
SELECT a.Name, b.Value
FROM TableA a
FULL OUTER JOIN TableB b ON a.ID = b.ID;
```

CROSS JOIN

Повертає **декартовий добуток**: кожен рядок з першої таблиці комбінується з кожним із другої.

```
SELECT *
FROM Sizes
CROSS JOIN Colors;
```

Часто використовується для генерації всіх комбінацій.

4.3. Як виконується JOIN-запит?

Порядок виконання:

1. FROM — обирається базова таблиця
2. JOIN — з'єднуються таблиці за умови (ON)
3. WHERE — фільтрація записів
4. SELECT — вибірка потрібних стовпців
5. ORDER BY — сортування

4.4. Практичні приклади JOIN-запитів

Вивести імена співробітників та ідентифікатори замовлень, які вони обробляли:

```
SELECT e.FirstName, e.LastName, o.OrderID
FROM Employees e
INNER JOIN Orders o ON e.EmployeeID = o.EmployeeID;
```

Вивести імена клієнтів і кількість їхніх замовлень:

```
SELECT c.ContactName, COUNT(o.OrderID) AS NumOrders
FROM Customers c
LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.ContactName;
```

Вивести всі продукти і назви категорій, до яких вони належать:

```
SELECT p.ProductName, c.CategoryName
FROM Products p
JOIN Categories c ON p.CategoryID = c.CategoryID;
```

Вивести всі міста, звідки були або клієнти, або співробітники (FULL JOIN):

```
SELECT e.City AS EmployeeCity, c.City AS CustomerCity
FROM Employees e
FULL OUTER JOIN Customers c ON e.City = c.City;
```

Контрольні запитання для самоперевірки

1. Чому у реляційній БД таблиці не зберігають усі дані в одній структурі?
2. Коли потрібно використовувати LEFT JOIN, а коли INNER JOIN?
3. Яка різниця між INNER JOIN і FULL OUTER JOIN?
4. Що таке декартовий добуток?
5. У чому логіка використання псевдонімів таблиць (e, o) при JOIN?

Словник термінів до Теми 4: З'єднання таблиць (JOIN-запити)

Термін	Пояснення
JOIN	З'єднання таблиць за певною умовою
INNER JOIN	Повертає лише співпадіння з обох таблиць
LEFT JOIN	Повертає всі рядки з лівої таблиці
RIGHT JOIN	Повертає всі рядки з правої таблиці
FULL JOIN	Повертає всі рядки з обох таблиць
CROSS JOIN	Повертає комбінації всіх рядків з обох таблиць
ON	Умова з'єднання
NULL	Значення, що позначає відсутність даних
Foreign Key	Поле, яке посилається на первинний ключ іншої таблиці

Практичні завдання: JOIN-запити

Увага: *орієнтуємось на базу даних Northwind або аналогічну структуру (Customers, Orders, Employees, Products, Categories, OrderDetails).*

Рівень 1 — Базові INNER JOIN

1. **Виведіть імена співробітників та ідентифікатори замовлень, які вони обробили.**

```
SELECT e.FirstName, e.LastName, o.OrderID
FROM Employees e
INNER JOIN Orders o ON e.EmployeeID = o.EmployeeID;
```

2. **Покажіть список клієнтів із номерами замовлень, які вони зробили.**

```
SELECT c.ContactName, o.OrderID
FROM Customers c
INNER JOIN Orders o ON c.CustomerID = o.CustomerID;
```

Рівень 2 — LEFT JOIN, агрегація з групуванням

3. **Покажіть усіх клієнтів, включаючи тих, які ще не зробили жодного замовлення.**

```
SELECT c.ContactName, o.OrderID
FROM Customers c
LEFT JOIN Orders o ON c.CustomerID = o.CustomerID;
```

4. **Порахуйте кількість замовлень для кожного клієнта.**

```
SELECT c.ContactName, COUNT(o.OrderID) AS NumOrders
FROM Customers c
LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.ContactName;
```

Рівень 3 — JOIN з кількома таблицями

5. **Виведіть назву продукту, його категорію та ціну.**

```
SELECT p.ProductName, c.CategoryName, p.UnitPrice
FROM Products p
JOIN Categories c ON p.CategoryID = c.CategoryID;
```

6. **Покажіть клієнтів, продукти, які вони замовляли, та кількість кожного з них.**

```
SELECT cu.ContactName, pr.ProductName, od.Quantity
FROM Customers cu
JOIN Orders o ON cu.CustomerID = o.CustomerID
JOIN OrderDetails od ON o.OrderID = od.OrderID
JOIN Products pr ON od.ProductID = pr.ProductID;
```

Рівень 4 — Завдання ускладнене

7. Покажіть працівників, які не обробляли жодного замовлення.

Використовуйте LEFT JOIN + WHERE ... IS NULL

8. Покажіть всі міста, звідки є або клієнти, або працівники (без повторів).

Можна реалізувати через FULL JOIN або UNION

Тести для самоконтролю

1. Який тип JOIN повертає лише ті записи, які мають відповідність у обох таблицях?
   - A) LEFT JOIN
   - B) FULL JOIN
   - C) INNER JOIN
   - D) CROSS JOIN

2. Який JOIN покаже всіх клієнтів, навіть якщо у них немає замовлень?
   - A) INNER JOIN
   - B) RIGHT JOIN
   - C) LEFT JOIN

- D) CROSS JOIN

3. Яке ключове слово використовується для вказання умови з’єднання?
   - A) WHERE
   - B) WITH
   - C) JOIN
   - D) ON

4. Що таке декартовий добуток у SQL?
   - A) Результат FULL JOIN
   - B) Спосіб злиття однакових таблиць
   - C) Випадкова вибірка рядків
   - D) Комбінація всіх рядків обох таблиць

5. Який результат поверне LEFT JOIN, якщо немає відповідності в правій таблиці?
   - A) NULL у всіх полях
   - B) Тільки повні відповідності
   - C) Рядок з лівої таблиці і NULL у правих стовпцях
   - D) Рядок не потрапить до результату

Ключі до тестів
1. Який тип JOIN повертає лише ті записи, які мають відповідність у обох таблицях?
Відповідь: INNER JOIN
2. Який JOIN покаже всіх клієнтів, навіть якщо у них немає замовлень?
Відповідь: LEFT JOIN
3. Яке ключове слово використовується для вказання умови з’єднання?
Відповідь: ON
4. Що таке декартовий добуток у SQL?
Відповідь: Комбінація кожного рядка з першої таблиці з кожним з другої
5. Який результат поверне LEFT JOIN, якщо немає відповідності в правій таблиці?
Відповідь: Дані з лівої таблиці + NULL у правій частині

Тема 5. Підзапити (Subqueries)

«Запит у запиті» як інструмент аналітичного та умовного вибору даних

5.1. Основна ідея: що таке підзапит?

Підзапит (subquery) — це SQL-запит, що знаходиться **всередині іншого запиту** та повертає проміжний результат, який використовується зовнішнім запитом для подальшої обробки.

Візуальна структура:

```
SELECT ...
FROM ...
WHERE [вираз з підзапитом: (SELECT ...)]
```

Умовно: підзапит → обчислює, зовнішній запит → використовує.

5.2. Навіщо використовувати підзапити?

Підзапити (subqueries) — це запити, які **вбудовуються всередину інших SQL-запитів**. Вони дозволяють отримати проміжні результати, які потім використовуються у зовнішньому запиті. Це надзвичайно потужний інструмент у SQL, що дає змогу писати більш гнучкі, динамічні та виразні запити.

Основна мета підзапиту — інкапсуляція логіки:
Замість того, щоб писати кілька окремих запитів і обробляти їх на прикладному рівні, можна все об’єднати в один SQL-запит. Це дозволяє делегувати більше обчислень самій СУБД, що підвищує продуктивність і зменшує складність коду.

Ключові переваги використання підзапитів

1. **Ізоляція логіки**
2. Підзапит дозволяє сконцентрувати складну частину логіки окремо від основного запиту. Наприклад, підрахунок середнього значення або вибір максимального ID.
3. **Динамічність**
4. Значення, отримані з підзапиту, можна використати у фільтрах основного запиту. Наприклад: «знайти усіх клієнтів, які зробили більше замовлень, ніж середній клієнт».
5. **Універсальність**
6. Підзапити можуть з’являтися в:
    - секції SELECT — для обчислення значення на льоту;
    - секції WHERE — для фільтрації;
    - секції FROM — для формування проміжної таблиці (інколи називається inline view).
7. **Покращення читабельності**
8. Часто підзапит дозволяє зробити головний запит простішим для розуміння, особливо якщо його можна сприймати як "проміжне обчислення".
9. **Замінник JOIN у деяких випадках**
10. У ситуаціях, де JOIN ускладнює структуру запиту або породжує зайві рядки, підзапит може надати лаконічніше рішення.

Приклади типових сценаріїв

- **Знайти працівників із вищою зарплатою, ніж середня:**

```
SELECT name
FROM employees
WHERE salary > (
    SELECT AVG(salary)
    FROM employees
);
```

- **Отримати список клієнтів, які оформили замовлення:**

```
SELECT name
FROM customers
WHERE customer_id IN (
    SELECT customer_id
    FROM orders
);
```

- **Підрахувати кількість замовлень для кожного клієнта:**

```
SELECT name,
       (SELECT COUNT(*)
        FROM orders o
        WHERE o.customer_id = c.customer_id) AS order_count
FROM customers c;
```

Коли краще використовувати підзапити?

- Коли необхідно порівняти значення з агрегованим результатом (наприклад, MAX, AVG, COUNT).
- Коли потрібно отримати значення з однієї таблиці, щоб використовувати його як фільтр для іншої.
- Коли необхідно уникнути складних з'єднань, але зберегти логіку запиту.

Підзапити — це один із найважливіших інструментів у SQL, який значно розширює можливості мови, дозволяючи вирішувати завдання, які складно або громіздко реалізувати лише за допомогою JOIN, GROUP BY або простих умов.

5.3. Типи підзапитів

За кількістю значень:

Тип	Пояснення	Використовується з
Скалярний	Повертає 1 значення	=, <, >
Множинний	Повертає список значень	IN, ANY, ALL
Табличний	Повертає таблицю	FROM, JOIN

За залежністю від зовнішнього запиту:

Тип	Приклад	Коли використовується
Некорельований	Виконується один раз	WHERE, SELECT, FROM
Корельований	Залежить від кожного рядка зовнішнього запиту	EXISTS, SELECT, WHERE

5.4. Поширені схеми використання підзапитів

1. Скалярний підзапит у WHERE

Знайти працівників, старших за середній вік:

```
SELECT FirstName, Age
FROM Employees
WHERE Age > (SELECT AVG(Age) FROM Employees);
```

2. Множинний підзапит з IN

Знайти клієнтів, які замовляли хоча б один з товарів з категорії 5:

```
SELECT DISTINCT CustomerID
FROM Orders
WHERE OrderID IN (
    SELECT OrderID
```

```
    FROM OrderDetails
    WHERE ProductID IN (
        SELECT ProductID FROM Products WHERE CategoryID = 5
    )
);
```

3. EXISTS (корельований)

Показати всіх клієнтів, які робили хоча б одне замовлення:

```
SELECT c.CustomerID, c.ContactName
FROM Customers c
WHERE EXISTS (
    SELECT 1
    FROM Orders o
    WHERE o.CustomerID = c.CustomerID
);
```

4. Підзапит у FROM (inline view)

Обчислити середню ціну продуктів по категоріях, і вивести категорії, де вона > 50:

```
SELECT CategoryID, AvgPrice
FROM (
    SELECT CategoryID, AVG(UnitPrice) AS AvgPrice
    FROM Products
    GROUP BY CategoryID
) AS SubAvg
WHERE AvgPrice > 50;
```

5.5. Порівняння: JOIN vs Subquery

Ознака	JOIN	Підзапит
Продуктивність	Часто швидший (індекси)	Може бути повільніший (особливо корельований)
Логіка	Поєднує таблиці	Інкапсулює запит
Гнучкість	Складні комбінації	Чисті обчислення, зручно в WHERE
Приклад	Дані про клієнтів та замовлення	Знайти клієнтів, які робили замовлення певного типу

Правило:

- якщо потрібно пов’язати таблиці → JOIN;
- якщо фільтрувати/порівнювати з результатами → Subquery.

5.6. Потенційні проблеми та оптимізація

- **Повільність корельованих підзапитів** → кожен рядок виконує вкладений запит → замінювати JOIN, де можливо
- **Велика кількість підзапитів у SELECT** → важко читати → використовувати CTE (Common Table Expressions) або WITH

- **Кешовані підзапити** → некорельовані виконуються один раз

Приклади запитів

Знайти найдорожчий продукт:

```
SELECT ProductName, UnitPrice
FROM Products
WHERE UnitPrice = (SELECT MAX(UnitPrice) FROM Products);
```

Знайти співробітника, який обробив найбільшу кількість замовлень:

```
SELECT EmployeeID, COUNT(*) AS NumOrders
FROM Orders
GROUP BY EmployeeID
HAVING COUNT(*) = (
    SELECT MAX(OrderCount)
    FROM (
        SELECT COUNT(*) AS OrderCount
        FROM Orders
        GROUP BY EmployeeID
    ) AS Sub
);
```

Знайти продукти, ціна яких вища за середню ціну в їхній категорії (корельований):

```
SELECT p.ProductName, p.UnitPrice
FROM Products p
WHERE UnitPrice > (
    SELECT AVG(UnitPrice)
    FROM Products
    WHERE CategoryID = p.CategoryID
);
```

Словник термінів до Теми 5

Термін	Пояснення
Subquery	Запит, який виконується усередині іншого SQL-запиту
Correlated Subquery	Підзапит, що залежить від кожного рядка зовнішнього запиту
Inline View	Підзапит у секції FROM, який обробляється як таблиця
Scalar	Повертає одне значення
EXISTS	Перевіряє наявність результатів у підзапиті
ANY, ALL	Використовуються з умовами >, <, = тощо для порівняння з множиною значень
Performance issue	Питання ефективності, що виникають при частому виконанні вкладених запитів

Тести для самоперевірки (вибір однієї правильної відповіді)

1. Який тип підзапиту виконується окремо, незалежно від зовнішнього запиту?
   - A) Корельований
   - B) Некорельований
   - C) Псевдозапит
   - D) JOIN

2. Що повертає скалярний підзапит?
   - A) Один стовпець
   - B) Один рядок
   - C) Таблицю
   - D) Одне значення

3. Який оператор застосовується до множини значень з підзапиту?
   - A) =
   - B) ALL
   - C) LIKE
   - D) IN

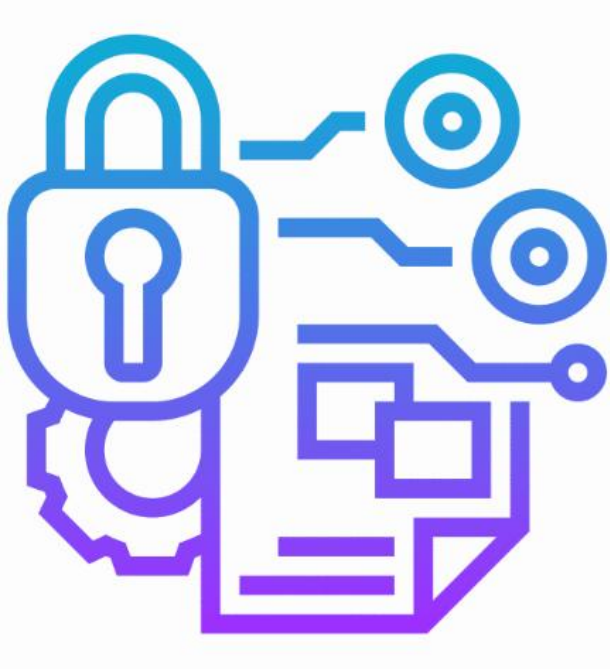
4. У якому випадку слід використовувати EXISTS?
   - A) Коли треба порівняти всі значення
   - B) Коли значень немає
   - C) Щоб фільтрувати порожні таблиці
   - D) Щоб перевірити наявність відповідних рядків

5. Який підзапит використовується в секції FROM і грає роль тимчасової таблиці?
   - A) IN-підзапит
   - B) EXISTS-підзапит
   - C) Inline view
   - D) Таблиця без JOIN

Ключі до тестів

1. Який тип підзапиту виконується окремо, незалежно від зовнішнього запиту?
Відповідь: Некорельований
2. Що повертає скалярний підзапит?
Відповідь: Одне значення
3. Який оператор застосовується до множини значень з підзапиту?
Відповідь: IN
4. У якому випадку слід використовувати EXISTS?
Відповідь: Коли потрібно перевірити наявність записів у підзапиті
5. Який підзапит використовується в секції FROM і грає роль тимчасової таблиці?
Відповідь: Inline view

Змістовий модуль 2: Оператори SQL визначення об’єктів бази даних. Захист даних. Оптимізація



Тема 6. Етапи проєктування бази даних. Визначення ключових полів. Визначення таблиць

Мета теми

Ознайомити студентів з основними етапами проєктування реляційної бази даних, навчити визначати структуру таблиць, вибирати ключі та формалізовувати її у вигляді SQL-коду.

6.1. Основи проєктування бази даних

Проєктування бази даних — це процес створення логічної та фізичної структури бази, яка дозволяє зберігати, організовувати та ефективно обробляти дані відповідно до потреб користувачів і систем. Це одна з ключових фаз у розробці будь-якої інформаційної системи.

Успішне проєктування бази забезпечує:

- узгодженість і цілісність даних,
- масштабованість системи,
- легкість підтримки та модифікації структури,
- високу продуктивність запитів.

6.2. Основні етапи проєктування

1. **Аналіз вимог**
    - Вивчення предметної області.
    - Визначення типів даних, які потрібно зберігати.
    - Виявлення зв’язків між сутностями.
    - Визначення бізнес-правил, які впливатимуть на структуру БД.

2. **Моделювання сутностей та зв'язків (ER-моделювання)**
    - Побудова **ER-діаграми** (Entity-Relationship), яка показує:
        - сутності (таблиці),
        - атрибути (поля),
        - зв'язки (один до одного, один до багатьох, багато до багатьох).
    - Визначення первинних і зовнішніх ключів.
3. **Нормалізація**
    - Процес поділу таблиць для усунення надлишковості та забезпечення цілісності.
    - Типові форми нормалізації:
        - Перша нормальна форма (1NF): усі поля — атомарні значення.
        - Друга нормальна форма (2NF): усі неключові атрибути залежать від усього ключа.
        - Третя нормальна форма (3NF): немає транзитивних залежностей.
    - У практиці часто зупиняються на 3NF, якщо не виникає специфічної потреби в денормалізації.
4. **Фізичне проєктування**
    - Вибір типів даних для кожного поля.
    - Створення індексів для пришвидшення пошуку.
    - Визначення обмежень (NOT NULL, UNIQUE, CHECK, DEFAULT).
    - Планування розгортання БД у СУБД (MySQL, PostgreSQL, Oracle тощо).
5. **Оптимізація структури**
    - Оцінка потенційних обсягів даних і навантаження.
    - Денормалізація у випадках, коли потрібна висока продуктивність читання.
    - Вибір ефективної структури з урахуванням запитів, що найчастіше виконуються.

Важливі міркування при проєктуванні

- **Цілісність даних:** забезпечення правильності даних через обмеження, зовнішні ключі та тригери.
- **Уніфікованість імен:** логічні та зрозумілі назви таблиць і полів.
- **Безпека:** контроль доступу до різних частин БД.
- **Масштабованість:** чи витримає структура ріст даних?
- **Змінюваність:** наскільки легко буде адаптувати БД до нових вимог?

Таким чином, проєктування бази даних — це не просто створення таблиць, а стратегічний процес, який потребує розуміння предметної області, чіткої логіки побудови та передбачення майбутніх змін. Грамотно спроєктована БД — це основа стабільної, продуктивної та довготривалої IT-системи.

6.3. Ключові поля в таблицях

Первинний ключ (PRIMARY KEY)

Первинний ключ — це фундаментальне поняття в реляційній моделі баз даних. Він забезпечує **унікальну ідентифікацію кожного рядка в таблиці**, що є критично важливим для цілісності даних і коректної роботи з відношеннями між таблицями.

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(50)
```

```
);
```

Основні властивості первинного ключа:

1. **Унікальність** Значення первинного ключа має бути **унікальним для кожного запису**. Два рядки з однаковим значенням первинного ключа не можуть існувати.
2. **NOT NULL** Значення первинного ключа **не може бути порожнім (NULL)**. Це логічно, адже NULL означає «невідоме значення», і ми не можемо однозначно ідентифікувати рядок з таким значенням.
3. **Єдиний у таблиці** У таблиці **може бути лише один первинний ключ**, хоча він може складатися з кількох полів (композитний ключ).

