Міністерство освіти та науки України
Карпатський національний університет імені Василя Стефаника

М.В. Семанькiв

МЕТОДИЧНІ ВКАЗІВКИ ДО ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ “БАЗИ ДАНИХ”

Для студентів всіх форм навчання
галузі знань F Інформаційні технології

Івано-Франківськ
2025

УДК 004.421
ББК 32.973-018.1

Рекомендовано до друку Вченою радою факультету математики та інформатики Карпатського національного університету імені Василя Стефаника (протокол № 9 від 22.10.2025р.).

Рецензенти :

Ровінський В.А., кандидат технічних наук, доцент (Карпатський національний університет імені Василя Стефаника)

Ізмайлов А.В., кандидат технічних наук (Карпатський національний університет імені Василя Стефаника)

Семаньків М.В. Методичні вказівки до лабораторних робіт з дисципліни “Бази даних”. (2-га редакція із доповненнями) – Івано-Франківськ: Карпатський національний університет імені Василя Стефаника, 2025. – 86 с.

Методичні вказівки містять необхідні відомості до виконання лабораторних завдань, завдання для виконання; сприяють розумінню особливостей при роботі з реляційними базами даних, закріпленню навичок роботи з основними об’єктами СУБД: таблицями, запитами, формами, звітами.

Для студентів спеціальностей Комп’ютерні науки та Інформаційні системи та технології.

ЗМІСТ

ВСТУП

Мета вивчення баз даних, як сукупності засобів для зберігання структурованої інформації, полягає в узагальнені та систематизації цих уявлень, формуванні відповідних теоретичних знань, з'ясуванні загальних принципів опрацювання структурованої інформації та оволодіння навичками опрацювання баз даних за допомогою системи управління базами даних.

Методичні вказівки включають завдання для лабораторних робіт з теоретичною частиною та практичними завданнями. Мета циклу лабораторного практикуму полягає в тому, щоб студенти отримали навички у побудові та використанні баз даних, виконувати проектування інформаційного забезпечення (логічну та фізичну структури баз даних) інформаційних систем, враховуючи особливості та відмінності різних систем управління баз даних. Під час виконання даної лабораторної роботи, реалізуючи поетапно індивідуальні завдання, студент засвоює теоретичний та практичний матеріал, що необхідний для формування знань та вмінь, які застосовуються при роботі з базами даних реляційного типу. Практикум включає дві підготовки до контрольної роботи із збіркою завдань для визначеної бази даних.

Після виконаного студентом лабораторного практикуму студент обирає тему індивідуального завдання та здійснює проектування бази даних згідно вказаних вимог.

ЛАБОРАТОРНА РОБОТА №1.
SELECT: НАЙПРОСТІША ВИБІРКА З ТАБЛИЦІ

1. Теоретичні відомості

У кожного об'єкту в БД є унікальне ім'я. Відповідно до стандарту ANSI/ISO, в SQL імена повинні містити від 1 до 18 символів, починатися з літери і не містити пропуски або спеціальні символи пунктуації. У стандарті SQL2 максимальне число символів в імені збільшене до 128, при цьому на практиці в різних СУБД дозволене використання в іменах таблиць спеціальних символів. Тому для підвищення переносності краще робити імена порівняно короткими і уникати використання в них спеціальних символів.

Стандарт ISO SQL не підтримує такі формальні терміни, як відношення, атрибут і кортеж. Замість них використовують терміни таблиця, стовпець, рядок.

Оператор SQL складається з зарезервованих слів, а також зі слів, що визначаються користувачем. Зарезервовані слова слід записувати саме так, як зазначено в стандарті, і не можна розбивати на частини для переносу з одного рядка на інший. Слова, визначені користувачем, задаються самим користувачем (у відповідності з певними синтаксичними правилами) і являють собою імена різних об'єктів бази даних – таблиць, стовпців, представлень, індексів і т.д.

Хоча в стандарті це не зазначено, однак у багатьох діалектах мови SQL вимагають завдання в кінці оператора деякого символу, що позначає закінчення його тексту; як правило, з цією метою використовується крапка з комою (;).

Більшість компонентів операторів SQL не відчутно до регістру. Одним важливим виключенням з цього правила є *символьні літерали* – дані, які повинні вводитися точно так, як були введені відповідні їм значення, що зберігаються в базі даних. Наприклад, якщо в базі даних зберігається значення прізвища ‘ІВАНОВ’, а в умові пошуку вказано символьний літерал ‘Іванов’, то цей запис не буде знайдений. Всі нечислові значення даних завжди повинні бути взяті в одинарні лапки, а всі числові дані не повинні бути взяті в одинарні лапки. Нижче наведено приклад використання літералів для вставки даних в таблицю.

Оскільки мова SQL має вільний формат, окремі оператори SQL і їх послідовності будуть мати більш зручний для читання вигляд при використанні відступів і вирівнювання. Рекомендується дотримуватися наступних правил:

- кожна конструкція в операторі повинна починатися з нового рядка;
- початок кожної конструкції має бути позначено таким же відступом, що і початок інших конструкцій оператора;

- якщо конструкція складається з декількох частин, кожна з них повинна починатися з нового рядка з деяким відступом відносно початку конструкції, що буде вказувати на їх підпорядкованість.

Вибірку і відображенні даних однієї або декількох таблиць бази даних виконують за допомогою оператора SELECT, який є найбільш затребуваним оператором мови SQL. Загальний формат оператора SELECT має наступний вигляд:

SELECT [DISTINCT | ALL] <список даних>
  FROM <список таблиць>
  [WHERE <умова вибірки>]
  [GROUP BY <ім'я стовпця>[, <ім'я стовпця>]…]
  [HAVING <умова пошуку>]
  [ORDER BY <специфікація>[, <специфікація>]…];

Обробка елементів оператора SELECT виконується в наступній послідовності.