Простий і складений ключ

- **Простий ключ** — це один стовпець, який ідентифікує кожен запис (наприклад, user_id, order_id).
- **Складений (композитний) ключ** — це комбінація кількох стовпців, значення яких разом є унікальними (наприклад, student_id + course_id у таблиці реєстрацій на курси).

```
-- Простий ключ
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    name VARCHAR(100)
);

-- Композитний ключ
CREATE TABLE enrollments (
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    PRIMARY KEY (student_id, course_id)
);
```

Навіщо потрібен первинний ключ?

- Забезпечує **ідентифікацію запису** — це основа зв'язків між таблицями.
- Служить **цільовим полем для зовнішніх ключів** (FOREIGN KEY) в інших таблицях.
- Дозволяє **оптимізувати індексацію**: більшість СУБД автоматично створюють індекс для первинного ключа.
- Гарантує **відсутність дублікатів** у таблиці.

Вибір первинного ключа — практичні поради:

- Використовуйте стабільне, незмінне поле (наприклад, не ім'я користувача).
- Не використовуйте поля, які можуть змінитися або містити NULL.
- Часто використовуються **автоматично згенеровані ідентифікатори** (наприклад, SERIAL, IDENTITY, UUID) — це підвищує продуктивність і спрощує інтеграцію.

Первинний ключ — це «паспортизатор» кожного рядка таблиці. Його правильне визначення — ключовий крок у проєктуванні стабільної та логічно побудованої бази даних.

Зовнішній ключ (FOREIGN KEY)

Зовнішній ключ — це обмеження (constraint), яке вказує, що значення одного або кількох полів у таблиці повинні відповідати значенням первинного ключа (або унікального ключа) іншої таблиці.

Він слугує для **реалізації зв'язків між таблицями**, що лежить в основі реляційної моделі даних.

Основна роль зовнішнього ключа:

- **Підтримка цілісності даних:** запобігання створенню «осиротілих» записів, які посилаються на неіснуючі об'єкти.
- **Визначення зв'язків між сутностями:** наприклад, зв'язок між замовленням і клієнтом (orders.customer_id → customers.customer_id).

Синтаксис створення зовнішнього ключа

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Таке обмеження гарантує, що customer_id в таблиці orders **повинен існувати** в таблиці customers.

Приклади використання

1. One-to-Many (один до багатьох)

Один клієнт може мати багато замовлень:

```
-- Таблиця клієнтів
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100)
);

-- Таблиця замовлень
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

2. Self-referencing foreign key (рекурсивний зв'язок)

Структура працівників із керівниками:

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
```

```
    name VARCHAR(100),
    manager_id INT,
    FOREIGN KEY (manager_id) REFERENCES employees(employee_id)
);
```

Властивості зовнішнього ключа

- **Цілісність:** не дозволяє вставити значення, якого немає в пов'язаній таблиці.
- **Каскадні дії:** можна визначити, що відбувається при видаленні або оновленні батьківського запису:
    - ON DELETE CASCADE — автоматично видалити пов'язані записи.
    - ON UPDATE CASCADE — оновити значення в дочірній таблиці.
    - SET NULL або SET DEFAULT — встановити значення NULL або значення за замовчуванням.

```
FOREIGN KEY (customer_id)
    REFERENCES customers(customer_id)
    ON DELETE SET NULL
```

Типові помилки при роботі з зовнішніми ключами

- Спроба вставити або оновити значення, якого не існує в батьківській таблиці.
- Видалення запису з батьківської таблиці без каскадного правила, якщо дочірні записи ще існують.
- Невизначені або неконсистентні типи даних у ключових полях.

Зовнішній ключ — це механізм, який допомагає пов'язувати таблиці логічно та контролювати **референційну цілісність** у базі даних. Завдяки йому дані зберігають зв'язність, а система може ефективно відслідковувати і контролювати їхню взаємодію.

Унікальний ключ (UNIQUE)

Унікальний ключ (UNIQUE constraint) — це обмеження, яке гарантує, що значення в одному або кількох стовпцях таблиці **не повторюються**. Його використовують для забезпечення **унікальності даних**, окрім випадку первинного ключа.

Відмінності від первинного ключа

Властивість	PRIMARY KEY	UNIQUE
Кількість у таблиці	Лише один	Може бути кілька
NULL дозволено	Ні (NOT NULL обов'язково)	Так (одне або кілька NULL)
Індекс створюється	Автоматично	Автоматично
Призначення	Основний ідентифікатор	Гарантування унікальності

Таким чином, **UNIQUE** є додатковим механізмом для контролю унікальності, коли потрібно забезпечити неповторність значень, але ці поля не є основними ідентифікаторами.

Приклади використання

1. Унікальний email для користувача:

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    username VARCHAR(50),
    email VARCHAR(100) UNIQUE
);
```

2. Комбінована унікальність (наприклад, один студент не може двічі зареєструватися на той самий курс):

```
CREATE TABLE enrollments (
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    UNIQUE (student_id, course_id)
);
```

Особливості

- Якщо не вказано явно, UNIQUE допускає **NULL-значення**. Згідно з SQL-стандартом, **кілька NULL у полі з обмеженням UNIQUE дозволені**, оскільки NULL ≠ NULL. Проте реалізація може відрізнятись у різних СУБД.
- Обмеження UNIQUE автоматично створює **унікальний індекс** — це пришвидшує пошук і сортування за відповідним полем.
- UNIQUE можна оголошувати:
  - у визначенні стовпця (inline),
  - або на рівні таблиці (table-level).

Коли застосовувати UNIQUE

- Коли потрібно **забезпечити неповторність логічного ідентифікатора**, який не є основним ключем (наприклад, email, паспорт, серійний номер).
- Коли потрібно **унікальність для комбінації полів** (наприклад, країна + код міста).
- У складних моделях — як **альтернативний ключ**, за яким можна будувати зв'язки через зовнішній ключ.

UNIQUE — це простий, але ефективний спосіб зберігати **правильність і неповторність** критичних даних у базі. Він не замінює первинний ключ, а доповнює його, дозволяючи встановлювати додаткові обмеження без порушення гнучкості структури таблиць.

6.4. Визначення таблиць в SQL

Основна команда створення:

```
CREATE TABLE TableName (
    Column1 DataType [Constraints],
    Column2 DataType [Constraints],
    ...
);
```

Типи даних (приклади):

Тип	Призначення
INT	Цілі числа
VARCHAR(n)	Рядки змінної довжини (до n символів)
DATE	Дата
DECIMAL(p, s)	Число з фіксованою точністю

6.5. Приклади створення таблиць

Просте створення:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(30),
    LastName VARCHAR(30),
    BirthDate DATE,
    HireDate DATE
);
```

Таблиця з зовнішнім ключем:

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

З композитним ключем:

```
CREATE TABLE OrderDetails (
    OrderID INT,
    ProductID INT,
    Quantity INT,
    PRIMARY KEY (OrderID, ProductID)
);
```

Словник термінів

Термін	Визначення
Первинний ключ (PK)	Поле або набір полів, що унікально ідентифікують рядок
Зовнішній ключ (FK)	Поле, що посилається на PK в іншій таблиці
CREATE TABLE	Команда створення нової таблиці
Нормалізація	Процес усунення надлишковості шляхом розбиття таблиць
Атрибут	Характеристика об'єкта (стовпець у таблиці)
Сутність	Об'єкт предметної області, що зберігається в таблиці

Практичні завдання

1. Створіть таблицю Students з такими полями:
    - StudentID (INT, PK)
    - FullName (VARCHAR 50)
    - GroupName (VARCHAR 10)
2. Створіть таблицю Courses з:
    - CourseID (INT, PK)
    - CourseName (VARCHAR 100)
3. Створіть таблицю Enrollments, що зв’язує студентів і курси (M:N), з композитним ключем.
4. Створіть таблицю Departments з унікальним полем DeptCode.

5. Створіть таблицю Products, де:
    - ProductID — PK
    - CategoryID — FK до Categories(CategoryID)
    - UnitPrice — DECIMAL(10,2)
    - ProductName — не може бути порожнім

Тести для самоперевірки

1. Що таке первинний ключ?
- A) Поле, що містить лише цілі числа
- B) Поле, яке визначає ім’я таблиці
- C) Поле, що зберігає паролі
- D) Поле, що ідентифікує запис у таблиці

2. Що відбудеться, якщо створити таблицю без PRIMARY KEY?
- A) Таблиця буде порожня
- B) SQL поверне помилку
- C) Всі поля стануть первинними ключами
- D) Таблиця буде створена, але без гарантованої унікальності записів

3. Який оператор використовується для створення таблиці?
- A) MAKE TABLE
- B) BUILD TABLE
- C) NEW TABLE
- D) CREATE TABLE

4. Який ключ використовується для встановлення зв'язку між таблицями?
- A) Первинний ключ
- B) Унікальний ключ
- C) Індекс
- D) Зовнішній ключ

5. Яка з команд створює таблицю з композитним ключем?

Ключі до тестів

1. Що таке первинний ключ?
Відповідь: Поле, що унікально ідентифікує кожен запис у таблиці
2. Що відбудеться, якщо створити таблицю без PRIMARY KEY?
Відповідь: Можна створити, але таблиця не матиме унікального ідентифікатора
3. Який оператор використовується для створення таблиці?
Відповідь: CREATE TABLE
4. Який ключ використовується для встановлення зв'язку між таблицями?
Відповідь: Зовнішній ключ
5. Яка з команд створює таблицю з композитним ключем?
Відповідь:

```
CREATE TABLE X (
  A INT,
  B INT,
  PRIMARY KEY (A, B)
);
```

Тема 7. DML-операції — додавання, оновлення та видалення даних

Мета теми

Навчитися додавати нові записи до таблиць, змінювати існуючі та безпечно видаляти їх із використанням стандартних SQL-операторів INSERT, UPDATE, DELETE.

7.1. Що таке DML?

DML (Data Manipulation Language) — це підмножина SQL-команд, які використовуються для маніпуляцій з даними, тобто для їх створення, оновлення, видалення або отримання з таблиць бази даних. Саме ці команди дозволяють змінювати вміст таблиць, не змінюючи при цьому їхню структуру.

DML включає:

Оператор	Призначення
INSERT	Додавання нових записів
UPDATE	Оновлення значень у записах
DELETE	Видалення записів
SELECT	(не змінює дані — лише вибірка)

7.2. Додавання даних: INSERT

Синтаксис:

```
INSERT INTO TableName (Column1, Column2, ...)
VALUES (Value1, Value2, ...);
```

Приклад:

```
INSERT INTO Students (StudentID, FullName, GroupName)
VALUES (1, 'Оксана Бондар', 'PM-21');
```

Примітка: *якщо перелік полів не вказаний — значення повинні відповідати порядку оголошення у CREATE TABLE.*

7.3. Оновлення даних: UPDATE

Синтаксис:

```
UPDATE TableName
SET Column1 = Value1, Column2 = Value2
WHERE condition;
```

Приклад:

```
UPDATE Students
SET GroupName = 'PM-22'
WHERE StudentID = 1;
```

*Без WHERE зміняться **всі** записи в таблиці!*

7.4. Видалення даних: DELETE
Синтаксис:

```
DELETE FROM TableName
WHERE condition;
```

Приклад:

```
DELETE FROM Students
WHERE StudentID = 1;
```

*Команда без WHERE видаляє **всі записи**.*

7.5. Захист і безпечне використання DML

- **Обов'язково вказуйте WHERE** при UPDATE і DELETE!
- Перед серйозними змінами — **робіть резервну копію** або використовуйте транзакції.
- У складних запитах — перевіряйте результат SELECT перед UPDATE.

7.6. Приклади
Масове оновлення:

```
UPDATE Employees
SET Salary = Salary * 1.1
WHERE Department = 'Sales';
```

Додавання декількох записів (підтримується в багатьох СУБД):

```
INSERT INTO Courses (CourseID, CourseName)
VALUES
(101, 'Math Logic'),
(102, 'Databases'),
(103, 'Machine Learning');
```

Словник термінів до Теми 7

Термін	Визначення
INSERT	Оператор для додавання нових записів

Термін	Визначення
UPDATE	Оператор для зміни наявних записів
DELETE	Оператор для видалення записів
SET	Ключове слово, що вказує нові значення у UPDATE
WHERE	Умова для фільтрації записів, які буде змінено або видалено

Практичні завдання до Теми 7

1. Додайте до таблиці Students нового студента:
    - ID = 2
    - ПІБ = "Іван Коваль"
    - Група = "РМ-21"
2. Оновіть групу всіх студентів із "РМ-21" на "РМ-22".
3. Видаліть усіх студентів із таблиці, у яких поле GroupName = NULL.
4. Змініть назву курсу з ID = 102 на "Бази Даних".
5. Додайте три нові курси до таблиці Courses одним запитом.
6. Видаліть із таблиці Products усі товари з ціною менше 10.

Тести для самоперевірки

1. Який оператор додає новий запис до таблиці?
    - A) ADD
    - B) CREATE
    - C) INSERT
    - D) PUT

2. Яке ключове слово задає нові значення при UPDATE?
    - A) VALUES
    - B) CHANGE
    - C) MODIFY
    - D) SET

3. Що відбудеться, якщо виконати DELETE FROM Students без WHERE?
    - A) SQL видасть помилку
    - B) Видалиться перший запис
    - C) Нічого не станеться
    - D) Видаляться всі записи

4. Яка команда змінює існуючі дані у таблиці?
    - A) MODIFY
    - B) SET
    - C) UPDATE
    - D) ALTER

5. Що означає конструкція SET Salary = Salary * 1.1?
    - A) Встановити зарплату = 1.1
    - B) Помножити всі зарплати на 0
    - C) Підвищити зарплату на 10%
    - D) Понизити зарплату

Ключі до тестів
1. Який оператор додає новий запис до таблиці?
Відповідь: INSERT
2. Яке ключове слово задає нові значення при UPDATE?
Відповідь: SET
3. Що відбудеться, якщо виконати DELETE FROM Students без WHERE?
Відповідь: Усі записи в таблиці будуть видалені
4. Яка команда змінює існуючі дані у таблиці?
Відповідь: UPDATE
5. Що означає конструкція SET Salary = Salary * 1.1?
Відповідь: Підвищити значення зарплати на 10%

Тема 8. Забезпечення функціональних вимог до БД (керування доступом, обмеження, перевірки)

Мета теми
Навчити студентів застосовувати SQL-механізми для захисту даних, підтримки цілісності та реалізації правил перевірки значень у таблицях.

8.1. Що таке функціональні вимоги до БД?
Функціональні вимоги до бази даних — це набір очікуваних дій, які система має виконувати для забезпечення правильного, безпечного та ефективного доступу до даних. Вони визначають, **які функції повинна підтримувати база даних для користувачів, розробників і системних процесів**.
Ці вимоги впливають як на **структуру** бази даних, так і на **механізми керування доступом, перевірки, обмеження цілісності, продуктивність та масштабованість**.

Основні категорії функціональних вимог
1. **Контроль доступу та автентифікація**
    - Визначення, хто може читати, змінювати, додавати або видаляти дані.
    - Забезпечення рівнів доступу (наприклад: адміністратор, оператор, користувач).
    - Приклад: лише менеджери можуть редагувати прайс-листи.
2. **Обмеження на значення (constraints)**
    - Забезпечення правильності введених даних через обмеження типу:
        - NOT NULL, UNIQUE, CHECK, DEFAULT, FOREIGN KEY.
    - Приклад: поле «вік» має бути більшим за 0.
3. **Перевірки цілісності (integrity checks)**
    - Гарантія того, що дані мають логічну узгодженість.
    - Наприклад, не можна видалити клієнта, якщо на нього є активні замовлення (referential integrity).
4. **Журналювання змін та історії (audit trails)**
    - Можливість відстежити, хто і коли змінив або видалив дані.
    - Важливо для безпеки та відповідності нормативам (наприклад, GDPR).
5. **Захист від помилок введення (validation)**
    - Система повинна виявляти некоректні значення до того, як вони потраплять у БД.

    - Перевірка на рівні СУБД або в логіці застосунку.
6. **Підтримка транзакцій і узгодженості**
    - Здатність об'єднати кілька дій у єдину транзакцію, яка або виконується повністю, або не виконується зовсім.
    - Забезпечення властивостей ACID: атомарність, узгодженість, ізольованість, довговічність.
7. **Масштабованість і продуктивність**
    - База має відповідати запланованому обсягу навантаження, кількості одночасних користувачів, обсягів збереження.
    - Вимоги можуть передбачати індексування, кешування, розподілення навантаження тощо.
8. **Звітність та пошук**
    - Здатність виконувати запити для генерації звітів, агрегації та пошуку даних.
    - Приклад: формування щомісячного звіту про продажі по регіонах.

Приклади формулювання функціональних вимог

- Система повинна зберігати всі замовлення щонайменше протягом 5 років.
- Користувачі повинні мати змогу знайти товар за назвою або категорією.
- Усі фінансові транзакції мають бути зафіксовані з точним часом операції.
- Доступ до медичних записів дозволено лише авторизованим лікарям.

Таким чином, **функціональні вимоги — це інструкція до того, що саме повинна "вміти" база даних** для підтримки бізнес-процесів, безпеки та користувацького досвіду. Вони тісно пов’язані з правилами бізнес-логіки і визначають очікувану поведінку системи при роботі з даними.

8.2. Обмеження цілісності (constraints)

Обмеження — це правила, що застосовуються до стовпців для перевірки допустимості значень.