1. **FROM.** Визначаються імена використовуваної таблиці або декількох таблиць.
2. **WHERE.** Виконується фільтрація рядків об'єкта відповідно до заданими умовами.
3. **GROUP BY.** Утворюються групи рядків, що мають одне і те ж значення в зазначеному стовпці.
4. **HAVING.** Фільтруються групи рядків об'єкта відповідно до зазначеної умови.
5. **SELECT.** Встановлюється, які стовпці повинні бути присутніми у вихідних даних.
6. **ORDER BY.** Визначається впорядкованість результатів виконання оператора.

Порядок конструкцій в операторі SELECT не може бути змінений. Тільки дві конструкції оператора – SELECT і FROM – є обов'язковими, решта конструкцій можуть бути опущені.

Безумовна вибірка

Структура найпростішого запиту на вибірку даних

SELECT список результуючих стовпчиків
FROM список таблиць-джерел даних
WHERE умова виводу стрічок;

Ключове слово SELECT означає запит на подання інформації. Вона буде подана у вигляді результуючої таблиці стрічки якої задовольнятимуть умові. Стовпчики, на основі яких формуються результуючі або перевіряється умова, повинні належати таблицям перерахованим у списку.

Якщо список результуючих стовпчиків співпадає із списком єдиної таблиці-джерела, то такий список зручно представляти у скороченому вигляді за допомогою зірочки - *.

Порядок результуючих стовпчиків може бути довільним

Наприклад

```
SELECT name, surname
  FROM student;
```

Усунення дублювання в виводі - DISTINCT

Для виключення із результату SELECT- запиту повторів використовується ключове слово DISTINCT (відмінний). Наприклад сформувати список міст можна наступним запитом

Наприклад

```
SELECT DISTINCT city
  FROM student;
```

Логічні оператори AND, OR, NOT

В умові WHERE можна використовувати операції порівняння = > >= < <= <>

та логічні операції AND, OR, NOT.

Наприклад

```
SELECT  surname, name, city
  FROM student
  WHERE city= "Харків" AND birthday > '1990-12-25' ;
```

Оператор належності IN

Оператори IN (рівний довільному елементу із списку) і NOT IN (не рівний жодному елементу із списку) дозволяють ускладнювати умови відбору результуючих стрічок.

Наприклад, вибрати студентів окремих міст

Наприклад

```
SELECT *
  FROM exam_marks
  WHERE city IN ("Київ", "Львів");
```

або тих студентів, які не отримували ніколи деяких оцінок

```
SELECT *
  FROM exam_marks
  WHERE mark NOT IN (3, 5);
```

Оператор діапазону BETWEEN

Оператор BETWEEN дозволяє перевірити входження елемента в заданий інтервал із його межами включно

Наприклад:

```
SELECT *
  FROM subject
  WHERE city  BETWEEN "К" AND "М";
```

Оператор подібності LIKE

Пошук стрічкових значень по заданому зразку, який використовує символи: “_” – наявність, у вказаному місці одного довільного символу, •“%” - наявність, у вказаному місці довільної послідовності символів. Цей засіб часто використовують для вибору слів, що починаються на певну букву

Наприклад:

```
SELECT *
FROM student
WHERE surname LIKE ”П%”;
```

Також ми можемо відбирати слова, деякі символи в яких пишуться неоднозначно, наприклад

Наприклад:

```
SELECT *
  FROM subject
  WHERE subj_name  LIKE ”Комп_ютер%”;
```

Оператор впорядкування ORDER BY

Оператор ORDER BY використовується для впорядкування стрічок результуючої таблиці за значеннями деякого стовпчику або стовпчиків. Можна також вказати вид впорядкування за зростанням (ASC) або за спаданням (DESC). Зростаюче впорядкування застосовується за замовчуванням, наприклад

Наприклад:

```
SELECT *
  FROM student
  ORDER BY  city;
```

Зворотнє впорядкування забезпечується наступним чином:

```
SELECT *
  FROM student
  ORDER BY  city  DESC;
```

ВИБІР ОБМЕЖЕНОЇ КІЛЬКОСТІ РЯДКІВ

MySQL, SQLite, Postgresql	**MS SQL Server**
LIMIT	***TOP***
SELECT * FROM table_name **LIMIT** number;	SELECT **TOP** number|percent column_name(s) FROM table_name;

Оператор **OFFSET** дозволяє вказати з якого рядка задати вибірку значень SELECT * FROM Products ORDER BY ProductName LIMIT 3 OFFSET 2; Якщо необхідно вибрати всі рядки починаючи із заданого то оператор LIMIT можна пропустити: SELECT * FROM Products ORDER BY ProductName OFFSET 2;	

2. ПРАКТИЧНІ ЗАВДАННЯ

РОБОТА З ЗАПИТАМИ ПО ГОТОВІЙ ТАБЛИЦІ

Таблиця знаходиться в https://sql.js.org/examples/GUI/index.html

-- Query the data **SELECT * FROM employees;**	Вивести всі дані з таблиці

5 rows 5 columns

id	name	department	salary	hire_date
1	Alice Smith	Engineering	85000	2020-01-15
2	Bob Johnson	Marketing	72000	2019-03-20
3	Carol Williams	Engineering	92000	2018-11-07
4	Dave Brown	Finance	115000	2017-05-12
5	Eve Davis	Engineering	110000	2021-08-30

Фільтрація

select * from назва_таблиці where УМОВА

SELECT * FROM **Sumproduct** WHERE **Product** = *'Bikes';*	1. Показати всіх інженерів (умова по стовпцю department)
	2. Показати в кого salary більше 100 000
Фільтрація по діапазону **SELECT** * FROM **Sumproduct** WHERE **Amount** **BETWEEN** *1000***AND** *2000;*	3. Показати в кого salary від 80 000 до 110 000

Вручну в коді поміняти значення на порожнє, щоб перевірити запустити насутпний запит

```
-- Insert sample data
INSERT INTO employees (name, department, salary, hire_date) VALUES
  ('Alice Smith', 'Engineering', 85000, '2020-01-15'),
  ('Bob Johnson', 'Marketing', 72000, '2019-03-20'),
  ('Carol Williams', 'Engineering', 92000, '2018-11-07'),
  ('Dave Brown', NULL, 115000, '2017-05-12'),
  ('Eve Davis', 'Engineering', 110000, '2021-08-30');
```

Порожні записи (IS NULL).	4. Показати в кого нема department
Розширена фільтрація (AND, OR).	5. Показати Engineering та зарплата більша 100000
Оператор IN **SELECT * FROM Sumproduct** **WHERE ID IN (4, 12, 58, 67);**	6. Показати людей із вказаною зарплатою 72000, 92000
Оператор NOT **SELECT * FROM Sumproduct** **WHERE NOT City IN ('Toronto', 'Mont');**	7. Показати всі крім Engineering
	8. Показати всіх в кого department не порожній
Метасимвол знак процента (%) і зірочка(*) **SELECT * FROM Sumproduct** **WHERE Product LIKE** *'%Skis%';*	9. Показати в кого ім'я починається на букву С
Обчислювальне поле	
SELECT kilkist*10/124 from Sumproduct	10. Відобразити зарплату працівників в доларовому еквіваленті (/41)
Сортування даних	
Сортування даних **SELECT * FROM Sumproduct** **ORDER BY Amount;** **SELECT * FROM Sumproduct** **ORDER BY Amount DESC;**	11. Посортувати по зростанню стовпця salary 12. Посортувати по спаданню стовпця hire_date
Виведення обмеженої кількості рядків	
SELECT * FROM LIMIT 2	13. Показати тільки перші два рядки
Без дублікатів	
SELECT DISTINCT city **FROM student;**	14. Вивести всі department без повторень

Виконати завдання для зарахування:

Частина 1: https://sqlzoo.net/wiki/SELECT_from_WORLD_Tutorial до 7 включно

Частина 2:
https://sqlbolt.com/lesson/select_queries_with_constraints
https://sqlbolt.com/lesson/select_queries_with_constraints_pt_2
https://sqlbolt.com/lesson/filtering_sorting_query_results

ЛАБОРАТОРНА РОБОТА №2.
АГРЕГАТНІ ФУНКЦІЇ

1. Теоретичні відомості

В SQL допускаються наступні агрегатні функції:

• COUNT - робить підрахунок кількості рядків або не NULL значень полів, які вибрав запит;

• SUM - розраховує арифметичну суму всіх вибраних значень даного поля;

• AVG - проводить усереднювання всіх вибраних значень даного поля;

• MAX - знаходить і повертає найбільше зі всіх вибраних значень даного поля;

• MIN - знаходить і повертає якнайменше зі всіх вибраних значень даного поля.

Агрегатні функції використовуються подібно іменам полів в SELECT запиту, але з урахуванням того, що вони беруть імена поля як аргументи. Варто мати на увазі, що з SUM і AVG використовуються тільки числові поля, а з COUNT, MAX, і MIN можуть використовуватися числові або символьні поля. Коли функції записуються з символьними полями, MAX і MIN використовують їх в еквівалент ASCII, відповідно до якого вибирається максимальне або мінімальне значення.

Розрахунки підсумкових значень в операторі SELECT

Розрахунок підсумкових значень в операторі *SELECT* здійснюється за допомогою агрегатних або статистичних функцій: *COUNT*(), *SUM*(), *AVG*(), *MAX*(), *MIN*().

Функція *COUNT*(<вираз>) підраховує число входжень значень виразу в усі записи результуючого набору даних. Вона працює з данними будь-якого типу: числові, символьні або типу «дата/час».

ПРИКЛАД:

Визначить, вартість скількох книжок перевищує 25 грн.

```
SELECT COUNT(*) AS cCount
   FROM BookInventoryNumbers
   WHERE Cost > 25;
```

Якщо із групи однакових записів при підрахунку необхідно враховувати тільки одну, то перед виразом у дужках вставляють пропозицію *DISTINCT*. Найчастіше як вираз виступають імена стовпців. Вираз може розраховуватися і за значеннями декількох таблиць:

***COUNT (DISTINCT** <вираз>)*

ПРИКЛАД

Визначіть, скільки різних бібліотекарів видавали книги

***SELECT COUNT(DISTINCT** OutLibrarianCode) **AS** cCount*
***FROM** BookInventoryNumbers*

Загальну кількість записів, що відповідають зазначеній умові, можна визначити за допомогою функції *COUNT* без фрази *DISTINCT*. Однак один і той самий бібліотекар може видати кілька книжок різним читачам. Пропозиція *DISTINCT* дозволяє виключити з розрахунку значення, що дублюються.

Функція *SUM*(<вираз>) розраховує суму всіх значень стовпця або виразу. При цьому стовпець має складатися з числових даних (містити цілі числа, числа із плаваючою комою або грошові величини). Результат, що повертається цією функцією, має той же тип даних, що й стовпець або вираз, однак точність (розрядність) результату може бути вищою. Наприклад, якщо застосувати функцію *SUM*() до стовпця, що містить 16-розрядні цілі числа, то вона може повернути 32-розрядне ціле число.