Основні типи обмежень:

- **PRIMARY KEY** – забезпечує унікальність записів
- **FOREIGN KEY** – забезпечує зв’язок між таблицями
- **UNIQUE** – гарантує, що значення в стовпці не повторюються
- **NOT NULL** – не дозволяє зберігати порожнє значення
- **CHECK** – задає умову, яку мають задовольняти значення
- **DEFAULT** – задає значення за замовчуванням

Приклад створення таблиці з обмеженнями:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    LastName VARCHAR(30) NOT NULL,
    Age INT CHECK (Age >= 18),
    Salary DECIMAL(10,2) DEFAULT 0.00,
    Email VARCHAR(50) UNIQUE
);
```

8.3. Керування доступом до БД

SQL дозволяє керувати тим, хто має доступ до таблиць, бази даних або окремих операцій.
Оператори:

- **GRANT** – надання прав
- **REVOKE** – відкликання прав

Приклад надання прав користувачу:

```
GRANT SELECT, INSERT ON Employees TO analyst_user;
```

Приклад відкликання прав:

```
REVOKE INSERT ON Employees FROM analyst_user;
```

Додатково:

- Можна надавати права на таблицю, окремі поля, перегляди (views)
- Права надаються користувачам або ролям
- У багатьох СУБД адміністратор БД створює ролі з певними привілеями

8.4. Приклади перевірок і обмежень

Перевірка обмеження зарплати:

```
CHECK (Salary BETWEEN 1000 AND 10000)
```

Поле з обов’язковим значенням:

```
LastName VARCHAR(50) NOT NULL
```

Автоматичне значення за замовчуванням:

```
Status VARCHAR(10) DEFAULT 'Active'
```

Унікальність email:

```
Email VARCHAR(100) UNIQUE
```

Зв’язок між таблицями:

```
KEY (DepartmentID) REFERENCES Departments(DepartmentID)
```

Словник термінів до Теми 8

Термін	Пояснення
PRIMARY KEY	Унікальний ідентифікатор запису
FOREIGN KEY	Посилання на інший запис у пов’язаній таблиці
CHECK	Обмеження, яке перевіряє умову
NOT NULL	Не допускає порожніх значень
UNIQUE	Значення не може повторюватися
DEFAULT	Значення за замовчуванням
GRANT	Надання прав доступу
REVOKE	Вилучення прав доступу

Практичні завдання

1. Створіть таблицю Departments з полями:
    - DepartmentID (PK)
    - Name (уникальне значення)
    - City
2. Створіть таблицю Employees з полями:
    - EmployeeID (PK)
    - LastName (обов’язкове)
    - Age (перевірка: не менше 18)
    - Email (унікальне)
    - DepartmentID (FK до Departments)
3. Додайте перевірку, що зарплата не може бути менше 5000.
4. Змініть таблицю Employees, щоб за замовчуванням статус нового співробітника був ‘Active’.
5. Надати користувачу hr_manager право на SELECT, UPDATE у таблиці Employees.
6. Заборонити користувачу temp_user можливість INSERT у таблицю Departments.

Тести для самоперевірки

1. Яке обмеження не дозволяє зберігати порожнє значення?

- A) UNIQUE
- B) NOT NULL
- C) PRIMARY KEY
- D) CHECK

2. Який оператор використовується для надання прав доступу до таблиці?

- A) GIVE
- B) ALLOW
- C) GRANT
- D) ACCESS

3. Яке обмеження гарантує, що значення в полі не повторюються?

- A) DEFAULT
- B) UNIQUE
- C) CHECK
- D) NOT NULL

4. Яка перевірка відповідає за діапазон допустимих значень?

- A) SET

- B) RANGE
- C) CHECK
- D) RULE

5. Що робить команда REVOKE INSERT ON Employees FROM user123?
- A) Встановлює INSERT за замовчуванням
- B) Наділяє користувача можливістю вставки
- C) Забирає у користувача право додавати записи
- D) Створює роль користувача

Ключі до тестів

1. Яке обмеження не дозволяє зберігати порожнє значення?
Відповідь: NOT NULL
2. Який оператор використовується для надання прав доступу до таблиці?
Відповідь: GRANT
3. Яке обмеження гарантує, що значення в полі не повторюються?
Відповідь: UNIQUE
4. Яка перевірка відповідає за діапазон допустимих значень?
Відповідь: CHECK
5. Що робить команда REVOKE INSERT ON Employees FROM user123?
Відповідь: Забирає у користувача право додавати записи

Тема 9. Тригери та транзакції

Мета теми

Надати студентам розуміння механізмів, що забезпечують автоматизацію дій у базі даних (тригери) та збереження узгодженості даних (транзакції). Навчити використовувати ці засоби для забезпечення надійності, безпеки та реактивності бази даних.

9.1. Що таке транзакції?

Транзакція в базах даних — це **послідовність операцій**, яка виконується як єдине логічне ціле. Якщо хоча б одна операція в транзакції не виконується успішно, **всі зміни скасовуються**, і база даних повертається до попереднього стану.

Транзакції є ключовим механізмом для забезпечення **надійності, узгодженості та цілісності даних**.

Основні властивості транзакцій: ACID

1. **Атомарність (Atomicity)**
2. Усі операції в межах транзакції виконуються повністю або не виконуються взагалі. Якщо одна з них завершується помилкою — відбувається "відкат" (rollback).
3. **Узгодженість (Consistency)**
4. Транзакція переводить базу даних з одного правильного стану в інший, не порушуючи обмежень цілісності.
5. **Ізольованість (Isolation)**
6. Кожна транзакція виконується **незалежно** від інших. Навіть якщо кілька транзакцій виконуються одночасно, результат повинен бути таким, ніби вони виконувались послідовно.

7. **Довговічність (Durability)**
8. Після фіксації (commit), результати транзакції **зберігаються назавжди**, навіть у випадку збою системи.

Приклад: транзакція на банківському рахунку
Припустимо, потрібно переказати гроші з рахунку A на рахунок B:
1. Зменшити баланс рахунку A на 100 грн
2. Збільшити баланс рахунку B на 100 грн

Ці дві дії повинні виконатися **разом**, як одне ціле. Якщо під час другої дії відбувається збій, **перша дія також повинна бути скасована**, щоби запобігти втраті коштів.

SQL-приклад транзакції:
```
BEGIN;

UPDATE accounts SET balance = balance - 100 WHERE account_id =
1;
UPDATE accounts SET balance = balance + 100 WHERE account_id =
2;

COMMIT;
```
Якщо під час виконання однієї з команд виникне помилка, можна виконати:
ROLLBACK;
Це поверне БД до стану перед початком транзакції.

Коли використовуються транзакції?
- Фінансові операції (перекази, оплати)
- Замовлення товарів
- Реєстрація користувачів
- Масове оновлення даних
- Пакетні імпорти або міграції

9.2. Команди керування транзакціями
SQL забезпечує транзакційність через такі команди:
- **BEGIN TRANSACTION** — початок транзакції
- **COMMIT** — збереження змін
- **ROLLBACK** — відкат транзакції

Приклад:
```
BEGIN TRANSACTION;

UPDATE Accounts SET Balance = Balance - 500 WHERE AccountID = 1;
UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 2;

IF @@ERROR <> 0
    ROLLBACK;
ELSE
    COMMIT;
```

Цей приклад забезпечує переказ коштів між рахунками. Якщо хоча б одна команда не виконується — обидві скасовуються.

9.3. Що таке тригер?

Тригер (trigger) — це спеціальний об’єкт у базі даних, який автоматично виконує **певні дії у відповідь на події**, що відбуваються в таблиці або поданні. Зазвичай тригери спрацьовують при **вставці (INSERT), оновленні (UPDATE)** або **видаленні (DELETE)** даних.
Іншими словами, тригери — це **автоматизовані реакції** бази даних на зміну її стану.

Основна ідея
Коли певна подія (наприклад, додавання нового рядка до таблиці) відбувається, тригер **"слухає"** цю подію й **виконує заздалегідь визначену інструкцію** (наприклад, логування, обмеження або виклик процедури).

Типові сценарії використання тригерів

- Автоматичне логування змін у таблиці.
- Підтримка цілісності, яку не можна виразити звичайними обмеженнями (CHECK, FOREIGN KEY).
- Обчислення агрегатних значень (наприклад, підрахунок кількості записів).
- Синхронізація даних між таблицями.
- Блокування небажаних змін або введення даних.

Приклад: логування видалень

```
CREATE TABLE deleted_users_log (
    user_id INT,
    deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TRIGGER log_user_deletion
AFTER DELETE ON users
FOR EACH ROW
INSERT INTO deleted_users_log (user_id)
VALUES (OLD.user_id);
```

Цей тригер автоматично додає запис до журналу deleted_users_log, коли з таблиці users видаляється користувач.

Обмеження

- Не всі СУБД підтримують тригери однаково (наприклад, у MySQL немає INSTEAD OF).
- Тригери не повинні містити нескінченних викликів (наприклад, якщо INSERT у тригері викликає ще один INSERT, який викликає цей самий тригер).
- Надмірне використання тригерів може призвести до **зниження продуктивності** або **ускладнення логіки** БД.

Тригери — це потужний інструмент автоматизації логіки на рівні бази даних. Вони дозволяють впроваджувати контроль, обмеження і реакції без потреби втручання зі сторони застосунку. Проте їх слід використовувати обачно, щоб уникнути прихованих побічних ефектів.

9.4. Структура тригера

Тригер має:

- назву
- таблицю, до якої прив’язаний
- подію (INSERT, UPDATE, DELETE)
- час виконання (AFTER, BEFORE, INSTEAD OF)
- тіло (дії, які слід виконати)

Синтаксис:

```
CREATE TRIGGER trg_UpdateLog
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
    INSERT INTO LogTable (Action, ChangedBy, ChangeDate)
    VALUES ('Updated Employee', CURRENT_USER,
    CURRENT_TIMESTAMP);
END;

B SQL Server:
CREATE TRIGGER trg_InsertBlocker
ON Orders
INSTEAD OF INSERT
AS

BEGIN
    RAISERROR ('Вставка нових замовлень заборонена у цьому
режимі', 16, 1);
    ROLLBACK;
END;
```

9.5. Типи тригерів

- **AFTER** — виконується після події (найбільш поширений)
- **BEFORE** — виконується до зміни (корисно для перевірок)
- **INSTEAD OF** — замінює подію, наприклад для заборони дії

У різних СУБД підтримуються різні типи тригерів (наприклад, у SQL Server — AFTER та INSTEAD OF, у PostgreSQL — BEFORE, AFTER, INSTEAD OF).

9.6. Зауваження щодо використання тригерів

- Тригери важко відлагоджувати
- Вони приховані — виконуються без явного виклику
- Можуть впливати на продуктивність (особливо вкладені)
- Тригери не повинні містити важку бізнес-логіку — лише обробку подій

Словник термінів до Теми 9

Термін	Визначення
Транзакція	Послідовність операцій, що виконується як єдине ціле

Термін	Визначення
COMMIT	Підтвердження змін
ROLLBACK	Відкат змін у межах транзакції
ACID	Властивості надійності транзакцій
Тригер	Автоматичне виконання коду при певній події
AFTER / BEFORE	Типи тригерів залежно від моменту виконання
INSTEAD OF	Тригер, що замінює дію
RAISERROR	Оператор для виведення помилки в SQL Server

Практичні завдання

1. Створіть таблицю AuditLog з полями: ID, Action, UserName, ChangeTime.
2. Створіть тригер, який після оновлення запису в таблиці Employees додає запис у AuditLog.
3. Реалізуйте тригер, який **забороняє видалення** записів зі списку співробітників, якщо їхній статус = 'Active'.
4. Створіть транзакцію, яка:
    - зменшує кількість товарів на складі
    - створює запис у таблиці продажів
    - у випадку помилки скасовує все
5. Напишіть тригер, який автоматично змінює значення Status на 'Archived' у таблиці Orders, якщо ShippedDate старіша 30 днів.

Тести для самоперевірки

1. Яка властивість транзакції гарантує, що вона або виконується повністю, або не виконується зовсім?

- A) Isolation
- B) Atomicity
- C) Durability
- D) Consistency

2. Що робить оператор ROLLBACK?

- A) Завершує тригер
- B) Виводить помилку
- C) Відміняє всі зміни в транзакції
- D) Виконує перевірку цілісності

3. Що таке тригер у базі даних?

- A) Зовнішній ключ
- B) Сценарій для експорту БД
- C) Автоматична реакція на подію
- D) Стандарт SQL для транзакцій

4. Коли виконується тригер типу BEFORE?

- A) Одразу після COMMIT
- B) До виконання основної операції
- C) Після транзакції
- D) Коли в БД є помилка

5. Який тип тригера дозволяє повністю замінити дію?

- A) AFTER
- B) BEFORE
- C) INSTEAD OF
- D) DEFERRED

Тести для самоперевірки

1. Яка властивість транзакції гарантує, що вона або виконується повністю, або не виконується зовсім?

Відповідь: Atomicity

2. Що робить оператор ROLLBACK?

Відповідь: Відміняє всі зміни в транзакц

3. Що таке тригер у базі даних?

Відповідь: Автоматична реакція на подію

4. Коли виконується тригер типу BEFORE?

Відповідь:До виконання основної операції

5. Який тип тригера дозволяє повністю замінити дію?

Відповідь: INSTEAD OF

Тема 10. Процедурний SQL. Оператори мови Transact-SQL і PL/SQL

Мета теми

Познайомити студентів з розширеннями SQL, що дозволяють описувати складну бізнес-логіку в БД у вигляді **процедур, умовних конструкцій, циклів та змінних**. Навчити використовувати **Transact-SQL (T-SQL)** у SQL Server та **PL/SQL** в Oracle для написання програмного коду на стороні сервера.

10.1. Що таке процедурний SQL?

Процедурний SQL (PL/SQL, T-SQL, PL/pgSQL) — це розширення стандартної мови SQL, яке дозволяє використовувати **елементи процедурного програмування**: змінні, умовні оператори (IF), цикли (LOOP, WHILE), обробку помилок, процедури та функції. У той час як стандартний SQL в основному описує **що потрібно зробити з даними** (декларативний підхід), процедурний SQL дозволяє визначити **як це зробити** — крок за кроком, з контролем над потоком виконання.

Основні характеристики процедурного SQL

1. **Контроль потоку**
    - Умовні конструкції: IF...THEN...ELSE, CASE
    - Цикли: FOR, WHILE, LOOP
    - Оператори переходу: EXIT, RETURN, GOTO (залежно від СУБД)
2. **Змінні**
    - Оголошення змінних: DECLARE
    - Присвоєння значень: := або SET
    - Типізація: автоматична (%TYPE, %ROWTYPE) або явна
3. **Процедури та функції**

- Можливість створювати **власні підпрограми**, які виконують дії з даними (наприклад, оновлення записів, обчислення значень)
- Процедури (PROCEDURE) не повертають результату, а функції (FUNCTION) повертають значення

4. **Обробка виключень**
   - Механізм EXCEPTION дозволяє обробляти помилки та виняткові ситуації (наприклад, поділ на нуль, помилки доступу до таблиці)

Поширені реалізації

СУБД	Реалізація процедурного SQL
Oracle	PL/SQL
PostgreSQL	PL/pgSQL
Microsoft SQL Server	T-SQL
MySQL	SQL/PSM (з обмеженнями)

Приклад: процедура в PL/pgSQL

```
CREATE OR REPLACE FUNCTION increase_salary(emp_id INT, percent
NUMERIC)
RETURNS VOID AS $$
BEGIN
  UPDATE employees
  SET salary = salary + (salary * percent / 100)
  WHERE id = emp_id;
END;
$$ LANGUAGE plpgsql;
```

Ця функція підвищує зарплату працівника на вказаний відсоток.

Коли варто використовувати процедурний SQL

- Для складних бізнес-правил, які не можна виразити одним SQL-запитом
- Для обробки даних у циклах (наприклад, покрокове оновлення)
- Для централізованої логіки в базі даних (менше залежності від коду застосунку)
- Для створення тригерів, процедур, пакетів

Переваги

- Підвищення продуктивності за рахунок зменшення кількості звернень до БД
- Централізація логіки (використання повторно у функціях і процедурах)
- Краща підтримка транзакційного контролю

Недоліки

- Залежність від конкретної СУБД (низька портативність)
- Складність налагодження та тестування
- Може ускладнювати структуру проєкту, якщо не дотримуватись модульності

Процедурний SQL — це міст між SQL-запитами та повноцінним програмуванням. Він дає змогу реалізовувати гнучку логіку всередині БД, оптимізувати роботу з великими обсягами даних та автоматизувати рутинні процеси. Однак його використання вимагає обережності й структурованого підходу.

10.2. Основні елементи процедурного SQL

Змінні

Оголошуються за допомогою DECLARE і ініціалізуються через SET або SELECT.

```
DECLARE @name VARCHAR(50);
SET @name = 'Olena';
```

Умовні конструкції

```
IF @age >= 18
    PRINT 'Доступ дозволено';
ELSE
    PRINT 'Відмовлено';
```

Цикли (у SQL Server)

```
WHILE @counter < 10
BEGIN
    SET @counter = @counter + 1;
END
```

Блок BEGIN...END

Об'єднує декілька операторів в один логічний блок.

```
BEGIN
    SET @sum = @a + @b;
    PRINT @sum;
END
```

10.3. Процедури та функції

Збережена процедура (Stored Procedure)

Це іменований блок SQL-коду, який може мати параметри та викликатись ззовні.

```
CREATE PROCEDURE GetEmployeeByID
    @EmpID INT
AS
BEGIN
    SELECT * FROM Employees WHERE EmployeeID = @EmpID;
END;
```

```

```

Виклик:
```
EXEC GetEmployeeByID @EmpID = 5;
```

Функція (User-Defined Function)
Повертає значення або таблицю, часто використовується у виразах.