ПРИКЛАД

Визначіть загальну вартість книжок бібліотечного фонду

***SELECT SUM**(Cost) **AS** cSum*
***FROM** BookInventoryNumbers*

Функції *MIN*(<вираз>), *MAX*(<вираз>) і *AVG*(<вираз>) дозволяють розрахувати відповідно найменше, найбільше й середнє значення в стовпці або виразі. Функції *MIN*(), *MAX*() працюють із числовими, рядковими й даними типу «дата/час». *AVG*() працює тільки з числовими даними. Результат, що повертається цими функціями, має такий самий тип даних, як і вираз.

ПРИКЛАД

Визначте мінімальну, максимальну й середню вартість книжок.

***SELECT MIN**(Cost) **AS** Mini, **MAX**(Cost) **AS** Maxi, **AVG** (Cost) **AS** Avgi*
***FROM** BookInventoryNumbers*

<u>Оператор SQL AS</u>

Оператор SQL AS використовується для задання назв результуючих стовпців у запиті.

SELECT DISTINCT Product, Amount+Quantity AS AvgPrice
FROM Sumproduct

Оператор створення таблиці має формат:

CREATE TABLE <ім’я таблиці>
(<ім’я стовпця><тип даних>[NOT NULL]
[,<ім’я стовпця><тип даних>[NOT NULL]]…);

Обов’язковими операндами оператора є ім’я створюваної таблиці та ім’я хоча б одного стопвця (поля) з вказанням типу даних, що зберігатимуться у цьому стовпці. При створенні таблиці для окремих полів можуть вказуватись деякі додаткові правила контролю введення в них даних, наприклад конструкція NOT NULL (не пусте) означає, що в цьому стовці повинно бути визначене значення.

Приклад створення таблиці. Нехай потрібно створити таблицю goods опису товарів, яка містить поля: type – вид товару, comp_id – ідентифікатор компанії-виробника, name – назва товару і price – ціна.

Оператор створення таблиці goods матиме вигляд:

```
CREATE TABLE goods
(type CHAR(8) NOT NULL,
comp_id CHAR(10) NOT NULL,
name VARCHAR(20),
price INTEGER);
```

Для додавання нових записів в таблицю використовується оператор INSERT INTO, який має такий формат:

INSERT INTO <ім’я таблиці> [(<список стовпців>)] VALUES (<список значень>)	або INSERT INTO <ім’я таблиці> [(<список стовпців>)] <вираз SELECT>

У першому форматі оператор INSERT призначений для введення нових записів з заданими значеннями у стовпцях. Порядок перерахування імен стовпців повинен відповідати порядку значень, перерахованих в списку операнда VALUES. Якщо <список стовпців> опущений, то у <списку значень> повинні бути перераховані всі значення в порядку стовпців структури таблиці.

У другому форматі оператор INSERT забезпечує можливість введення в задану таблицю нових рядків, які відібрані з іншої таблиці за допомогою виразу SELECT.

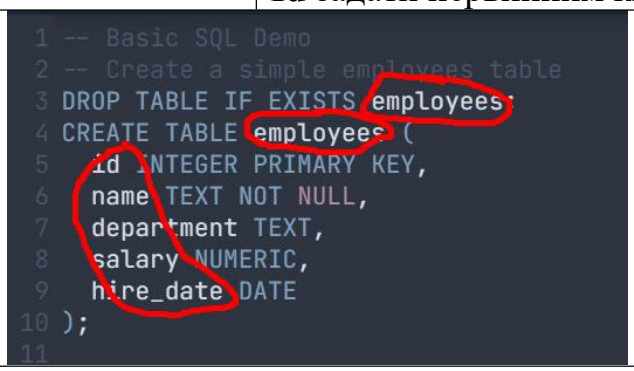
Наприклад: Введення записів.

Ввести в таблицю EMP запис про нового співробітника.

Оператор INSERT матиме такий вигляд:

```
INSERT INTO emp    VALUES (‘Ivanov’, 7500, ‘Lee’, ‘cosmetics’);
```

2. ПРАКТИЧНІ ЗАВДАННЯ

<table>
<tr><td>Запустіть онлайн компілятор</td><td>1. Перейти на сайт https://sql-js.github.io/sql.js/examples/GUI/index.html</td></tr>
<tr><td colspan="2">СТВОРЕННЯ ТАБЛИЦІ</td></tr>
<tr><td>CREATE TABLE <Ім'я таблиці>
(<ім'я стовпця> <тип>,
[<ім'я стовпця> <тип>, ...]);</td><td>1. На основі заданих у компіляторі команд переробити їх для створення таблиці
Products за наступними полями:
Id, ProductName , Manufacturer, ProductCount, Price, вказати для полів тип
Id задати первинним ключем</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td colspan="2">ДОДАВАННЯ ДАНИХ В ТАБЛИЦЮ</td></tr>
<tr><td>- Додавання одного повного рядка

INSERT INTO Sellers VALUES (6, '1st Street', 'Los Angeles', 'Harry Monroe', 'USA') – якщо значення стовпців вказані в тому порядку як вони йдуть при заданні таблиці</td><td>2. Заповнити:
(1, 'iPhone X', 'Apple', 2, 71000),
(2, 'iPhone 8', 'Apple', 3, 56000),
(3, 'Galaxy S9', 'Samsung', 6, 56000),
(4, 'Galaxy S8 Plus', 'Samsung', 2, 46000),
(5, 'Desire 12', 'HTC', 3, 26000);</td></tr>
<tr><td>- Додавання частини рядка
INSERT INTO Sellers (ID, City, Seller_name) VALUES (6, 'Los Angeles', 'Harry Monroe')</td><td>3. Додати ще один повний рядок даними на свій вибір
4. Додати рядок де задано тільки id та ProductName</td></tr>
<tr><td><u>Унікальні дані DISTINCT (без повторень)</u>
SELECT DISTINCT Quantity</td><td>5. Показати виробників Manufacturer <u>без повторень</u></td></tr>
</table>

FROM Sumproduct	
ОБЧИСЛЮВАЛЬНІ ПОЛЯ	
SELECT Product, Amount+Quantity FROM Sumproduct	6. Показати стовпчик назва товару та обчислити виручку по товару (кількість помножити на ціну)
ЗАДАННЯ НАЗВ РЕЗУЛЬТУЮЧИХ СТОВПЦІВ	
Використання псевдонімів - AS SELECT DISTINCT Product, Amount+Quantity **AS** AvgPrice FROM Sumproduct	7. Підправити попередній запит задавши псевдонім Sum для обчислювального поля
	8. Створити запит: товари, їх стара ціна і їх підвищена 30% ціна та посортувати по спаданню ціни
	9. Показати три найдорожчі товари
	10. Показати всі товари всіх виробників крім Samsung
	11. Показати всі iPhone
АГРЕГАТНІ ФУКЦІЇ	
AVG()	12. Показати середнє значення ціни, назвати стовпчик middle
SELECT MIN(ProductCount) FROM Products	13. Мінімальну ціну, назвати стовпчик for_free
COUNT(*)	14. Порахувати кількість рядків, тобто кількість товарів в таблиці, назвати стовпчик total
	15. Скільки різних виробників є в таблиці
	16. Порахувати суму всієї виручки по всій таблиці (суму всіх добутків ціни та кількості)
Count, where	17. Для скількох товарів не вказано ціну
	18. Знайти кількість товарів 'Apple'

ЛАБОРАТОРНА РОБОТА №3.
СТВОРЕННЯ ТАБЛИЦЬ

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

КЛЮЧІ

Ключ — це стовпець (може бути декілька стовпців), що додається до таблиці і дозволяє встановити зв'язок із записами в іншій таблиці.

Існують ключі двох типів: первинні і вторинні (зовнішні).

Первинний ключ — це одне або кілька полів (стовпців), комбінація значень яких однозначно визначає кожний запис у таблиці. Первинний ключ не допускає значень Null і завжди повинен мати унікальний індекс. Первинний ключ використовується для зв'язування таблиці з зовнішніми ключами в інших таблицях.

Зовнішній (вторинний) ключ — це одне або кілька полів (стовпців) у таблиці, що містять посилання на поле або поля первинного ключа в іншій таблиці. Зовнішній ключ визначає спосіб об'єднання таблиць.

З двох логічно пов'язаних таблиць одну називають таблицею первинного ключа або головною таблицею, а іншу таблицею вторинного (зовнішнього) ключа або підпорядкованою таблицею. СУБД дозволяють зіставити споріднені записи з обох таблиць і спільно вивести їх у формі, звіті або запиті.

Існує три типи первинних ключів: ключові поля лічильника (лічильник), простий ключ і складовий ключ.

Поле лічильника (Тип даних «Счетчик»). Для кожного запису цього поля таблиці автоматично заноситься унікальне числове значення.

Простий ключ. Якщо поле містить унікальні значення, такі як коди чи інвентарні номери, то це поле можна визначити як первинний ключ. В якості ключа можна визначити всі поля, що містить дані, якщо це поле не містить повторювані значення або значення Null.

Складові ключі. У випадках, коли неможливо гарантувати унікальність значень кожного поля, існує можливість створити ключ, що складається з декількох полів. Найчастіше така ситуація виникає для таблиці, використовуваної для зв'язування двох таблиць відношенням *«багато — до — багатьох»*.

Необхідно ще раз відзначити, що в полі первинного ключа повинні бути тільки унікальні значення в кожному рядку таблиці, тобто збіг не допускається, а в полі вторинного або зовнішнього ключа збіг значень у рядках таблиці допускається.

Якщо виникають труднощі з вибором потрібного типу первинного ключа, то в якості ключа доцільно вибрати поле лічильника.

Типи даних

Текстові типи даних:

Тип	Опис
CHAR(size)	Містить рядок фіксованої довжини (може містити букви, цифри, та інші символи). Фіксована довжина задається в дужках. Може зберігати до 255 символів.
VARCHAR(size)	Містить рядок змінної довжини. Найбільша довжина задається в дужках. Може зберігати до 255 символів. Примітка: Якщо ви покладете туди значення більше за 255, тип буде перетворений на TEXT.
TINYTEXT	Рядок з найбільшою довжиною 255 символів
TEXT	Зберігає рядок з найдовшою довжиною 65,535 символів
BLOB	Великий двійковий об'єкт (Binary Large OBject). Зберігає до 65,535 байт даних
MEDIUMTEXT	Зберігає рядок з максимальною довжиною в 16,777,215 символів.
MEDIUMBLOB	Великий двійковий об'єкт. 16 Мегабайт даних
LONGTEXT	Рядок з найбільшою довжиною в 4,294,967,295 символів.
LONGBLOB	Великий двійковий об'єкт. 4 Гігабайти даних
ENUM(x,y,z,i т.д.)	Дозволяє ввести список можливих значень. Можна перелічити до 65535 різних значень типу. Якщо значення що вставляють в поле не буде належати списку, вставиться порожнє значення. Зауваження: Значення будуть відсортовані в тому порядку в якому ви їх запишете. Можливі значення вводяться в такому форматі: ENUM('X','Y','Z')
SET	Подібно до ENUM окрім того, що SET може містити до 64 значень списку, і не може зберігати більше одного вибору.