```
CREATE FUNCTION GetBonus(@Salary DECIMAL)
RETURNS DECIMAL
AS
BEGIN
    RETURN @Salary * 0.1;
END;
```

Виклик:
```
SELECT dbo.GetBonus(Salary) FROM Employees;
```

10.4. Відмінності між Transact-SQL і PL/SQL

Ознака	Transact-SQL (T-SQL)	PL/SQL
Використовується в	SQL Server	Oracle
Синтаксис	@var, BEGIN...END	DECLARE, BEGIN...END;
Підтримка курсорів	Так	Так
Обробка помилок	TRY...CATCH	EXCEPTION
Основні об'єкти	Procedure, Function, Trigger	Procedure, Function, Package, Trigger

10.5. Переваги використання процедурного SQL

- Мінімізація трафіку між застосунком і БД
- Підвищення безпеки (контроль доступу до логіки)
- Повторне використання коду (DRY-принцип)
- Централізація правил обробки даних

Словник термінів до Теми 10

Термін	Пояснення
Transact-SQL	Процедурне розширення SQL у SQL Server
PL/SQL	Процедурне розширення SQL в Oracle
Змінна	Локальний контейнер для значення
BEGIN...END	Блок для групування команд
WHILE	Цикл із перевіркою умови
PROCEDURE	Збережений блок логіки, який не повертає значення
FUNCTION	Об'єкт, який повертає значення
EXEC	Виклик збереженої процедури

Практичні завдання

1. Створіть збережену процедуру, яка приймає ID клієнта та виводить усі його замовлення.
2. Створіть збережену процедуру, яка оновлює зарплату співробітника на вказаний відсоток.
3. Напишіть функцію, яка обчислює ПДВ (20%) для переданої ціни товару.
4. Реалізуйте цикл WHILE, що додає числа від 1 до 10 і виводить суму.
5. Створіть процедуру, яка перевіряє вік працівника. Якщо менше 18 — виводить повідомлення.
6. Порівняйте оголошення змінної в PL/SQL і T-SQL.

Тести для самоперевірки

1. Що таке збережена процедура?
- A) Таблиця в пам'яті
- B) Об'єкт, який зберігає код SQL і може викликатися
- C) Обмеження цілісності
- D) Курсовий об'єкт

2. Що робить оператор WHILE?
- A) Перевіряє тип даних
- B) Виконує команду один раз
- C) Повторює блок, поки виконується умова
- D) Виводить значення

3. Яка конструкція групує декілька SQL-команд?
- A) BLOCK
- B) BUNDLE
- C) BEGIN...END
- D) GROUP

4. Який символ використовується в T-SQL для змінних?
- A) $
- B) #
- C) @
- D) &

5. Чим відрізняється функція від процедури?
- A) Функція завжди викликає тригер
- B) Функція не може повертати значення
- C) Процедура не може використовуватись у SELECT
- D) Функція повертає значення, процедура — ні

Ключі до тестів

1. Що таке збережена процедура?
   Відповідь: Об'єкт, який зберігає код SQL і може викликатися
6. Що робить оператор WHILE?
   Відповідь: Повторює блок, поки виконується умова
7. Яка конструкція групує декілька SQL-команд?
   Відповідь: BEGIN...END
8. Який символ використовується в T-SQL для змінних?

Відповідь: @

9. Чим відрізняється функція від процедури?
   Відповідь: Функція повертає значення, процедура — ні

Тема 11. Збережені процедури і функції

Мета теми

Навчити студентів створювати, використовувати та розрізняти **збережені процедури (stored procedures)** і **користувацькі функції (user-defined functions)** як інструменти організації бізнес-логіки на рівні бази даних.

11.1. Що таке збережена процедура?

Збережена процедура (stored procedure) — це **програмний блок SQL-коду**, збережений у базі даних, який можна викликати багаторазово для виконання певної задачі або послідовності операцій з даними. Вона дозволяє **інкапсулювати бізнес-логіку** на рівні БД, зменшуючи кількість запитів між застосунком і сервером бази даних. Збережена процедура виконується на сервері й може містити інструкції SELECT, INSERT, UPDATE, DELETE, а також **логіку керування потоком**: змінні, умови, цикли, обробку помилок тощо.

Основні особливості

- Зберігається в БД та виконується за запитом користувача або програми
- Може мати **вхідні (IN)**, **вихідні (OUT)** та **вхідно-вихідні (INOUT) параметри**
- Підтримує **модульність**: логіка не дублюється в коді
- Часто використовується для **автоматизації типових операцій**

Переваги використання

- **Продуктивність**: менше трафіку між клієнтом і сервером — уся логіка виконується на сервері
- **Безпека**: можна обмежити прямий доступ до таблиць і надавати лише право виклику процедур
- **Управління транзакціями**: легко реалізується через BEGIN, COMMIT, ROLLBACK
- **Повторне використання**: одну процедуру можна викликати з різних частин застосунку

Приклад: створення простої збереженої процедури (MySQL)

```
DELIMITER $$

CREATE PROCEDURE GetCustomerOrders (IN cust_id INT)
BEGIN
    SELECT * FROM orders WHERE customer_id = cust_id;
END $$

DELIMITER ;
```

Виклик процедури:

```
CALL GetCustomerOrders(102);
```

Приклад із вихідним параметром (PostgreSQL)

```
CREATE OR REPLACE PROCEDURE update_salary(emp_id INT, increase
NUMERIC)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE employees
    SET salary = salary + increase
    WHERE id = emp_id;
END;
$$;
```

Виклик у PostgreSQL:

```
CALL update_salary(5, 1000);
```

Відмінність від функції

Ознака	Збережена процедура	Функція
Повертає значення	Не обов'язково	Обов'язково (через RETURN)
Виклик у запитах	Ні (зазвичай через CALL)	Так (SELECT func())
Параметри	IN, OUT, INOUT	IN
Може змінювати дані	Так	Залежить від СУБД
Підтримка транзакцій	Так	Обмежена або залежить від СУБД

Використання збережених процедур

- Під час **вставки складних даних** із перевіркою
- Для **періодичних задач** (розсилки, звіти)
- Для **обробки подій**, викликаних з фронтенду
- В рамках **архітектури мікросервісів** для інкапсуляції логіки

Збережені процедури — це **ефективний механізм розмежування логіки обробки даних і прикладного коду**, який дає змогу централізувати управління, покращити продуктивність і забезпечити кращу безпеку даних. Їх застосування особливо доречне у великих інформаційних системах із багаторазовими типами запитів.

11.2. Синтаксис створення збереженої процедури

У SQL Server (Transact-SQL):

```
CREATE PROCEDURE GetCustomerOrders
    @CustomerID INT
AS
BEGIN
    SELECT * FROM Orders WHERE CustomerID = @CustomerID;
END;
```

Виклик:

```
EXEC GetCustomerOrders @CustomerID = 5;
```

В Oracle PL/SQL:

```
CREATE OR REPLACE PROCEDURE GetCustomerOrders (cust_id IN NUMBER) AS
BEGIN
    SELECT * FROM Orders WHERE CustomerID = cust_id;
END;
```

11.3. Що таке користувацька функція?

Функція (UDF — user-defined function) — це іменований SQL-об'єкт, який повертає значення: скалярне або табличне. Функції дозволяють **використовувати логіку обчислення в запитах**, наприклад у SELECT, WHERE, JOIN.

Особливості:

- Можна використовувати як частину запиту
- Обов'язково має оператор RETURN
- Не може змінювати стан бази (у більшості реалізацій)
- Використовується в SELECT, SET, WHERE

Скалярна функція повертає одне значення:

```
CREATE FUNCTION GetVAT (@price DECIMAL(10,2))
RETURNS DECIMAL(10,2)
AS
BEGIN
    RETURN @price * 0.2;
END;
```

Використання:

```
SELECT ProductName, UnitPrice, dbo.GetVAT(UnitPrice) AS VAT FROM Products;
```

11.4. Табличні функції

Табличні функції повертають таблицю. Це зручно, коли потрібно вставити обчислювану таблицю в FROM:

```
CREATE FUNCTION GetTopProducts(@minPrice DECIMAL)
RETURNS TABLE
AS
RETURN (
    SELECT ProductID, ProductName
    FROM Products
    WHERE UnitPrice > @minPrice
);
SELECT * FROM dbo.GetTopProducts(50);
```

11.5. Порівняння: процедура vs функція

Ознака	Процедура	Функція
Повернення значення	Не обов'язкове	Обов'язково

Ознака	Процедура	Функція
Підтримка змін таблиць	Так	Як правило — ні
Виклик у запиті	Ні	Так (у SELECT, WHERE, тощо)
Параметри	IN, OUT, INOUT	лише IN
RETURN	Немає або RETURN без значення	RETURN зі значенням обов’язковий
Виклик	EXEC або CALL	використовується в SQL-виразі

11.6. Права доступу до процедур і функцій

У БД можна:

- надати **доступ лише до процедури**, а не до таблиці
- обмежити **виклик функції** окремими ролями
- використовувати **обгортки над важливими таблицями** у вигляді функцій/процедур

```
GRANT EXECUTE ON GetCustomerOrders TO sales_user;
```

Словник термінів до Теми 11

Термін	Визначення
Збережена процедура	Іменований SQL-блок, який виконує операції і може приймати параметри
Користувацька функція	Об’єкт БД, який повертає значення і використовується в запитах
RETURN	Оператор повернення значення у функції
EXEC	Команда виклику процедури
INPUT parameter	Вхідний параметр
OUTPUT parameter	Параметр, який дозволяє виводити результат

Практичні завдання

1. Створіть збережену процедуру GetStudentByGroup, яка приймає назву групи і повертає студентів цієї групи.
2. Створіть функцію CalcScholarship — вона приймає середній бал і повертає суму стипендії за таким правилом:
   - якщо бал ≥ 95 — 2000 грн
   - 90–94 — 1500 грн
   - 85–89 — 1000 грн
   - інакше — 0 грн
3. Напишіть табличну функцію GetTop3ExpensiveProducts, яка повертає три найдорожчі товари.
4. Створіть процедуру LogDeleteAttempt, яка записує в таблицю LogTable ID запису, дату і користувача, який намагався видалити запис.
5. Додайте GRANT EXECUTE для вашої процедури GetStudentByGroup ролі instructor.
6. Виведіть назву продукту та обчислену ПДВ через власну функцію GetVAT(UnitPrice).

Тести для самоперевірки

1. Чим функція відрізняється від процедури?
- A) Функція не приймає параметри
- B) Функція не може бути викликана з SELECT
- C) Функція завжди повертає значення
- D) Функція змінює таблиці

2. Яка команда викликає збережену процедуру?
- A) RUN
- B) SELECT
- C) EXEC
- D) USE

3. Яке ключове слово використовується для повернення значення у функції?
- A) RETURN
- B) OUTPUT
- C) SET
- D) SELECT

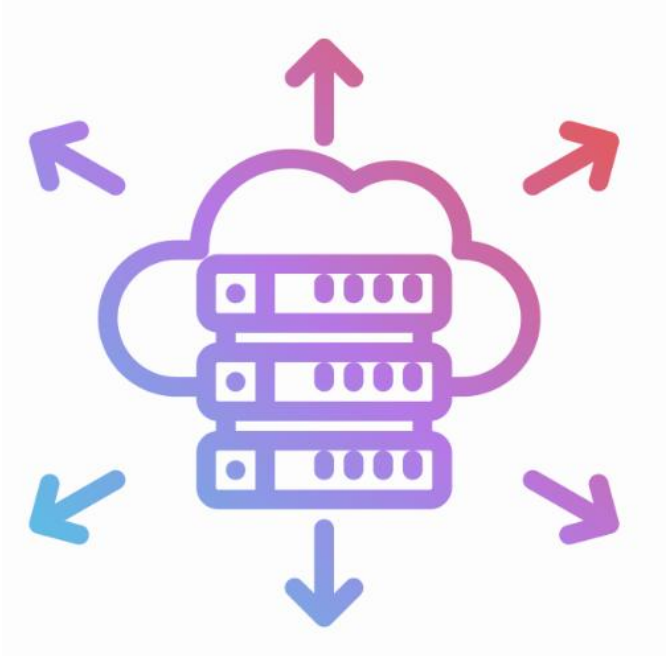
4. Чи може функція змінювати дані у таблицях?
- A) Так, завжди
- B) Ні, це заборонено
- C) Тільки при використанні TRIGGER
- D) Тільки в PostgreSQL

5. Як називається функція, яка повертає таблицю?
- A) Inline function
- B) Table-valued function
- C) Procedure
- D) Subselect

Ключі до тестів

1. Чим функція відрізняється від процедури?
Відповідь: Функція завжди повертає значення
2. Яка команда викликає збережену процедуру?
Відповідь: EXEC
3. Яке ключове слово використовується для повернення значення у функції?
Відповідь: RETURN
4. Чи може функція змінювати дані у таблицях?
Відповідь: Ні, це заборонено
5. Як називається функція, яка повертає таблицю?
Відповідь: Table-valued function

Змістовий модуль 3: Нереляційні моделі баз даних.



Тема 12. Концепція NoSQL. Нереляційні моделі БД

Мета теми
Ознайомити студентів з основами NoSQL — сучасного підходу до зберігання даних, що не базується на реляційній моделі. Навчити розрізняти основні типи NoSQL систем, розуміти переваги та сценарії використання.

12.1. Що таке NoSQL?
NoSQL (Not Only SQL) — це категорія систем управління базами даних, які **не використовують традиційну реляційну модель**. Вони призначені для зберігання, обробки та масштабування **нестрогоструктурованих або напівструктурованих даних**, таких як JSON-документи, графи, ключ-значення або стовпцеві формати.
Попри назву, NoSQL не означає повну відмову від SQL — багато сучасних NoSQL-систем підтримують гнучкі мови запитів, подібні до SQL, або навіть його підмножини.

Основні риси NoSQL-систем

1. **Гнучка схема (schema-less):**
2. Дані можуть зберігатися без заздалегідь визначеної структури. Наприклад, два документи в одній колекції можуть мати різні поля.
3. **Масштабування по горизонталі:**
4. NoSQL орієнтовані на розподілені архітектури й можуть масштабуватись шляхом додавання нових серверів.
5. **Висока доступність:**

6. Багато NoSQL-систем використовують реплікацію і шардинг, щоб забезпечити безперервну роботу навіть при збоях.
7. **Оптимізація під конкретні задачі:**
8. Наприклад, графові БД — для зв'язків, документні — для зберігання JSON, колонкові — для аналітики.
9. **Збереження великих обсягів даних:**
10. NoSQL підходить для Big Data, IoT, логування, рекомендованих систем тощо.

Причини появи NoSQL

- Зростання кількості **неструктурованих даних** (соцмережі, лог-файли, мультимедіа)
- Обмеження реляційних БД у **масштабуванні й гнучкості**
- Потреба у **швидкому доступі до даних** у глобально розподілених системах
- Вимоги **гнучкого проєктування** схеми на ранніх стадіях розробки стартапів або прототипів

NoSQL — це **сучасна відповідь на потреби гнучкості, масштабування та роботи з неструктурованими даними**. Вона не замінює реляційні бази, а **доповнює їх**, забезпечуючи нові можливості для розробників, особливо в умовах швидкозростаючих систем і великих обсягів даних.

12.2. Типи NoSQL баз даних

Сімейство NoSQL-баз даних включає декілька підходів до зберігання та організації даних, кожен з яких оптимізований під певний тип запитів, структуру інформації та навантаження. Основні типи NoSQL СУБД — це:

1. Документно-орієнтовані бази даних (Document Stores)

У таких БД дані зберігаються у вигляді **документів** — зазвичай у форматі JSON, BSON або XML. Кожен документ є самодостатнім — містить всю інформацію про об’єкт, включаючи вкладені структури (масиви, об’єкти). Відсутність фіксованої схеми дозволяє динамічно додавати або змінювати поля без модифікації структури таблиць.

Переваги:

- Гнучкість структури
- Підтримка вкладених даних
- Природна відповідність до форматів обміну (JSON)

Приклади: MongoDB, CouchDB, RavenDB

Коли використовувати: контент-менеджмент, каталоги товарів, мобільні додатки

2. Ключ-значення (Key-Value Stores)

Цей тип БД зберігає дані як **пари ключ–значення**, подібно до словників у мовах програмування. Ключ є унікальним і використовується для швидкого доступу до значення, яке може бути простим або складним (наприклад, JSON-рядок).

Переваги:

- Надзвичайно швидкий доступ до даних за ключем
- Легкість у масштабуванні по горизонталі

Приклади: Redis, Riak, Amazon DynamoDB

Коли використовувати: кешування, зберігання сесій, профілів користувачів, налаштувань

3. Колонкові бази даних (Wide-Column Stores)

Такі СУБД зберігають дані у вигляді **колонок**, а не рядків. Це дозволяє ефективно зберігати великі обсяги даних, де не всі поля присутні у кожному записі, і проводити аналітику над окремими колонками.

Переваги:

- Підходить для аналітичних запитів і OLAP-навантажень
- Висока продуктивність при роботі з окремими колонками

Приклади: Apache Cassandra, HBase, ScyllaDB

Коли використовувати: системи рекомендацій, логування, інтернет речей (IoT)

4. Графові бази даних (Graph Databases)

У графових СУБД дані моделюються як **вузли (nodes)** та **ребра (relationships)**. Вони зручно відображають структури з багатьма зв'язками — соціальні мережі, транспортні маршрути, ієрархії.

Переваги:

- Природне відображення складних зв'язків
- Висока ефективність запитів до зв'язків (на відміну від JOIN у реляційних БД)
- Можливість глибокого аналізу графів (centrality, path-finding)

Приклади: Neo4j, ArangoDB, OrientDB

Коли використовувати: аналіз соціальних мереж, побудова графів знань, fraud detection

5. Об'єктно-орієнтовані БД (рідше)

Дані представляються у вигляді об'єктів, подібно до об'єктів у мовах програмування (Java, Python). Підтримують інкапсуляцію, спадкування, зв'язки між класами. Цей підхід не такий популярний, але зустрічається у складних системах, де потрібна повна відповідність між об'єктною моделлю застосунку та базою даних.

Приклади: db4o, ObjectDB

Порівняння основних типів

Тип NoSQL БД	Схема	Підтримка зв'язків	Масштабування	Підходить для
Document	Гнучка	Через вкладення	Горизонтальне	Веб-додатки, CMS
Key-Value	Відсутня	Немає	Дуже хороше	Кеші, сеанси, швидкий доступ
Wide-Column	Часткова	Через розділення	Добре	Аналітика, Time Series
Graph	Гнучка	Перша роль	Обмежене	Соціальні мережі, графи знань

Типи NoSQL баз даних мають суттєві відмінності за структурою, принципами роботи та оптимізованими сценаріями використання. Правильний вибір типу бази даних залежить не лише від формату даних, а й від вимог до масштабування, швидкості доступу, складності зв'язків та частоти змін у схемі.

12.3. Реляційні vs Нереляційні БД

Параметр	Реляційна БД	NoSQL БД
Структура даних	Таблиці	Документи, графи, ключ-значення
Схема	Статична (фіксована)	Динамічна (schema-less)
Мова запитів	SQL	Власна (MongoQL, Cypher...)
Масштабованість	Вертикальна	Горизонтальна

Параметр	Реляційна БД	NoSQL БД
Узгодженість	ACID	Часто — Eventual consistency
Гнучкість	Слабка	Висока

12.4. Чому NoSQL популярні?

- **Гнучкість**: ви можете зберігати різні типи записів у одній колекції
- **Продуктивність**: оптимізовані під конкретні сценарії доступу
- **Масштабованість**: зручна обробка великих обсягів даних
- **Підтримка сучасних форматів**: JSON, BSON, XML
- **Підходять для Big Data і хмарних архітектур**

12.5. Де використовуються NoSQL БД?

NoSQL бази даних стали основою для багатьох сучасних цифрових рішень, особливо там, де потрібна **висока масштабованість, гнучка структура даних, швидкий доступ і обробка великих обсягів неструктурованої інформації**. Залежно від типу NoSQL БД (документні, графові, ключ-значення, колонкові), сфери їх застосування можуть бути різними — від кешування до аналітики в реальному часі.

1. Веб-застосунки та мобільні додатки

Документно-орієнтовані БД, як-от **MongoDB**, є популярним вибором для бекендів сучасних веб-сервісів та мобільних додатків, оскільки:

- підтримують гнучку схему;
- дозволяють зберігати вкладені структури (JSON);
- легко інтегруються з Node.js, Python, React Native тощо.

Приклад: системи управління контентом (CMS), форуми, блоги, онлайн-магазини.

2. Системи рекомендацій

NoSQL БД (особливо графові та колонкові) активно застосовуються для зберігання історії взаємодії користувачів з продуктами, обчислення подібностей та персоналізованих рекомендацій.

Приклади:

- Amazon використовує DynamoDB для зберігання інформації про перегляди й покупки.
- Netflix використовує Cassandra для зберігання телеметрії та даних переглядів.
- Spotify застосовує Neo4j для аналізу уподобань і побудови графів подібності.

3. Соціальні мережі та платформи обміну

Соціальні мережі з високим навантаженням потребують збереження мільярдів зв’язків (користувачі, пости, лайки, коментарі, підписки). Тут ідеально підходять **графові бази**, як-от **Neo4j**.

Приклади застосування:

- відображення структури друзів;
- пошук «хто знайомий з ким»;
- аналіз впливу, поширення контенту;
- виявлення спільнот.

4. Інтернет речей (IoT)

IoT-пристрої генерують величезні обсяги структурованих і напівструктурованих даних (температура, координати, стан). Для такого навантаження часто використовують **колонкові бази** (Cassandra, HBase), які дозволяють:

- масштабуватись по горизонталі;
- ефективно зберігати часоорієнтовані дані;
- швидко агрегувати та аналізувати показники.

5. Системи логування та моніторингу

Служби логування, як-от системи безпеки, журналювання подій або аудиту, потребують зберігання великої кількості однотипних записів. Для цього часто використовують:

- **ElasticSearch** (векторний пошук + аналітика)
- **MongoDB** (JSON-документи)
- **Cassandra** (швидке записування)

6. Фінансові технології (FinTech)

Фінансові додатки, особливо ті, які працюють з криптовалютами, API-інтерфейсами або мікросервісною архітектурою, потребують високої надійності та масштабованості.

Приклади застосування NoSQL у FinTech:

- кешування (Redis) для зменшення навантаження на основну БД;
- моніторинг транзакцій;
- fraud detection за допомогою графових зв'язків.

7. Кешування та управління сесіями

NoSQL системи типу **Key-Value** (як-от Redis, Memcached) чудово підходять для тимчасового зберігання:

- сесій користувача;
- токенів автентифікації;
- проміжних результатів запитів;
- часто запитуваних даних.

8. Big Data та аналітика в реальному часі

NoSQL часто використовується в архітектурах **data lake**, **stream processing** і **real-time analytics**. Наприклад:

- Apache Cassandra + Apache Spark для потокової аналітики;
- MongoDB Atlas + Charts для швидкого візуального аналізу;
- ClickHouse або Druid — для OLAP-навантажень.

9. Електронна комерція

E-commerce сервіси потребують збереження інформації про:

- товари (з багатьма характеристиками),
- динамічні кошики,
- історії покупок,
- профілі клієнтів.

Документні або key-value БД дозволяють забезпечити масштабованість та швидкий доступ, не потребуючи жорсткої фіксованої схеми.

NoSQL бази даних є **необхідним інструментом у сучасних інформаційних системах**, де ключову роль відіграють масштабованість, гнучкість структури даних, обробка великих обсягів інформації та ефективна робота з нестандартними структурами. Їх

застосування варто розглядати там, де класичні реляційні БД виявляються надто жорсткими або повільними.

Словник термінів до Теми 12

Термін	Пояснення
NoSQL	Нереляційна система управління базами даних
Document Store	База, що зберігає дані у вигляді JSON-документів
Key-Value	Модель, де дані зберігаються як пара ключ–значення
Column-Family	Модель, де дані групуються за стовпцями
Graph DB	Модель, що базується на графовому представленні даних
Schema-less	Гнучка структура без фіксованої схеми
Horizontal Scaling	Масштабування за рахунок додавання нових вузлів

Практичні завдання

1. Наведіть приклади задач, які краще вирішуються в:
    - Key-Value моделях
    - Документних БД
    - Графових БД
2. Побудуйте уявну модель товарного каталогу для e-commerce у MongoDB (у форматі JSON-документа)
3. Побудуйте приклад структури соціальної мережі у вигляді графа (вузли: користувачі, ребра: "дружить з")
4. Порівняйте структуру реляційної таблиці "Клієнти" з документом у MongoDB
5. Знайдіть приклад реально використаної NoSQL БД в IT-компанії, опишіть переваги для цього кейсу

Тести для самоперевірки

1. Яка з БД є документно-орієнтованою?

- A) Oracle
- B) MongoDB
- C) HBase
- D) MySQL

2. Що означає "schema-less"?

- A) Відсутність таблиць
- B) Відсутність первинних ключів
- C) Гнучка структура без заздалегідь заданої схеми
- D) Автоматичне створення таблиць

3. Для чого найкраще підходять графові бази?

- A) Зберігання ключ-значення
- B) Зв'язки між об'єктами
- C) Фінансові транзакції
- D) SQL-аналітика

4. Що означає горизонтальне масштабування?

- A) Оптимізація SQL-запитів
- B) Додавання нових колонок

- C) Додавання нових серверів
- D) Компресія таблиць

5. Яка з моделей є найпростішою серед NoSQL?
- A) Graph
- B) Column-Family
- C) Key-Value
- D) Object-Oriented

Ключі до тестів
1. Яка з БД є документно-орієнтованою?
Відповідь: MongoDB
Що означає "schema-less"?
Відповідь: Гнучка структура без заздалегідь заданої схеми
Для чого найкраще підходять графові бази?
Відповідь: Зв'язки між об'єктами
Що означає горизонтальне масштабування?
Відповідь: Додавання нових серверів
Яка з моделей є найпростішою серед NoSQL?
Відповідь: Key-Value

Тема 13. СУБД MongoDB

Мета теми
Познайомити студентів із принципами роботи MongoDB — найпопулярнішої документно-орієнтованої NoSQL СУБД. Навчити працювати з базами даних MongoDB, використовуючи команди CRUD, оператори фільтрації, агрегації та базові функції індексування.

14.1. Що таке MongoDB?
MongoDB — це популярна документно-орієнтована NoSQL система управління базами даних, яка зберігає дані у форматі **JSON-подібних документів** (фактично BSON — Binary JSON). Вона була розроблена компанією **MongoDB Inc.** і вперше представлена у 2009 році як база даних нового покоління, здатна ефективно працювати з великими обсягами неструктурованих або слабо структурованих даних.

Основні особливості MongoDB
1. **Документна модель зберігання**
   - Дані організовані у вигляді документів (об’єктів JSON), які зберігаються у колекціях (аналог таблиць).
   - Документи самодостатні, можуть мати вкладені структури, масиви, об’єкти.
2. **Схема на льоту (schema-less)**
   - Не потребує визначення чіткої структури таблиці заздалегідь.
   - Кожен документ у колекції може мати власну структуру.
3. **Горизонтальне масштабування (sharding)**
   - Підтримує розподілення даних по кількох вузлах кластера.
   - Дає змогу обробляти великі обсяги даних та запитів.

4. **Висока продуктивність**
    - Оптимізована для швидкого запису та читання.
    - Ідеальна для веб-додатків, які обробляють багато одночасних запитів.
5. **Мова запитів MongoDB**
    - Не SQL, а власна мова на основі JSON.
    - Наприклад, замість SELECT * FROM users WHERE age > 25, використовується:
    - db.users.find({ age: { $gt: 25 } })

Архітектура MongoDB

MongoDB складається з таких основних компонентів:

- **База даних (database)** → об'єднує колекції.
- **Колекція (collection)** → аналог таблиці, групує документи одного типу.
- **Документ (document)** → основна одиниця зберігання, представлена JSON/BSON.
- **Поле (_id)** → кожен документ має унікальний ідентифікатор (аналог первинного ключа).

Приклад документа MongoDB