Числові типи:

Тип	Опис
TINYINT(size)	Цілий від -128 до 127 . Від 0 до 255 UNSIGNED[1]. Максимальне число цифр задається в дужках.
SMALLINT(size)	Від -32768 до 32767. Від 0 до 65535 UNSIGNED[1]. Максимальне число цифр задається в дужках.
MEDIUMINT(size)	Від -8388608 до 8388607. Від 0 до 16777215 UNSIGNED[1]. Максимальне число цифр задається в дужках.

INT(size)	Від -2147483648 до 2147483647. Від 0 до 4294967295 UNSIGNED[1]. Максимальне число цифр задається в дужках.
BIGINT(size)	Від -9223372036854775808 до 9223372036854775807. Від 0 до 18446744073709551615 UNSIGNED[1]. Максимальне число цифр задається в дужках.
FLOAT(size,d)	Число з плаваючою крапкою. Максимальне число цифр задається в параметрі size. Максимальне число цифр після десяткової крапки задається в параметрі d.
DOUBLE(size,d)	Точніше число з плаваючою крапкою. Максимальне число цифр задається в параметрі size. Максимальне число цифр після десяткової крапки задається в параметрі d.
DECIMAL(size,d)	DOUBLE, що зберігається як рядок з фіксованою крапкою.. Максимальне число цифр задається в параметрі size. Максимальне число цифр після десяткової крапки задається в параметрі d.

Типи дати і часу

Тип	Опис
DATE()	Дата. Формат: YYYY-MM-DD Зауваження: Підтримується діапазон від '1000-01-01' до '9999-12-31'
DATETIME()	Формат: YYYY-MM-DD HH:MM:SS [2]. Зауваження: Підтримується діапазон від '1000-01-01 00:00:00' до '9999-12-31 23:59:59'
TIMESTAMP()	Значення TIMESTAMP зберігаються як кількість секунд з початку епохи Unix ('1970-01-01 00:00:00' UTC). Формат: YYYY-MM-DD HH:MM:SS[2] Зауваження: Підтримується діапазон від '1970-01-01 00:00:01' UTC до '2038-01-09 03:14:07' UTC
TIME()	Час. Формат: HH:MM:SS Зауваження: Підтримується діапазон від '-838:59:59' до '838:59:59'
YEAR()	Рік в двоцифровому, або чотирицифровому форматі. Зауваження: Значення, що дозволені в чотирицифровому форматі: від 1901 до 2155. Значення дозволені в двоцифровому форматі: від 70 до 69, що відповідає 1970 та 2069.

2. **ПРАКТИЧНІ ІНДИВІДУАЛЬНІ ЗАВДАННЯ:**

Створити таблицю з шістьма полями (типи integer, char(), date і т.д.) на тему згідно свого номера:

1. Ювелірний магазин
2. Магазин посуду
3. Магазин аудіо та відео апаратури
4. Меблевий магазин
5. Магазин сувенірів
6. Автосалон
7. Магазин солодощів
8. Магазин іграшок
9. Магазин спецій
10. Зоомагазин
11. Молочний магазин
12. Кондитерський магазин
13. Овочевий магазин
14. Магазин фруктів
15. Магазин виробів з дерева
16. Магазин одягу
17. Магазин спорттоварів
18. Магазин техніки
19. Магазин книг
20. Магазин автомобільний
21. Магазин фотоапаратури
22. Магазин сухофруктів
23. Іграшковий магазин
24. Абітурієнти для вступу в універ
25. Облік вчителів школи
26. Бібліотека
27. Обладнання шкільного комп'ютерного класу
28. Магазин меблів
29. Картинна галерея
30. Магазин канцтоварів

Заповнити таблицю 10 рядками-даними.

Вимоги:

два рядки повинні бути **не повністю** заповненими та містити **порожні** клітинки (наприклад, у стовпчику ціна товару не вказати ціну даного товару).

В одному стовпці повинні обов'язково **повторюватись** значення

Повинен бути стовпчик з типом **date** (‘2023-09-27’, ‘рік-місяць-день’ в лапках)

а) Посортувати по одному з Ваших стовпців

b) Показати товари(об'єкти) з ціною визначеного діапазону (наприклад від 100 до 200)
c) Показати товари визначеної категорії
d) Показати товар на задану букву
e) Показати в кого не вказана ціна
f) Знайти максимальну ціну
g) Вивести кількість рядків у Вашій таблиці

ЛАБОРАТОРНА РОБОТА 4
ПІДЗАПИТИ. ГРУПУВАННЯ

1. Теоретичні відомості

Підзапит

Підзапит – це інструмент створення тимчасової таблиці, вміст якої обробляється зовнішнім оператором. Внутрішні запити можуть бути використані для обчислення значень після пропозиції *SELECT* та в умовах пошуку після операторів порівняння (=, <,>, >=, <=, <<) або *IN* у пропозиціях *WHERE* і *HAVING* зовнішнього оператора *SELECT*. Крім того, підзапити можуть застосовуватися в операторах *INSERT*, *UPDATE* і *DELETE*. Текст підзапита береться у дужки.

Існує три види підзапитів:

1. скалярний – повертає таблицю, що складається з одного стовпця й одного рядка, тобто одного значення; він може використовуватися усюди, де потрібно навести одне значення;
2. рядковий – повертає значення декількох стовпців таблиці, але у вигляді єдиного рядка; він може використовуватися всюди, де застосовується конструктор рядкових значень;
3. табличний – повертає значення одного або більше стовпців таблиці, розміщених в декількох рядках; він може використовуватися всюди, де допускається вказувати множини значень, наприклад, з оператором *IN*.

До підзапитів установлюються такі правила й обмеження:

1. У підзапитах не повинна використовуватися фраза *ORDER BY*, хоча вона може бути у зовнішньому запиті.
2. Список у пропозиції *SELECT* має складатися з імен окремих стовпців або складених з них виразів, за винятком випадку, коли підзапит є операндом оператора *EXISTS*.
3. За замовчуванням імена стовпців у підзапиті відносяться до таблиці, ім'я якої є в пропозиції *FROM*. Однак допускається посилатися й на стовпці таблиці, зазначеної у *FROM* зовнішнього запиту, для чого використовуються кваліфіковані імена стовпців. Такі підзапити називаються співвіднесеними.
4. Якщо підзапит є одним із двох операндів, що брали участь в операції порівняння, то він має наводитися в правій частині цієї операції.

Коли неможливо обійтися одним підзапитом, тоді в ньому використовують вкладений підзапит і т.д.

Скалярні підзапити

Скалярні підзапити повертають таблицю, що складається з одного стовпця й одного рядка. Таку таблицю можна одержати, якщо після пропозиції *SELECT* вказати один стовпець, а в пропозиції *WHERE* умову пошуку створити за допомогою потенційного ключа.

ПРИКЛАД

Виведіть на екран усі дати повернення книжок читачем з номером читацького квитка 28

```
SELECT ReturnDate, FactreturnDate
   FROM BookGiveOutRecord
   WHERE ReaderCode = (SELECT Code
                       FROM Readers
                       WHERE ReaderCardNumber = 28)
```

Внутрішній оператор

```
SELECT Code
   FROM Readers
   WHERE ReaderCardNumber = 28)
```

призначений для визначення коду читача з номером квитка 28. Після цього працює зовнішній запит. Інакше кажучи, внутрішній оператор *SELECT* повертає таблицю, що має єдине значення – *Code* = 2. У результаті зовнішній оператор *SELECT* набуває такого вигляду:

```
SELECT ReturnDate, FactreturnDate
   FROM BookGiveOutRecord
   WHERE ReaderCode = 2
```

Дати повернення книг читачем, у якого номер читацького квитка 28

ReturnDate	FactreturnDate
25-SEP-04	24-SEP-04

Таблицю з одним стовпцем і одним рядком можна одержати, якщо після *SELECT* вказати одну агрегатну функцію без використання угруповання записів у пропозиції *GROUP BY*. У цьому разі умови відбору які наведені в *WHERE* і *HAVING*, не відіграють неякої ролі.

ПРИКЛАД

Складіть список інвентарних номерів книжок, вартість яких вище від середньої. Вкажіть, наскільки їхня вартість вище від середньої вартості всіх книжок бібліотеки

```
SELECT InventoryNumber,
          Cost – (SELECT AVG(Cost)
                  FROM BookInventoryNumbers) AS Cost_diff
   FROM BookInventoryNumbers
   WHERE Cost > (SELECT AVG(Cost)
                 FROM BookInventoryNumbers)
```

Інвентарні номери книг, ціна яких вище середньої по бібліотеці, та різниця між їхньою та середньою ціною

InventoryNumber	Cost_diff
4678532	18.51
7569832	35.23
5478956	6.83
2145876	20.98
5268933	35.93
7812639	9.86

Слід зазначити, що не можна вживати «*WHERE Cost > AVG*(*Cost*)», оскільки використовувати агрегатні функції в пропозиції *WHERE* заборонено. Для досягнення бажаного результату слід створити підзапит, в якому розраховують середню вартість книжок, а потім застосувати його у зовнішньому операторі *SELECT*, який призначений для вибірки відомостей про ті книжки, вартість яких перевищує середню. Інакше кажучи, підзапит повертає середню вартість книжок бібліотеки: 38,27 грн. Результат виконання цього скалярного підзапиту використовується в зовнішньому операторі *SELECT* як для розрахунку відхилення вартості від середньої, так і для відбору відомостей про книжки. Тому зовнішній оператор *SELECT* має такий вигляд:

```
SELECT InventoryNumber, Cost – 38.27 AS Cost_diff
  FROM BookInventoryNumbers
  WHERE Cost > 38.27
```

Групування в підмножини - GROUP BY

Оператор GROUP BY (групувати) дозволяє групувати записи в підмножини, які визнаються однаковими значеннями якого-небудь поля.

Нехай потрібно знайти максимальне значення оцінки, отриманої кожним студентом:

Наприклад:

```
SELECT student_id, MAX(mark)
  FROM exam_marks
  GROUP BY student_id;
```

Параметр, по якому здійснюється групування, необхідно вказати в вивідному списку для ідентифікації агрегованих значень групи.

В конструкції GROUP BY для групування може бути використано більше одного стовпчика. Спочатку йде формування групи по значеннях першого стовпчика, а всередині групи – по значеннях другого стовпчика : :

Наприклад:

```
SELECT student_id, subj_id, MAX(mark)
  FROM exam_marks
  GROUP BY  student_id, subj_id;
```

Угруповання записів використовується тоді, коли треба отримати агреговані значення не для всього відношення, а окремо по кожній групі кортежів, які характеризуються загальною ознакою. Найчастіше це однакові значення в якому-небудь атрибуті або групі атрибутів. Тоді в операторі *SELECT* використовують пропозицію

GROUP BY *стовпець[, стовпець ...]*

При цьому необхідно, щоб один зі стовпців результуючого набору даних оброблявся агрегатною функцією.