```
{
  "_id": ObjectId("650123abc9b1f9a1a01a1abc"),
  "name": "Іван",
  "age": 28,
  "address": {
    "city": "Львів",
    "street": "Шевченка, 12"
  },
  "skills": ["Python", "MongoDB", "Docker"]
}
```

MongoDB у порівнянні з реляційною БД

Ознака	**MongoDB**	**Реляційна СУБД (наприклад, PostgreSQL)**
Структура	JSON-документи	Таблиці
Схема	Гнучка (необов'язкова)	Статична (чітко задана)
Зв'язки	Вкладення або посилання	JOIN між таблицями
Масштабування	Горизонтальне (sharding)	Переважно вертикальне
Ідеальне застосування	Веб-сервіси, Big Data, CMS	Класичні транзакційні системи

Коли обирати MongoDB?

MongoDB доцільно використовувати у випадках, коли:

- дані часто змінюють структуру (гнучка схема);
- потрібно швидко масштабуватися на декілька серверів;
- потрібна висока швидкість читання/запису;
- ви працюєте з JSON-даними або API;
- база повинна витримувати велику кількість одночасних користувачів.

MongoDB — це не просто NoSQL-база, а **сучасна, масштабована і гнучка платформа для зберігання даних**, яка прекрасно підходить для розробки хмарних, веб- і мобільних застосунків.

14.2. Основні поняття

Термін	Пояснення
Database	Колекція колекцій
Collection	Аналог таблиці в реляційних БД
Document	Один запис (аналог рядка), зберігається у BSON
Field	Атрибут документа (аналог стовпця)
ObjectId	Унікальний ідентифікатор документа
Index	Об’єкт, який покращує швидкість пошуку
Schema-less	Колекція може містити документи з різною структурою

14.3. Основні операції (CRUD)

Створення (Insert)

```
db.students.insertOne({
  name: "Olena",
  group: "PM-21",
  grades: [90, 85, 95]
});
```

Читання (Find)

```
db.students.find({ group: "PM-21" });
db.students.find({ grades: { $gt: 90 } });
```

Оновлення (Update)

```
db.students.updateOne(
  { name: "Olena" },
  { $set: { group: "PM-22" } }
);
```

Видалення (Delete)

```
db.students.deleteOne({ name: "Olena" });
```

14.4. Оператори фільтрації та логіки

- $gt, $lt, $gte, $lte — порівняння чисел
- $in, $nin — множинна відповідність
- $and, $or, $not — логічні оператори
- $exists, $type — перевірка наявності поля та типу

Приклад:

```
db.products.find({
  $and: [
    { price: { $gt: 100 } },
    { category: "Electronics" }
  ]
});
```

14.5. Агрегація

MongoDB дозволяє виконувати багатоступеневу обробку даних через **Aggregation Pipeline**.

Приклад:

```
db.orders.aggregate([
  { $match: { status: "delivered" } },
  { $group: { _id: "$customerId", total: { $sum: "$amount" }
} },
  { $sort: { total: -1 } }
]);
```

Етапи:

- $match — фільтрація
- $group — агрегація
- $project — вибір потрібних полів
- $sort — сортування
- $limit — обмеження результатів

14.6. Індекси

```
db.students.createIndex({ name: 1 });
```

- 1 — індекс у порядку зростання
- -1 — у порядку спадання
- Індекси значно прискорюють пошук, особливо у великих колекціях

Словник термінів до Теми 14

Термін	Пояснення
insertOne, insertMany	Додавання документів
find	Пошук документів

Термін	Пояснення
$set	Оператор оновлення поля
$match, $group, $project	Команди агрегування
createIndex	Створення індексу
BSON	Двоїчний аналог JSON (Binary JSON)
Aggregation Pipeline	Послідовна обробка даних по етапах
Schema-less	Гнучка структура документів без фіксованої схеми

Практичні завдання

1. Створіть колекцію students та додайте 3 записи з полями: name, group, grades (масив чисел).
2. Напишіть запит, який виведе студентів із групи РМ-21, у яких є оцінки > 90.
3. Оновіть групу певного студента на нову, використовуючи $set.
4. Видаліть усіх студентів, які не мають оцінок (grades не існує).
5. Створіть індекс по полю group та перевірте його вплив на продуктивність запиту.
6. Порахуйте середній бал кожного студента за допомогою $project і $avg.

Тести для самоперевірки

1. Який формат використовує MongoDB для зберігання документів?

- A) CSV
- B) XML
- C) BSON
- D) YAML

2. Як виконується фільтрація в MongoDB?

- A) WHERE
- B) MATCH
- C) $match
- D) HAVING

3. Що робить оператор $set?

- A) Видаляє поле
- B) Оновлює або створює поле
- C) Повертає лише зазначені поля
- D) Застосовується лише у find

4. Яка команда створює індекс?

- A) create_table
- B) createIndex
- C) addIndex
- D) ensureIndex

5. Що робить $group у агрегації?

- A) Розділяє документи
- B) Групує документи за вказаним ключем
- C) Створює індекси
- D) Додає поле до документа

Ключі до тестів

1. Який формат використовує MongoDB для зберігання документів?

Відповідь: BSON

6. Як виконується фільтрація в MongoDB?

Відповідь: $match

7. Що робить оператор $set?

Відповідь: Оновлює або створює поле

8. Яка команда створює індекс?

Відповідь: createIndex

9. Що робить $group у агрегації?

Відповідь: Групує документи за вказаним ключем

Тема 14. Створення web-додатку з підключенням до БД

Мета теми

Навчити студентів реалізовувати базову архітектуру веб-застосунку, який взаємодіє з базою даних. Продемонструвати, як здійснюється підключення, зчитування, запис і оновлення даних з бази через веб-інтерфейс.

14.1. Компоненти web-застосунку

Сучасний **web-застосунок** — це програмне рішення, яке працює через веб-браузер і взаємодіє з користувачем у реальному часі. Його архітектура зазвичай базується на **клієнт-серверній моделі**, яка включає в себе низку взаємозалежних компонентів, що забезпечують функціональність, продуктивність і безпеку сервісу.

Основні компоненти web-застосунку

1. **Клієнтська частина (frontend)**
2. Це інтерфейс, з яким безпосередньо взаємодіє користувач у веб-браузері.

Складається з:

- HTML — структура веб-сторінки.
- CSS — стилізація елементів.
- JavaScript — взаємодія, динаміка та логіка на стороні клієнта.
- Фреймворки: React, Angular, Vue.js тощо.

Відповідає за:

- рендеринг вмісту сторінки,
- обробку подій користувача,
- надсилання запитів до сервера.

3. **Серверна частина (backend)**
4. Серверна логіка виконується на віддаленому сервері. Саме тут обробляються запити, бізнес-логіка та взаємодія з базою даних.

Складається з:

- Мов програмування: Python (Django, Flask), JavaScript (Node.js), PHP, Java, C# (.NET)
- Сервери: Apache, Nginx
- REST або GraphQL API для обміну даними з клієнтом.

Відповідає за:

- обробку HTTP-запитів,
- авторизацію та автентифікацію,
- роботу з базами даних,

- виконання логіки застосунку.

5. **База даних (Database)**
6. Відповідає за збереження і доступ до даних.

Типи:

- Реляційні: PostgreSQL, MySQL, SQL Server
- NoSQL: MongoDB, Redis, Cassandra

База даних обробляє:

- запити на збереження, оновлення та отримання даних,
- транзакції,
- забезпечення цілісності даних.

7. **API (Application Programming Interface)**
8. Це інтерфейс взаємодії між клієнтом і сервером. Дає змогу клієнтській частині отримувати або надсилати дані.
    - REST API (через HTTP-методи GET, POST, PUT, DELETE)
    - GraphQL — альтернатива REST, дозволяє клієнту визначати структуру відповіді.
9. **Система маршрутизації (Routing)**
10. Визначає, які дії слід виконати при отриманні певного HTTP-запиту.

Наприклад:

- GET /products → показати список товарів
- POST /login → обробити вхід користувача

11. **Сховище статичних файлів**
12. Сюди входять зображення, стилі, скрипти, які не змінюються динамічно і доступні напряму з сервера або через CDN.
13. **Система автентифікації та авторизації**
    - Автентифікація: перевірка, хто користувач (логін/пароль, OAuth, JWT).
    - Авторизація: перевірка прав доступу (чи може цей користувач побачити/змінити ресурс).
14. **Інфраструктура (хостинг, деплоймент, CI/CD)**
15. Включає:
    - хостинг (Heroku, AWS, Azure, Render),
    - контейнери (Docker),
    - інструменти CI/CD для автоматизації тестування і розгортання.

Як все працює разом?

1. Користувач відкриває сайт у браузері (frontend).
2. Надсилається HTTP-запит до сервера (backend).
3. Сервер обробляє запит, звертається до бази даних (DB).
4. Дані повертаються через API на клієнтську частину.
5. Браузер відображає отриману інформацію.

Ця структура — основа більшості сучасних web-додатків: від інтернет-магазинів до SaaS-платформ і чат-ботів.

14.2. Приклад: веб-застосунок зі зберіганням студентів у MongoDB

База даних: MongoDB
Колекція: students

Backend: Python + Flask

Приклад створення застосунку з використанням бібліотеки pymongo.

```
from flask import Flask, request, jsonify
from pymongo import MongoClient

app = Flask(__name__)
client = MongoClient("mongodb://localhost:27017/")
db = client["university"]
students = db["students"]

@app.route("/students", methods=["GET"])
def get_students():
    data = list(students.find({}, {"_id": 0}))
    return jsonify(data)

@app.route("/students", methods=["POST"])
def add_student():
    student = request.json
    students.insert_one(student)
    return jsonify({"msg": "Student added"}), 201

if __name__ == "__main__":
    app.run(debug=True)
```

Тестування:
Можна надсилати запити через Postman або браузер:

- GET /students → отримати всіх студентів
- POST /students з JSON-тілом → додати нового

14.3. Структура збережених документів

```
{
  "name": "Ірина Степаненко",
  "group": "РМ-22",
  "grades": [90, 95, 88]
}
```

MongoDB не потребує створення схеми. Поля можна варіювати.

14.4. Варіант на основі SQL (SQLite + Flask)

Для демонстрації принципу з SQL можна замінити MongoDB на SQLite:

```
import sqlite3

conn = sqlite3.connect("students.db")
c = conn.cursor()
c.execute('''CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    group_name TEXT
)''')
conn.commit()
```

Після цього створюються маршрути, що працюють із SQL-запитами через sqlite3.

14.5. Типові задачі web-застосунку з БД

- Додавання нових записів через форму
- Виведення списку всіх записів (таблиця)
- Редагування або оновлення запису
- Видалення запису
- Побудова фільтрації, пошуку, сортування
- Автоматична валідація даних

Словник термінів до Теми 14

Термін	Пояснення
API	Інтерфейс прикладного програмування, через який frontend звертається до backend
CRUD	Набір операцій: створити, прочитати, оновити, видалити
Route	Шлях на сервері, який обробляє конкретний запит
POST / GET	HTTP-методи для надсилання і отримання даних
JSON	Формат передачі даних між клієнтом і сервером
ORM	Object Relational Mapping – бібліотека для зручної роботи з SQL-БД (наприклад, SQLAlchemy)

Практичні завдання

1. Встановіть MongoDB локально або використайте MongoDB Atlas (хмара).
2. Створіть Python-застосунок із Flask, який:
    - додає студентів
    - виводить список студентів
    - дозволяє видаляти студента за ім’ям
3. Розширте модель, додавши поле average_grade, яке розраховується автоматично.
4. Додайте фільтрацію по групі через параметри URL (/students?group=PM-22).
5. Замість MongoDB спробуйте реалізувати аналогічну систему з SQLite.
6. Побудуйте простий HTML-інтерфейс, який взаємодіє з API (через JavaScript fetch або axios).

Тести для самоперевірки

1. Який формат зазвичай використовується для передачі даних між frontend і backend?

- A) CSV
- B) XML
- C) JSON
- D) PDF

2. Що таке CRUD?

- A) Алгоритм шифрування
- B) Тип мережевого з’єднання
- C) Набір базових операцій із даними
- D) Метод кешування

3. Який метод HTTP використовується для додавання даних?

- A) GET
- B) DELETE

- C) POST
- D) SELECT

4. Що таке маршрут (route) у Flask?
- A) Назва таблиці
- B) Функція, що виконує агрегацію
- C) Шлях, який обробляє запити
- D) Запис у базі даних

5. Чим відрізняється MongoDB від SQL у контексті веб-застосунку?
- A) MongoDB використовує XML
- B) MongoDB жорстко типізована
- C) MongoDB не потребує фіксованої схеми
- D) У MongoDB не можна видалити запис

Ключі до тестів

1. Який формат зазвичай використовується для передачі даних між frontend і backend?
Відповідь: JSON
6. Що таке CRUD?
Відповідь: Набір базових операцій із даними
7. Який метод HTTP використовується для додавання даних?
Відповідь: POST
8. Що таке маршрут (route) у Flask?
Відповідь: Шлях, який обробляє запити
9. Чим відрізняється MongoDB від SQL у контексті веб-застосунку?
Відповідь: MongoDB не потребує фіксованої схеми

Завдання до лабораторних робіт

Лабораторна робота 1: «Вибірка даних». Робота з базою даних Northwind

Сформулюйте SQL-запит для кожного з наведених завдань. Скопіюйте запит і результат (скріншот) у окремий .doc-файл. Завантажте файл у репозиторій.

1. Показати всю інформацію про працівника з ID = 8.

```
SELECT *
FROM employees e
WHERE employee_id = 8;
```

2. Показати імена та прізвища працівників, які працюють у Лондоні.
3. Показати імена та прізвища працівників, чиє ім’я починається з літери "A".
4. Показати імена, прізвища та вік працівників, яким більше ніж 55 років. Результати впорядкувати за прізвищем.
5. Обчислити кількість працівників із Лондона.
6. Обчислити максимальний, мінімальний і середній вік працівників із Лондона.
7. Обчислити максимальний, мінімальний і середній вік працівників для кожного міста.
8. Показати список міст, у яких середній вік працівників перевищує 60 років (також вивести середній вік).
9. Показати імена та прізвища найстаршого(-их) працівника(-ів). Використайте підзапит.
10. Показати імена, прізвища та вік трьох найстарших працівників.
11. Показати список усіх міст, з яких родом працівники.
12. Показати імена, прізвища та дати народження працівників, які святкують день народження цього місяця.
13. Показати імена та прізвища працівників, які обслуговували замовлення, доставлені в Мадрид.
14. Показати імена, прізвища працівників та кількість замовлень, які кожен із них обробив у 1997 році (використовуйте LEFT JOIN).
15. Показати імена, прізвища працівників та кількість замовлень, які кожен із них обробив у 1997 році (використовуйте підзапит).
16. Показати імена, прізвища працівників та кількість їхніх замовлень, доставлених **після** вказаної дати доставки у 1997 році (LEFT JOIN).
17. Показати кількість замовлень, зроблених кожним клієнтом із Франції.
18. Показати імена клієнтів із Франції, які зробили більше одного замовлення (використовуйте групування).
19. Показати імена клієнтів із Франції, які зробили більше одного замовлення (використовуйте підзапит).
20. Показати імена клієнтів, які замовляли продукт «Tofu» (використовуйте підзапит).
21. *Показати імена клієнтів, які замовляли продукт «Tofu», а також загальну кількість замовленого товару і загальну суму за цей продукт.
22. *Показати імена клієнтів із Франції, які замовляли товари **не французького** походження (LEFT JOIN).

23. *Показати імена клієнтів із Франції, які замовляли товари **не французького** походження (підзапит).
24. *Показати імена клієнтів із Франції, які замовляли **французькі** товари.
25. *Показати загальну суму замовлень, обчислену для кожної країни клієнтів.
26. *Показати загальні суми замовлень для кожної країни клієнтів окремо для внутрішніх і зовнішніх товарів (наприклад: "France – французькі товари – не французькі товари").
27. *Показати список категорій товарів разом із загальними сумами замовлень, обчисленими для кожної категорії за 1997 рік.
28. *Показати список назв товарів, їхні поточні ціни та історію цін із замовлень (вивести: назва товару – поточна ціна – історична ціна). Усунути дублікати. Якщо товар ніколи не замовляли, вивести: назва товару – поточна ціна – NULL. Впорядкувати за назвою товару.
29. *Показати імена працівників разом із іменами їхніх керівників (використати LEFT JOIN до тієї ж таблиці).
30. *Показати список міст, звідки родом працівники, клієнти та куди було здійснено доставку замовлень. Усунути дублікати.

Лабораторна робота 2: Проєктування тематичних баз даних

Завдання:

Варіанти завдань для розробки тематичних баз даних під час виконання практичних робіт наведені у таблиці.

№	Назва бази даних та опис предметної області	Мінімальний набір базових таблиць	Мінімальний набір полів таблиць
1	**База даних "Готель"** Клієнти готелю бронюють номери на певні періоди часу (дні). Номери мають кількість місць та тип (економ, стандарт, люкс тощо), від яких залежить вартість проживання. Для адміністрації готелю автоматично формується розклад заселення/виселення та рейтинги популярності номерів.	Клієнти	Ім’я клієнта
		Номери	Кількість осіб, які перебуватимуть з клієнтом Паспортні дані клієнта Номер кімнати Кількість місць у кімнаті Тип кімнати Дата і час заселення/виселення
2	**База даних "Лікарня"**		

	Лікарня складається з відділень із власною назвою, має штат лікарів, один з яких є завідувачем відділення. Пацієнтів приймають із певним діагнозом. Кожному призначається лікуючий лікар з цього відділення. За потреби залучаються інші спеціалісти, які також ставлять діагнози. Під час виписки формується остаточний діагноз.	Пацієнти	Ім’я пацієнта
		Лікарі	Дата народження пацієнта
		Відділення	Назва відділення
			Ім’я лікаря
			Спеціальність лікаря
			Діагноз
			Палата
			Дата початку лікування
			Дата виписки
3	**База даних "Музичний каталог"**		
	Онлайн-каталог дозволяє зареєстрованим користувачам публікувати власну музику. Композиції мають назву, авторів, жанр, формат файлу, теги. До кожної композиції можна додавати матеріали (текст, ноти, логотип). Коментарі можуть залишати лише зареєстровані користувачі. Автор може відповідати, утворюючи дерево обговорення.	Користувачі	Ім’я користувача
		Композиції	Контактна інформація
		Коментарі	Назва композиції
			Жанр
			Текст коментаря
4	**База даних "Інтернет-магазин"**		
	Магазин продає товари зареєстрованим клієнтам. Товари мають категорію, характеристики, ціну, одиниці виміру.	Товари	Назва товару

	Клієнт формує кошик, обирає кількість і адресу доставки.	Клієнти Кошик	Категорія товару Характеристики Одиниця виміру Ціна за одиницю Дані клієнта Дата покупки Кількість обраних одиниць Адреса доставки
5	**База даних "Прокат авто"** Компанія надає дані про авто в оренду (виробник, марка, місця, колір, ціна) і фіксує угоди з клієнтами (термін, авто).	Авто Клієнти Угоди	Номер авто Виробник Марка Тип Кількість місць Колір Вартість оренди на день Дані клієнта Термін оренди
6	**База даних "Турфірма"** Агентство організовує відпочинок на курортах (тип, рівень зірок, країна, місце). Клієнт обирає курорт, агент підписує договір і отримує комісію.	Курорти Клієнти Агенти	Назва курорту Тип курорту Рівень якості Місце розташування

			Пропозиції щодо вартості Особисті дані клієнта Дані агента Відсоток комісії агента Дата підписання договору Період відпочинку
7	**База даних "Інтернет-розсилка"** Видавництво публікує нотатки. Клієнти отримують повідомлення про нові публікації за ключовими словами.	Клієнти Нотатки	Дані клієнта Тематика Назва нотатки Анотація Автори Вартість підписки Ключові слова
8	**База даних "ФотоСервіс"** Сервіс (типу Instagram) дозволяє публікувати фото з коментарями, які можуть бути публічними або приватними.	Клієнти Фото	Особисті дані клієнта Друзі клієнта Назва фото Файл фото Теги Коментарі
9	**База даних "Агентство нерухомості"** Агентство веде базу продавців і покупців. Об'єкти характеризуються адресою, станом, вартістю, згодою власника. Клієнти мають обмеження по	Нерухомість	Адреса

	бюджету та вимоги. Угода оформлюється агентом.		
		Клієнти	Тип об'єкта
		Агенти	ПІБ власника
		Договори	Площа, кількість кімнат
			Поверх, поверховість
			Житлова, кухня
			Тип санвузла
			Вартість
			ПІБ клієнта
			Бюджет
			Контакти
			Номер договору
			Сума
			Дата
			Агент
10	**База даних "Страхування"** Договір укладається між компанією та клієнтом, містить тип, суму, премію, терміни, вартість. Агент отримує комісію. Звіти — за активністю агентів, доходами, популярністю послуг.	Клієнти	ПІБ клієнта
		Агенти	Дані клієнта
		Договори	ПІБ агента
			Номер договору
			Страхова сума
			Премія
			Дата укладання
			Термін дії

			Платіж (щомісячний або разовий) Комісія
11	**База даних "Екзамени"** Містить інформацію про студентів, групи, предмети. Студенти складають кілька екзаменів. Створюється розклад та звіти.	Студенти Викладачі Предмети	ПІБ студента Номер залікової Номер групи ПІБ викладача Посада Назва предмета Кількість годин Дата екзамену Оцінка
12	**База даних "Розклад"** Під час формування розкладу враховується завантаженість аудиторій, груп, тип занять. Формуються версії розкладу.	Групи Викладачі Предмети Аудиторії	Номер групи Спеціальність Факультет Предмет Тип заняття День тижня Час Частота (щотижня, чисельник/знаменник) Номер аудиторії Корпус

			Тип Кількість місць
13	**База даних "Абітурієнт"** Абітурієнт подає документи на спеціальності. Оригінал може бути лише до однієї. Є звіти по рейтингу й популярності.	Абітурієнти Спеціальності Документи	ПІБ абітурієнта Контакти Назва документа, серія, номер, дата, орган Бали ЗНО Назва спеціальності Факультет
14	**База даних "Персонал"** Штат організації формується за підрозділами. Кожен працівник обіймає посаду із зазначеною ставкою, надбавками. Звіти — за вакансіями та складом.	Підрозділи Працівники Штатні одиниці	Назва підрозділу Тип Надбавка за умови Посада Оклад Надбавка за графік ПІБ працівника Табельний номер Стаж Дата початку Освіта
15	**База даних "Науковий журнал"**		

	Автори надсилають статті з анотаціями, ключовими словами. Призначаються рецензенти. Адміністрація отримує звіти. Автори — відгуки анонімно.	Автори Статті Рецензенти	ПІБ і контакти автора Назва статті Анотації 3 мовами Ключові слова Посилання на файл ПІБ рецензента Тематика Оцінки Відгук для редакції Коментарі для автора

Коментар: виконані завдання до кожної теми слід зберігати у відповідній директорії.

Лабораторна робота 3: Створення структури БД (DDL)

Завдання:

1. Створіть SQL-інструкції для створення таблиць відповідно до моделі даних вашої предметної області.
2. Визначте первинні та зовнішні ключі.
3. Для атрибутів, які мають обмеження (наприклад, тип, діапазон значень), додайте обмеження CHECK.
4. Установіть NOT NULL для обов’язкових полів.
5. Додайте принаймні одну таблицю із зв'язком багато-до-багатьох (якщо це логічно).

Приклад для “Готель”

```
CREATE TABLE Clients (
    client_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    passport_number VARCHAR(20) UNIQUE NOT NULL,
    num_people INT CHECK (num_people > 0)
);
```

Лабораторна робота 4: Основні DML-операції (INSERT, UPDATE, DELETE, SELECT)

Завдання:

1. Заповніть створені таблиці тестовими даними (INSERT).
2. Змініть дані для окремих записів (UPDATE).
3. Видаліть обрані записи за умовами (DELETE).
4. Виконайте прості SELECT-запити: вибірка всіх записів, фільтрація, сортування.
5. Побудуйте запити з умовами WHERE, BETWEEN, LIKE.

Приклад для "Музичний каталог"

- Додайте щонайменше 3 композиції різних авторів.
- Оновіть жанр однієї з композицій.
- Видаліть коментарі, які містять певне ключове слово.
- Виведіть список усіх композицій жанру 'Jazz'.

Лабораторна робота 5: Агрегування та групування. JOIN-запити

Завдання:

1. Обчисліть загальну кількість записів у певній таблиці.
2. Обчисліть середні значення, максимум, мінімум.
3. Побудуйте запити з GROUP BY та HAVING.
4. Порахуйте кількість записів у кожній категорії.
5. Виконайте запити з INNER JOIN між таблицями.
6. Побудуйте LEFT JOIN і RIGHT JOIN для виводу повної інформації.
7. Використайте JOIN між 3 таблицями.
8. Додайте підзапит (subquery).

Приклад для "Інтернет-магазин"

- Визначте середню ціну товарів кожної категорії.
- Знайдіть категорію з найбільшою кількістю товарів.
- Порахуйте кількість замовлень, зроблених кожним клієнтом.

Приклад для "Прокат авто"

- Виведіть список клієнтів та інформацію про авто, які вони орендували.
- Покажіть усі авто, які ніколи не орендувались.
- Виведіть список марок авто, що були орендовані більше 5 разів.

Лабораторна робота 6: Керування обмеженнями та доступом

Завдання:

1. Створіть користувача в СУБД і надайте йому обмежені права доступу (наприклад, тільки SELECT).
2. Створіть тригер, який автоматично заповнює дату останньої модифікації.
3. Додайте перевірку цілісності за допомогою FOREIGN KEY та CHECK.
4. Встановіть транзакцію з ролбеком при помилці.

Приклад для "Страхування"

- Створіть тригер, що обчислює комісію агента при додаванні договору.
- Додайте обмеження: страхова сума має бути > 1000.

Лабораторна робота 7: Резервне копіювання, відновлення БД та оптимізація запитів

Завдання:

1. Створіть повну резервну копію вашої БД.
2. Імітуйте втрату даних (видаліть частину записів).
3. Виконайте відновлення з резервної копії.
4. Побудуйте складний SELECT-запит із JOIN, фільтрацією, агрегацією.
5. Проаналізуйте план виконання запиту (через EXPLAIN, ANALYZE).
6. Додайте індекси на поля, які часто використовуються у WHERE, ORDER BY.
7. Повторно проаналізуйте продуктивність запиту після оптимізації.

Приклад для БД «Лікарня»:

- Резервне копіювання таблиць Patients, Doctors, Departments.
- Видалення випадкових записів з Patients.
- Побудова запиту зі з'єднанням Patients, Doctors, Departments.
- Додавання індексу на specialty, start_treatment_date.

Лабораторна робота 8: Безпека, права доступу, звіти, експорт

Завдання:

1. Створіть трьох користувачів БД із різними рівнями прав (адміністратор, редактор, переглядач).
2. Наддайте кожному користувачеві доступ лише до потрібних таблиць і дій.
3. Створіть VIEW для користувача-переглядача.
4. Сформуйте агрегований звіт SQL-запитом: загальні показники (сума, середнє, кількість).
5. Експортуйте результат у .csv або .xlsx.
6. Побудуйте графік або таблицю за отриманими даними (у Excel / Google Sheets / Python).
7. Опишіть можливі способи інтеграції вашої БД з веб або BI-додатками.

Приклад для БД «ФотоСервіс»:

- Користувач moderator: доступ до Photos та Comments.
- Звіт: кількість фото кожного користувача.
- Експорт у Excel, побудова кругової діаграми за кількістю фото.

Лабораторна робота 9: Створення та робота з NoSQL БД (MongoDB)

Завдання:

1. Створіть структуру бази даних для вашої предметної області у форматі MongoDB (JSON-документи).
2. Додайте дані до колекцій insertMany() з вкладеними об'єктами або масивами.
3. Виконайте пошукові запити за різними критеріями (find, findOne, з умовами на вкладені поля).
4. Використайте агрегування: aggregate, $group, $match, $sort.

5. Створіть індекси для полів і поясніть їх ефективність.
6. Побудуйте звіт на базі запитів із MongoDB — у вигляді JSON/таблиці.
7. Порівняйте підхід MongoDB до збереження даних з реляційною БД у рамках вашої теми.

Приклад для БД «Інтернет-магазин»:

Колекція orders:

```
{
  "order_id": 123,
  "customer": {
    "id": "c001",
    "name": "Іван Петренко"
  },
  "items": [
    { "product": "Ноутбук", "price": 1200, "qty": 1 },
    { "product": "Мишка", "price": 25, "qty": 2 }
  ],
  "total": 1250
}
```

Агрегатний запит:

```
db.orders.aggregate([
  { $unwind: "$items" },
  { $group: { _id: "$customer.name", totalSpent: { $sum: { $multiply: ["$items.price", "$items.qty"] } } } }
])
```

Додаткові теми для поглибленого вивчення

Розподілені бази даних (Distributed Databases)

Розподілена база даних (Distributed Database, DDB) — це сукупність логічно пов'язаних баз даних, розміщених на різних вузлах комп'ютерної мережі, які працюють як єдина цілісна система для користувача.
Система управління РБД забезпечує прозорість розподілу та створює ілюзію централізованої БД.

Архітектура
Розподілена система складається з:

- **вузлів прийому запитів (coordinator)—** відповідають за прийом і маршрутизацію запитів;
- **вузлів зберігання—** фактично зберігають дані. Вузли можуть бути незалежними машинами з власними ОС.

Управління включає:

- **реплікацію** (копіювання даних),
- **фрагментацію (шардування)** — поділ таблиць чи даних,
- **дотримання ACID або доступності** згідно з теоремою CAP.

Переваги та виклики

Переваги	Виклики
Горизонтальна масштабованість та висока доступність	Складність забезпечення консистентності даних
Резистентність до відмов вузлів або мережевих розривів	Обробка розподілених транзакцій та узгодження (CAP теорема)

Переваги	Виклики
Зниження затримки доступу до даних (geo-distributed)	Складне адміністрування (каталоги, маршрутизація, моніторинг)

Класифікації
Гомогенні vs. Гетерогенні РБД:

- **Гомогенні** — всі вузли мають однакову структуру, керівництво СУБД, ОС, модель даних.
- **Гетерогенні** — кожен вузол може мати різні СУБД, формат, що вимагає складного інтеграційного рівня.

Distributed SQL — новітній підхід у реляційних РБД
Distributed SQL комбінує традиційну реляційну структуру з горизонтальною масштабованістю, високою доступністю, консистентністю, транзакційністю та SQL-інтерфейсом. Приклади: CockroachDB, YugabyteDB, TiDB.

Приклади сучасних розподілених систем БД:

- **Apache Cassandra** — wide-column NoSQL, орієнтований на доступність і масштабування, AP-тип (за CAP), підтримує асинхронну реплікацію.
- **Apache HBase** — CP-тип NoSQL, побудований поверх HDFS із Bigtable-подібним API.
- **Apache Ignite** — in-memory distributed DBMS з підтримкою SQL, ACID транзакцій, кешування та обробкою MapReduce.
- **MySQL Cluster (NDB)** — shared-nothing кластер із автоматичним шардуванням та реплікацією, ACID, розподілені транзакції.
- **Oracle Globally Distributed Database** — логічна база, яка автоматично реплікує дані по регіонах, забезпечує високий рівень доступності та продуктивності.

Словник термінів

Термін	Значення
Distributed Database (РБД)	БД, логічно цілісна, фізично розподілена
Реплікація	Метод дублювання даних між вузлами
Фрагментація / Шардування	Розділення даних на частини між вузлами
Homogeneous / Heterogeneous	Типова vs різнорідна архітектура
Distributed SQL	Масштабована реляційна БД із SQL-інтерфейсом
CAP-теорема	Парадокс, що описує компроміси між консистентністю, доступністю і стійкістю до розривів
ACID	Атомарність, консистентність, ізольованість, стійкість

Усунення затримок між транзакціями й аналітикою — HTAP (Hybrid Transactional/Analytical Processing)

Мета теми
Ознайомити студентів із сучасною концепцією HTAP (Hybrid Transactional/Analytical Processing), яка дозволяє поєднати в межах однієї СУБД як транзакційні (OLTP), так і аналітичні (OLAP) навантаження. Розглянути принципи, переваги, архітектури та приклади реалізацій.

1. Що таке HTAP?
HTAP — це архітектурна модель баз даних, яка забезпечує **одночасну** обробку:

- транзакцій у режимі реального часу (OLTP),
- аналітичних запитів на тих самих даних (OLAP).

На відміну від класичної парадигми ETL (Extract-Transform-Load), яка передбачає копіювання даних із транзакційної бази у сховище для аналітики, HTAP мінімізує або взагалі усуває ці затримки.
Ключовий принцип: **одне джерело правди (single source of truth)** для обох типів навантаження.

2. Чому це важливо?

Класична модель (OLTP → ETL → OLAP)	HTAP
Затримки в годинах або днях	Аналітика в реальному часі
Потреба у складній ETL-інфраструктурі	Дані не дублюються
Паралельне адміністрування OLTP/OLAP	Централізоване управління
Обмежена адаптивність до змін	Висока гнучкість і адаптивність

3. Архітектурні підходи до HTAP
3.1. Shared-storage підхід
OLTP і OLAP працюють на одному сховищі, але використовують різні двигуни запитів. Наприклад: **SAP HANA**, **Oracle ADW**.
3.2. Двоєдині рушії (dual-engine architecture)

- **Окремі компоненти для OLTP і OLAP**
- Спільна пам’ять або проміжний шар синхронізації
- Приклад: **TiDB** — об’єднує TiKV (OLTP) і TiFlash (OLAP)

3.3. Розподілена обробка в оперативній пам’яті (in-memory)

- Дані зберігаються в RAM, що знижує час доступу
- Приклад: **MemSQL / SingleStore**

4. Технології, що підтримують HTAP

Назва	Тип	Особливості
TiDB	Розподілений SQL	OLTP на TiKV, OLAP на TiFlash
SingleStore (MemSQL)	In-memory SQL	Обробка потоків і запитів у RAM
SAP HANA	In-memory column-store	OLTP і OLAP на єдиному рушії

Назва	Тип	Особливості
Oracle Autonomous Database	Cloud-native	Автоматичне масштабування
CockroachDB	Distributed SQL	HTAP-підхід з ACID транзакціями

Словник термінів

Термін	Пояснення
HTAP	Гібридна обробка OLTP і OLAP
OLTP	Online Transaction Processing — операційна БД
OLAP	Online Analytical Processing — аналітика
ETL	Extract, Transform, Load — процес переносу даних
TiFlash	Колонкове сховище для аналітичних запитів у TiDB
In-Memory	Архітектура зберігання даних у RAM
Single Source of Truth	Єдине джерело правдивих даних

Практичне завдання

1. Ознайомтесь із архітектурою TiDB на офіційному сайті (https://docs.pingcap.com/)
2. Порівняйте можливості HTAP-обробки у TiDB та Oracle ADW.
3. Опишіть приклад ситуації, коли HTAP дозволить приймати бізнес-рішення швидше, ніж у класичній ETL-схемі.
4. Побудуйте логічну схему HTAP-системи для онлайн-магазину, що потребує як реєстрації замовлень, так і аналітики продажів у реальному часі.

AI-автоматизація в управлінні БД та автономні бази даних

Мета теми

Надати студентам розуміння того, як сучасні СУБД використовують алгоритми штучного інтелекту для автоматизації адміністрування, оптимізації та самоуправління. Ознайомити з концепцією автономних баз даних як майбутнім стандартом в IT-інфраструктурі.

1. Що таке AI-автоматизація в СУБД?

AI-автоматизація в СУБД — це застосування методів машинного навчання (ML), обробки природної мови (NLP), евристик та предиктивної аналітики для:

- автоматичного налаштування параметрів БД (self-tuning),
- прогнозування навантаження,
- виявлення аномалій і витоків продуктивності,
- оптимізації планів виконання запитів,
- автошардінгу та балансування ресурсів.

Ці технології поступово зменшують потребу в ручному втручанні DBA (Database Administrator) і відкривають шлях до **автономних баз даних**.

2. Автономна база даних: визначення

Автономна БД (Autonomous Database) — це база даних, яка:

- **Налаштовується автоматично** (self-configuring),
- **Оптимізується в реальному часі** (self-optimizing),
- **Відновлюється після збоїв без втручання людини** (self-healing),
- **Автоматично масштабується** (self-scaling),
- **Захищає себе** (self-securing, зокрема в хмарних середовищах).

Приклад: **Oracle Autonomous Database**, **AWS Aurora**, **Azure SQL Database with AI optimization**, **Google Cloud Spanner**.

3. Основні можливості AI у СУБД

Напрямок	Приклади застосування
Self-tuning	Автоматичне налаштування кешу, пулу з'єднань
Query Optimization	Вибір кращого плану виконання запиту на основі ML
Indexing	Рекомендації щодо створення/видалення індексів
Resource Management	Автоматичне масштабування CPU, RAM, IO
Security	Виявлення підозрілих запитів та захист від SQL-ін'єкцій
Incident Response	Самовідновлення після збою або DDoS-атаки

4. Приклади застосування

Oracle Autonomous Database (OADB)

- Навчається на патернах запитів користувачів
- Оптимізує плани запитів без участі DBA
- Виконує автоматичне резервне копіювання та патчинг

Google BigQuery AI

- Автоматичне збереження та візуалізація запитів
- Побудова ML-моделей прямо в SQL-запитах

Azure SQL Database

- Інтелектуальні поради щодо індексів
- Аналіз використання запитів

5. Поточні виклики

- **Довіра до автоматичних рішень**: DBA не завжди готові довірити контроль системі
- **Прозорість рішень**: ML-алгоритми часто — «чорна скринька»
- **Налаштування під контекст**: універсальні оптимізації не завжди ефективні для складних систем
- **Безпека самонавчальних систем**: атаки через отруєння даних (data poisoning)

Словник термінів

Термін	Пояснення
DBA	Database Administrator — адміністратор баз даних
Self-tuning	Автоматичне налаштування параметрів без втручання людини
Query Plan	Послідовність дій, яку СУБД виконує для реалізації запиту
Index Recommendation	Система, що пропонує створення або видалення індексу
Data Poisoning	Впровадження шкідливих даних для спотворення навчання ML-моделі

Практичне завдання

1. Дослідіть можливості Oracle Autonomous Database на сайті Oracle.
2. Порівняйте AI-функції у трьох провідних хмарних платформах: AWS, Azure, Google Cloud.
3. Змоделюйте сценарій, у якому автономна СУБД:
    - автоматично додає індекс,
    - масштабує ресурси при піковому навантаженні,
    - створює бекап після інциденту.
4. Проаналізуйте ризики застосування AI в управлінні БД у медичних або банківських системах.

Sketch-based indexing, сучасні доступи та оптимізація запитів

Мета теми

Ознайомити студентів із сучасними техніками прискорення запитів до великих обсягів даних через використання **скетчів**, **пробних індексів**, **оптимізованих структур доступу** та **адаптивних стратегій планування запитів**. Пояснити застосування й обмеження цих підходів у реальних аналітичних базах.

1. Що таке Sketch?

Sketch — це компактне представлення даних, що дозволяє:

- оцінювати частоти, кількість унікальних елементів,
- працювати з великими потоками даних у пам’яті,
- підтримувати аппроксимації з допустимою похибкою.

Це математично гарантовані наближення результатів, застосовувані для агрегацій та фільтрації на етапі планування запитів.

Приклади скетчів:

- **HyperLogLog** — оцінка кількості унікальних значень,
- **Count-Min Sketch** — частотність елементів,
- **Bloom Filter** — перевірка належності до множини,
- **Top-k Sketch** — найчастіші значення.

2. Sketch-based indexing: як це працює?

Sketch-індекс — це *не класичний індекс*, а попередньо згенерований легкий об’єкт, який:

- зберігає статистику про набір даних,
- дозволяє фільтрувати дані на ранніх етапах виконання запиту,
- використовується для оптимізації плану запиту ще до доступу до повного обсягу таблиці.

Це особливо корисно для:

- аналітичних запитів,
- розподілених СУБД,
- роботи з потоками (streaming).

3. Сучасні структури доступу

Структура	Особливості
Zone Maps	Зберігають min/max для блоків даних. Застосовується в колонкових БД

Структура	Особливості
Bloom Index	Перевірка входження без повного сканування
Adaptive Indexing (Cracking)	Побудова індексу під час виконання запитів
Learned Index Structures	Використовують ML-моделі для передбачення позицій елементів у масиві
Column-Store	Підтримка швидкого агрегаційного доступу до окремих стовпців

4. Оптимізація запитів

Скетчі використовуються для:

- **Selectivity estimation**: оцінка кількості записів, що задовольнять умову.
- **Early filtering**: швидке відкидання непотрібних блоків.
- **Rewriting queries**: адаптація запитів на основі статистики.
- **Materialized Views + Sketch**: передагреговані перегляди зі скетчами замість повних даних.

Сучасні БД (наприклад, **Apache Druid**, **ClickHouse**, **BigQuery**) активно використовують ці техніки для OLAP.

Словник термінів

Термін	Пояснення
Sketch	Структура даних для швидкої, приблизної оцінки
Zone Map	Мінімальні/максимальні значення у блоці даних
Bloom Filter	Структура для перевірки належності елементів
Learned Index	Індекс на основі ML для передбачення позицій
Adaptive Query Processing	Автоматична перебудова плану запиту
Selectivity	Частка записів, які відповідають умові
Top-k	Найпопулярніші k елементів за певною метрикою

Практичне завдання

1. Розгляньте приклад Count-Min Sketch: реалізуйте функцію в Python, що обчислює частотність символів у потоці з 1% похибкою.
2. Порівняйте Bloom Filter і звичайний список для перевірки наявності значення — змоделюйте ситуацію з 10 млн значень.
3. Спробуйте підключити Google BigQuery або Apache Druid і дослідити, як там працюють попередньо побудовані агрегати.
4. Побудуйте SQL-запит, який міг би скористатися zone maps (на прикладі ClickHouse або аналогічної БД).
5. Опишіть сценарій, у якому Adaptive Indexing вигідний, але класичний індекс — ні (наприклад, у випадку часто змінюваних фільтрів).

Контроль схеми — Schema-Agnostic Databases

Мета теми
Пояснити, що таке **schema-agnostic** (незалежні від схеми) бази даних, які проблеми вони вирішують у гнучких і динамічних системах, як реалізується контроль даних без чітко визначеної схеми, та які ризики і переваги несе такий підхід.

1. Що таке Schema-Agnostic Databases?
Schema-Agnostic Database — це система керування базами даних, яка не вимагає жорстко визначеної структури (схеми) даних під час запису.
Це означає:

- Кожен запис (документ) може мати **власний набір полів**.
- Схема може **еволюціонувати динамічно**.
- **Жодної потреби** у ALTER TABLE або попередньому проєктуванні структури.

Приклад:
У MongoDB два документи в одній колекції можуть виглядати так:

```
{ "_id": 1, "name": "Olena", "email": "olena@example.com" }
{ "_id": 2, "name": "Taras", "phone": "+380501112233" }
```

2. Навіщо це потрібно?

Переваги	Пояснення
Гнучкість	Дані змінюються разом із потребами застосунку
Швидка розробка	Не потрібно змінювати схему під час нових релізів
Підтримка різнорідних джерел	Зручно при імпорті JSON, логів, API-відповідей

Особливо актуально для:

- NoSQL баз (MongoDB, Couchbase, Firebase),
- даних з IoT, логів, REST API,
- систем з частими змінами бізнес-логіки.

3. Як реалізується контроль структури?
Хоча схема не фіксується жорстко, контроль усе ж потрібен. Основні стратегії:

3.1. Schema Validation (на етапі запису)
MongoDB дозволяє налаштувати **валидацію схем** за допомогою правил JSON Schema:

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["name", "email"],
    "properties": {
      "email": {
        "bsonType": "string",
        "pattern": "^.+@.+$"
      }
    }
  }
}
```

3.2. Schema Discovery (на етапі читання)
Інструменти типу **MongoDB Compass**, **NoSQLBooster**, **Dataform** або власний код дозволяють аналізувати схему за збереженими даними.

3.3. External Schema Definition
У мікросервісах часто застосовують **контракти** — схема визначається поза базою (наприклад, у OpenAPI, GraphQL або Avro) і перевіряється програмно.

4. Ризики та виклики

Проблема	Потенційне рішення
Втрата цілісності	JSON Schema Validation, програмна перевірка
Неможливість гарантувати тип	Динамічне типізування, тестування
Складнощі з аналітичними запитами	Нормалізація при ETL
Зниження читабельності	Документація + автогенерація схем

Словник термінів

Термін	Пояснення
Schema-Agnostic	Такий, що не вимагає жорсткої схеми
JSON Schema	Формат опису структури документів JSON
Validation	Перевірка відповідності структури/типів
Schema Discovery	Автоматичне виявлення структури за даними
Flexible Schema	Можливість зберігати записи з різною структурою

Практичне завдання

1. У MongoDB створіть колекцію зі schema validation (name — обов'язковий, email — у правильному форматі).
2. Додайте документ, що **не відповідає** схемі, та проаналізуйте помилку.
3. Згенеруйте звіт про структуру даних у колекції (використайте MongoDB Compass).
4. Опишіть сценарій, у якому schema-agnostic підхід економить час розробника, але створює ризики для аналітики.

Графові бази даних — прогрес, виклики й тенденції

Мета теми
Ознайомити студентів з сучасним станом, практичним значенням і викликами використання графових баз даних (Graph Databases) у складних інформаційних системах. Проаналізувати тенденції розвитку та типові сфери застосування.

1. Що таке графові бази даних?
Графові БД — це тип NoSQL СУБД, де дані зберігаються у вигляді **вузлів (nodes)** та **зв'язків (edges)** між ними. Кожен вузол або зв'язок може мати властивості (атрибути). На відміну від реляційних БД, де зв'язки задаються через JOIN, у графових БД вони є частиною структури.
Переваги:

- Висока ефективність при роботі з пов’язаними даними.
- Миттєвий перехід по зв’язках без складних об'єднань.
- Гнучкість у моделюванні (схема може змінюватися динамічно).

2. Сфери застосування

Сфера	Приклад
Соціальні мережі	Друзі, підписки, взаємодії
Рекомендаційні системи	Продукти, які переглядали схожі користувачі
Кібербезпека	Виявлення аномалій, зв’язків між IP, логінами
Управління IT-інфраструктурою	Мережеві топології, залежності між сервісами
Фінансовий моніторинг	Відстеження трансакцій, виявлення шахрайства
Біоінформатика	Взаємодії генів, білків, молекул

3. Прогрес останніх років

- **Cypher як стандарт**: Мова запитів у Neo4j стала де-факто стандартом (підтримується також у Memgraph, SAP Graph).
- **GQL (Graph Query Language)**: ISO-стандартизація графової мови запитів (2024).
- **Hybrid models**: Злиття реляційного та графового підходів (наприклад, ArangoDB, Microsoft SQL Graph).
- **Visualization engines**: Прогрес в інтерактивній візуалізації (GraphXR, Neo4j Bloom).
- **Graph embeddings + ML**: Побудова векторних подань графів для класифікації, рекомендацій, виявлення аномалій.

4. Технічні виклики

Виклик	Деталі
Масштабованість	Зростання числа зв’язків у великих графах створює труднощі з партиціюванням
Відсутність єдиного стандарту	Cypher, Gremlin, GSQL, SPARQL — різні підходи
Складність оптимізації запитів	Труднощі з оцінкою selectivity в графових переходах
Слабка підтримка транзакційності	Не всі графові БД забезпечують ACID

5. Тенденції розвитку

- **Graph + AI**: злиття з GNN (Graph Neural Networks), використання графів як основи для генерації фіч у ML.
- **GraphQL ≠ Graph DB**: зростання популярності GraphQL, але це лише API-підхід — не варто плутати.
- **Graph as infrastructure**: концепція "Everything is a graph" (Google Knowledge Graph, Facebook Social Graph).
- **GraphDB у хмарах**: Amazon Neptune, Azure CosmosDB (graph mode), Neo4j Aura Cloud — хмарні рішення з autoscaling.

Словник термінів

Термін	Пояснення
Node	Об'єкт (персона, подія, предмет) у графі
Edge	Зв’язок між об’єктами
Cypher	Декларативна мова запитів для графів
Graph Embedding	Представлення вузлів у вигляді векторів
Graph Neural Network (GNN)	Мережа, що працює з графовими структурами
Traversal	Прохід по графу
SPARQL	Мова запитів до RDF-графів (W3C стандарт)

Практичне завдання

1. У Neo4j створіть графову модель: студенти, викладачі, курси, оцінки. Опишіть запити: хто викладає що, хто з ким спільно навчається, хто має найвищий середній бал.
2. Дослідіть приклад open-source проєкту з GitHub, що використовує графову БД (наприклад, Metaphactory або Linkurious).
3. Побудуйте векторне подання вузлів на основі структури графа.
4. Порівняйте продуктивність JOIN-запитів у SQL і графових traversals на прикладі пошуку друзів друзів (friend-of-a-friend).

Data Mesh — децентралізована аналітична архітектура

Мета теми

Ознайомити студентів із сучасною парадигмою **Data Mesh** — підходом до управління даними в масштабі підприємства, який пропонує **децентралізовану** модель володіння та обробки даних. Пояснити ключові принципи, архітектурні рішення та виклики реалізації.

1. Що таке Data Mesh?

Data Mesh — це архітектурний і організаційний підхід, який:

- відмовляється від централізованих Data Lake чи DWH,
- передає відповідальність за дані безпосередньо доменним командам,
- розглядає дані як **продукт**, а не лише сировину для звітів,
- забезпечує масштабованість і адаптивність у великих організаціях.

Автор концепції: Zhamak Dehghani (ThoughtWorks), 2019 рік.

2. Чотири принципи Data Mesh

Принцип	Пояснення
Децентралізоване доменне володіння	Команди, які створюють дані, самі ними і управляють
Дані як продукт	Кожен набір даних має чітке призначення, якість і API
Платформа самообслуговування	Інфраструктура, що дозволяє командам легко публікувати й споживати дані
Федеративне управління	Загальні стандарти, без централізації контролю

3. Як це працює?

- **Кожна команда-домен** (наприклад, маркетинг, продажі, логістика) публікує свої **Data Products** — добре описані набори даних, які можна використовувати іншими командами.
- Дані **мають API**, документацію, політику безпеки, опис структури.
- Платформа забезпечує:
  - моніторинг якості,
  - безпечне підключення,
  - підтримку каталогів (Data Catalog),
  - автоматичну валідацію.

4. Чим відрізняється від Data Lake / DWH?

Характеристика	Data Lake / DWH	Data Mesh
Власність	Централізована	Децентралізована
Архітектура	Монолітна	Мережна
Інтеграція	ETL/ELT	Data products + API
Зміна структури	Через централізовані зміни	У межах домену
Масштабованість	Обмежена	Висока

5. Виклики реалізації

- **Зміна культури**: необхідно перевчити аналітиків і розробників думати "продуктово".
- **Складність управління**: збільшується кількість джерел даних і відповідальностей.
- **Якість та узгодженість**: різні команди можуть мати різні підходи до якості та структури.
- **Інфраструктурні витрати**: потрібна потужна платформа для DataOps.

6. Технології для Data Mesh

- **Data Catalog**: Amundsen, DataHub, Collibra
- **Data Platform**: Snowflake, Databricks, GCP BigQuery, AWS Redshift
- **Streaming**: Kafka, Flink
- **API Management**: GraphQL, REST, OpenAPI
- **Data Contracts**: JSON Schema, Avro, Protobuf

Словник термінів

Термін	Пояснення
Data Product	Якісний набір даних з API та документацією
DWH (Data Warehouse)	Централізоване сховище даних
Data Lake	Необроблене сховище великих обсягів даних
Data Contract	Угода про структуру й типи даних
Federated Governance	Розподілене управління якістю та безпекою
Schema Evolution	Зміна структури даних без порушення сумісності

Практичне завдання

1. Опишіть 3 Data Products, які могла б публікувати команда університету (наприклад, “Студенти”, “Викладачі”, “Успішність”).

2. Визначте API-інтерфейс для одного з продуктів (поле, тип, призначення, джерело).
3. Складіть JSON Schema для валідації такого продукту.
4. Побудуйте порівняльну таблицю “Data Mesh vs Data Lake” за 5 критеріями.
5. Запропонуйте модель реалізації Data Mesh у системі аналітики для великої IT-компанії або університету.

Concurrency Control та Fault Tolerance у розподілених БД

Мета теми
Розкрити механізми забезпечення **одночасного доступу (Concurrency Control)** та **відмовостійкості (Fault Tolerance)** у розподілених базах даних, продемонструвати сучасні алгоритми, їх обмеження та вплив на продуктивність і консистентність систем.

1. Що таке Concurrency Control?
Concurrency Control (контроль паралелізму) — це **набір механізмів**, що забезпечують **узгодженість і цілісність даних** при паралельному виконанні транзакцій.
У розподілених БД — це особливо складне завдання, бо:

- Транзакції можуть запускатись у різних вузлах.
- Мережа — ненадійна (затримки, розриви).
- Дані можуть реплікуватися та оновлюватися одночасно в кількох місцях.

2. Підходи до керування паралелізмом
2.1. Two-Phase Locking (2PL)

- Гарантує **серіалізовність** (strict serializability)
- Блокує ресурси (read/write locks), але може спричинити **взаємоблокування (deadlocks)**

2.2. Timestamp Ordering (TO)

- Усі транзакції отримують мітки часу.
- Операції виконуються відповідно до часової шкали.
- Може призвести до відхилення (abort) транзакцій із запізненням.

2.3. Optimistic Concurrency Control (OCC)

- Застосовується в системах з **невеликим конфліктом**.
- Перевірка узгодженості — лише при коміті.
- Зручно для Web API, мобільних застосунків, CRDT.

2.4. Multiversion Concurrency Control (MVCC)

- Користувачі читають *знімок* даних (snapshot), не блокуючи інші транзакції.
- Популярний у PostgreSQL, Spanner, Aurora.

3. Відмовостійкість (Fault Tolerance)
Fault Tolerance — це здатність системи **продовжувати працювати**, навіть коли окремі вузли виходять з ладу або зникає з’єднання.

Основні механізми:

Механізм	Пояснення
Реплікація	Копіювання даних між вузлами (Primary-Secondary, Multi-leader)

Механізм	Пояснення
Quorum-based protocols	Читання/запис тільки при досягненні кворуму (наприклад, majority vote)
Consensus (Paxos, Raft)	Алгоритми досягнення згоди у кластері
Reconciliation	Злиття конфліктних змін після відновлення зв’язку
Failure Detector	Механізм, який виявляє непрацюючі вузли

Приклад: Paxos vs Raft

Критерій	Paxos	Raft
Зрозумілість	Складний	Простий і навчальний
Впровадження	Використовується в Chubby, Cassandra	Використовується в etcd, Consul
Основна ідея	Пропозиція, обіцянка, акцепт	Вибір лідера, лог змін, реплікація

4. Взаємозв’язок із CAP-теоремою

CAP-теорема стверджує, що **неможливо забезпечити одночасно**:

- C — узгодженість (Consistency)
- A — доступність (Availability)
- P — стійкість до розділення мережі (Partition tolerance)

У розподілених БД:

- **CP-системи** (Spanner) жертвують доступністю.
- **AP-системи** (Cassandra, DynamoDB) жертвують консистентністю.

Словник термінів

Термін	Пояснення
MVCC	Зберігання кількох версій даних
Deadlock	Стан, коли транзакції блокують одна одну
Quorum	Мінімальна кількість реплік для підтвердження
Log Replication	Реплікація змін через лог (як у Raft)
Consensus	Досягнення згоди про порядок операцій
Leader Election	Вибір головного вузла для синхронізації

Практичне завдання

1. Наведіть приклад ситуації, коли виникає deadlock при 2PL.
2. Порівняйте поведінку MVCC та OCC на прикладі двох конкурентних транзакцій.
3. Реалізуйте простий приклад timestamp ordering на SQL-рівні (за допомогою TIMESTAMP поля).
4. Поясніть, як можна виявити і "зцілити" розбіжності між репліками у випадку падіння одного вузла.
5. Побудуйте таблицю порівняння систем PostgreSQL, Cassandra та Spanner за параметрами: тип узгодженості, fault tolerance, concurrency control.

Streaming Databases та обробка подій у реальному часі

Мета:
Надати студентам глибоке розуміння концепцій потокової обробки даних, відмінностей між потоковою та пакетною обробкою, а також ознайомити з сучасними платформами та сценаріями використання streaming databases.

1. Що таке потокова обробка даних?
Streaming data — це безперервний потік подій або записів, які генеруються системами в реальному часі (датчики, логи, транзакції, повідомлення).
На відміну від пакетної обробки (batch), яка працює з фіксованими наборами даних, потокова обробка дозволяє реагувати **миттєво** на події, що відбуваються.

2. Основні концепції
- **Stream** — нескінченна послідовність даних
- **Event time vs Processing time** — коли подія відбулася vs коли її оброблено
- **Windowing** — обробка за часовими інтервалами (tumbling, sliding, session)
- **Stateful stream processing** — збереження інформації про стан між подіями

3. Архітектура Streaming System
- **Producers** (наприклад, Kafka producers)
- **Message brokers** (Kafka, Pulsar, Redpanda)
- **Stream processors** (Flink, Spark Structured Streaming, Materialize)
- **Sinks** — системи призначення (Data lake, OLAP, dashboards)

4. Streaming Databases
На відміну від класичних баз:
- Дозволяють виконання **безперервних запитів** (continuous queries)
- Дані автоматично агрегуються або фільтруються в реальному часі
- Часто підтримують **SQL-подібні** інтерфейси для обробки стрімів

Приклади:
- **Materialize** — PostgreSQL-стиль запитів до Kafka-стрімів
- **RisingWave** — cloud-native stream warehouse
- **ClickHouse** з інтеграцією Kafka (через Materialized Views)

5. Сценарії застосування
- **Фінансовий моніторинг**: обробка транзакцій з виявленням шахрайства
- **Інтернет речей (IoT)**: агрегація та аналіз показників датчиків у реальному часі
- **Рекомендаційні системи**: врахування останніх дій користувача
- **Моніторинг логів/DevOps**: alerting на базі стрімів логів (Prometheus + Loki)

6. Словник термінів

Термін	Пояснення
Event Time	Час події згідно з її джерелом
Processing Time	Час, коли подія потрапила в систему
Windowing	Групування подій за часовими рамками
Watermark	Оцінка прогресу у часі в stream processing

State Store	Місце збереження проміжних результатів (Flink RocksDB)
Exactly-once	Семантика доставки, яка гарантує один раз і тільки один

7. Практичні завдання

1. Поясніть, чому обробка в режимі реального часу може бути критичною у фінансових транзакціях.
2. Побудуйте SQL-запит до Kafka-стріму (використовуючи синтаксис Materialize або Flink SQL), що підраховує кількість подій за кожні 5 хвилин.
3. Порівняйте windowing-стратегії: tumbling, sliding, session — на прикладах.
4. Розробіть простий pipeline з Kafka → Flink → PostgreSQL.
5. Опишіть, як забезпечити exactly-once delivery у Flink при записі в базу даних.

Zero-ETL архітектури: пряме з'єднання аналітики з джерелом

Мета:
Ознайомити студентів з концепцією Zero-ETL, що дозволяє виконувати аналітику без традиційних процесів Extract-Transform-Load (ETL), шляхом безпосереднього підключення аналітичних систем до джерел операційних даних.

1. Що таке Zero-ETL?
Zero-ETL — це архітектурний підхід, за якого усувається необхідність у традиційних ETL-конвеєрах, а дані з OLTP-систем (операційних) стають доступними для OLAP-запитів (аналітичних) практично миттєво.

2. Основні концепції

- **CDC (Change Data Capture)** — виявлення змін у джерелі (INSERT/UPDATE/DELETE) і передача в реальному часі
- **Federated queries** — виконання запитів до різних джерел, як до єдиної логічної бази
- **Data lakehouse** — поєднання гнучкості сховища (lake) з властивостями класичної БД (ACID, schema)
- **Virtualization layer** — рівень абстракції для доступу до даних без фізичного копіювання

3. Платформи та сервіси

- **Amazon Aurora Zero-ETL → Redshift**
- **BigQuery Data Federation** (до CloudSQL, Sheets, S3)
- **Databricks Lakehouse**
- **Presto / Trino** — рушії для віртуалізованого доступу

4. Переваги

- Значне зменшення затримки між транзакційною та аналітичною обробкою
- Відсутність копій даних → зменшення витрат
- Менша складність обслуговування пайплайнів

5. Недоліки

- Обмеження по latency та продуктивності при великих запитах

- Обмеження у трансформаціях
- Проблеми з консистентністю даних у реальному часі

6. Словник термінів

Термін	Пояснення
Zero-ETL	Архітектура без ETL-пайплайнів
CDC	Change Data Capture – механізм фіксації змін у БД
Federated query	Запит, що виконується до декількох джерел одночасно
Virtualization	Абстракція доступу до даних без фізичної реплікації
Lakehouse	Гібрид lake + warehouse

7. Практичні завдання

1. Опишіть кейс, коли Zero-ETL може зменшити час підготовки звітності з годин до секунд.
2. Порівняйте: традиційний ETL vs Zero-ETL — по 4 параметрах (час, складність, затрати, гнучкість).
3. Опишіть, як буде реалізовано доступ до PostgreSQL через Presto без копіювання таблиць.
4. Складіть SQL-запит у стилі federated query, що об'єднує дані з таблиці PostgreSQL і CSV на S3.
5. Дослідіть Aurora + Redshift Zero-ETL та поясніть як реалізовано оновлення в режимі "майже в реальному часі".

Data Contracts та контроль змін у схемі (Schema Evolution)

Мета:
Пояснити студентам, як **контракти даних (Data Contracts)** і процес **еволюції схем (Schema Evolution)** допомагають підтримувати узгодженість, надійність і стабільність систем обробки даних у середовищах з великою кількістю залежностей.

1. Що таке Data Contract?
Data Contract — це формалізована угода між **продюсером даних** (хто надає дані) і **консьюмером** (хто їх використовує), що описує структуру, типи та поведінку даних.
Це не лише документація, а й **технічний механізм контролю змін у схемі**.

2. Проблеми без контрактів

- Незаплановані зміни в полях (наприклад, зміна типу) ламають аналітичні пайплайни
- Дані в Data Lake не мають чіткої структури → важко відлагодити помилки
- CI/CD-тести не перевіряють сумісність схем

3. Компоненти контракту

- **Типи полів** (наприклад, int, string)
- **Статус поля** (обов’язкове, необов’язкове, deprecated)
- **Опис призначення**
- **Семантика змін** (чи можна змінювати назву, тип, значення)

4. Схемна сумісність

Тип зміни	Backward-compatible	Forward-compatible	Примітка
Додано поле (optional)	✅	✅	Безпечно для обох сторін
Видалено поле	❌	✅	Consumer може залежати від нього
Змінено тип	❌	❌	Ламає все

5. Інструменти та формати

- **Avro**, **Protobuf**, **JSON Schema** — формати опису схем
- **OpenAPI / AsyncAPI** — для API-контрактів
- **Pact**, **ContractTest**, **Schema Registry (Confluent)**
- **CI-плагіни** для перевірки сумісності при Pull Request

6. Словник термінів

Термін	Пояснення
Data Contract	Формальна угода щодо структури даних
Schema Evolution	Керовані зміни у структурі даних
Backward Compatibility	Нові дані працюють зі старими програмами
Schema Registry	Сховище описів схем для перевірок і валідації
Contract Test	Тест, що перевіряє відповідність контракту

7. Практичні завдання

1. Наведіть приклад, як зміна типу int на string в полі "age" може зламати пайплайн.
2. Розпишіть, які зміни можна вносити у схему без порушення контракту (назвіть щонайменше 3).
3. Створіть JSON Schema для події "нове замовлення", що містить ідентифікатор, дату, список товарів.
4. Поясніть, як CI/CD може інтегрувати перевірку сумісності схем.
5. Дослідіть можливості Apache Avro та порівняйте з JSON Schema за гнучкістю та валідністю.

Vector Databases та векторні пошуки для AI-застосунків

Мета:
Познайомити студентів з основами зберігання та пошуку векторних представлень даних, що використовуються в штучному інтелекті, особливо в NLP, комп'ютерному зорі та рекомендаційних системах.

1. Що таке векторна база даних?
Векторна БД зберігає дані у вигляді векторів (найчастіше — списків чисел, що представляють сенс об'єкта), та дозволяє ефективно виконувати **пошук найближчих сусідів** (Nearest Neighbor Search).

2. Векторні подання (embeddings)

- **Текст:** Sentence Transformers, Word2Vec, BERT embeddings

- **Зображення:** ResNet, CLIP embeddings
- **Аудіо, відео, графи** — через специфічні енкодери

3. Методи пошуку

- **Brute-force** — точний пошук, але повільний
- **Approximate Nearest Neighbor (ANN)**:
  - HNSW (Hierarchical Navigable Small World Graph)
  - IVF (Inverted File Index)
  - PQ (Product Quantization)

4. Популярні інструменти

Назва	Особливості
FAISS	Open-source, швидкий пошук, підтримка GPU
Milvus	Платформа з UI, кластеризація, масштабування
Qdrant	API-first підхід, фільтрація + ANN
Weaviate	Інтеграція з OpenAI, класи, GraphQL

5. Застосування

- **RAG-системи (Retrieval-Augmented Generation)**
- **Пошук по документах та PDF** (Semantic search)
- **Рекомендаційні системи** (схожі продукти, користувачі)
- **Image similarity search**, **visual tagging**
- **Індексація відеофрагментів, аудіо-семантика**

6. Словник термінів

Термін	Пояснення
Embedding	Векторне представлення об'єкта (тексту, зображення)
ANN	Приблизний пошук найближчих векторів
Index	Структура для пришвидшення пошуку векторів
Similarity metric	Міра близькості: cosine, Euclidean, dot product
RAG	Архітектура генерації відповідей з пошуком контексту

7. Практичні завдання

1. Побудуйте векторні представлення для набору речень з використанням Sentence Transformers.
2. Встановіть FAISS та реалізуйте пошук за cosine similarity по 1000 embedding.
3. Опишіть, як векторні бази використовуються в RAG-архітектурах для LLM.
4. Побудуйте індекс у Qdrant для пошуку схожих продуктів за назвою та описом.
5. Порівняйте cosine similarity та Euclidean distance на прикладі: яка метрика краще для семантичного текстового пошуку?

Список використаних джерел

1. ХНЕУ ім. С. Кузнеця. Бази даних: лабораторний практикум [Електронний ресурс] / уклад. В. В. Федько, В. П. Бурдаєв. — Харків: ХНЕУ ім. С. Кузнеця, 2019. — 215 с. repository.hneu.edu.ua
2. Каштан, В. Ю., Іванов, Д. В. Конспект лекцій: Бази даних в інформаційних системах. Частина 1. — Дніпро: НТУ «Дніпровська політехніка», 2020. — 58 с. it.nmu.org.ua
3. Нікітчук, О. І. Реляційні та нереляційні бази даних: навч. посібник. — Тернопіль: ТНТУ, 2023. — 208 с.
4. Elmasri, R., & Navathe, S. B. Fundamentals of Database Systems. — 7th ed. — Boston: Pearson, 2020. — 1272 p.
5. Coronel, C., & Morris, S. Database Systems: Design, Implementation, & Management. — 14th ed. — Boston: Cengage Learning, 2020. — 784 p.
6. MongoDB Inc. MongoDB Manual: The Official MongoDB Documentation — Version 5.0. — New York: MongoDB Press, 2021. — [Електронний ресурс]. — Режим доступу: https://www.mongodb.com/docs/manual/
7. PostgreSQL Global Development Group. PostgreSQL Documentation — Version 13. — 2021. — [Електронний ресурс]. — Режим доступу: https://www.postgresql.org/docs/13/ MongoDBw3schoolsua.github.io
8. Date, C. J. Database Design and Relational Theory: Normal Forms and All That Jazz. — 2nd ed. — O’Reilly Media, 2020. — 384 p.
9. Stefanie Scherzinger, Sebastian Sidortschuck. An Empirical Study on the Design and Evolution of NoSQL Database Schemas. — 2020. — [Електронний ресурс]. — arXiv. arXiv+2ela.kpi.ua+2
10. Yaser Mansouri, M. Ali Babar. The Impact of Distance on Performance and Scalability of Distributed Database Systems in Hybrid Clouds. — 2020. — [Електронний ресурс]. — arXiv. arXiv