ПРИКЛАД

Визначіть кількість книжок, що зберігаються в різних бібліотечних фондах, а також їх сумарну вартість

SELECT *FundCode,* ***COUNT****(BookCode)* ***AS*** *cCount,* ***SUM****(Cost)* ***AS*** *cSum*
FROM *BookInventoryNumbers*
GROUP BY *FundCode*
ORDER BY *FundCode*

Нема сенсу включати імена стовпців *BookCode* і *Cost* у список фрази *GROUP BY*, оскільки у списку пропозиції *SELECT* вони використовуються тільки в узагальнюючих функціях. Однак, стовпець *FundCode* у списку пропозиції *SELECT* не зв'язаний із жодною узагальнюючою функцією, а тому обов'язково має бути зазначений у фразі *GROUP BY*.

Сумарна кількість та вартість книжок, що зберігаються у різних бібліотечних фондах

FundCode	cCount	cSum
1	9	307.95
2	6	266.05

При обробці цього запиту логіка виконання операцій така.

Логіка роботи агрегатних функцій при використовуванні угруповання строк таблиці

FundCode	BookCode	Cost	COUNT(BookCode)	SUM(Cost)
1	1	15.56	9	307.95
1	2	22.33		

Логіка роботи агрегатних функцій при використовуванні угруповання строк таблиці

FundCode	BookCode	Cost	COUNT(BookCode)	SUM(Cost)
1	3	34.01		
1	4	12.99		
1	9	36.05		
1	10	74.2		
1	12	36.69		
1	13	48.13		
1	14	27.99		
2	5	56.78	6	266.05
2	6	10.10		
2	7	73.50		
2	7	45.10		
2	8	59.25		
2	11	21.32		

Рядки таблиці *BookInventoryNumbers* поділяються на групи згідно зі значеннями коду фонду (*FundCode*). У нашому прикладі буде створено дві групи, за кількістю унікальних значень у стовпці *FundCode*. Для кожної із груп розраховується загальне число рядків, яке дорівнює кількості кодів книжок, а також сума їх вартості (стовбець *Cost*), що нас цікавить. Потім генерується єдиний зведений рядок для всієї групи вихідних рядків. Нарешті, отримані рядки результуючої таблиці перегруповуються у порядку зростання номера коду фонду, який зазначений у стовпці *FundCode*.

2. ПРАКТИЧНІ ЗАВДАННЯ

Створити наступні запити в таблиці
https://sql.js.org/examples/GUI/index.html

```
DROP TABLE IF EXISTS zoo;
CREATE TABLE zoo( id integer, name text,type char(10), delivered date, price float, country integer);

 INSERT INTO zoo VALUES (1,'JOHNSON','DOG','1990-12-17',1800,NULL);
 INSERT INTO zoo VALUES(2,'MEGGY','CAT','1999-11-16',4500,'Kongo');
INSERT INTO zoo VALUES(3,'SANNY','FISH','2000-10-17',7500,'USA');
```

```
INSERT INTO zoo VALUES(4,'WILL','CAT','1997-03-04',NULL,'Germany');
INSERT INTO zoo VALUES(5,'HILL','FISH','2011-01-16',5000,NULL);
INSERT INTO zoo VALUES(6,'SIN','DOG','2003-05-10',7856,'USA');
INSERT INTO zoo VALUES(7,'MIN','FISH','2005-08-07',1256,NULL);
INSERT INTO zoo VALUES(8,'KIB','CAT','2010-09-10',NULL,'Belgium');
INSERT INTO zoo VALUES(9,'LINNY','FISH','2015-04-17',9876,'Turkey');
INSERT INTO zoo VALUES(10,'LOBBY','DOG','2013-06-
15',5000,'Germany');
```

ПІДЗАПИТИ ПІСЛЯ WHERE

1. Знайти максимальну вартість тварини в таблиці
2. Показати через підзапит дані про найдорожчу тварину, використовуючи попередній запит:

Select * from zoo where price=(Попередній запит)

3. Знайти середню вартість по таблиці
4. Показати через підзапит дані про тварин, в яких вартість менша за середню (попередній запит використати)
5. Вивести країни звідки закуплено котів
6. Показати дані про тварин, яких закуплено в тих самих країнах, що і котів через попередній запит

ПІДЗАПИТИ ПІСЛЯ FROM

7. Вивести всі дані про тварин, що закуплено в USA
8. Показати максимальну ціну американських тварин через підзапит:

Select _____ from (Попередній запит)

9. Показати країни без повторень
10. Порахувати кількість країн без повторень не використовуючи підзапити

ПІДЗАПИТИ ПІСЛЯ SELECT

11. Показати мінімальну вартість в таблиці
12. Показати дані про тварин та наскільки][вартість перевищує мінімальну вартість, тобто різницю його ціни та мінімальної ціни

Select _____ (Попередній запит) from zoo

ГРУПУВАННЯ

Поле по якому групується

SELECT Product, SUM(Quantity) **FROM** Sumproduct **GROUP BY** Product

Агрегатна функція

13.Порахувати кількість тварин **кожної країни**

14.Яка в середньому ціна на **кожен** type тварин

Агрегативні функції, Групування	Записати запити (6-7) https://sqlzoo.net/wiki/SUM_and_COUNT
6.	Для кожного континенту покажіть його назву та кількість країн.
7.	Для кожного континенту покажіть його назву та кількість країн з населенням не менше 10 мільйонів

ЛАБОРАТОРНА РОБОТА №5.
HAVING. МОДИФІКАЦІЯ ДАНИХ

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Умова відбору згрупованих записів наводиться в пропозиції

HAVING *<умови пошуку>*

Тут умови пошуку задають згідно з тими самими правилам, що й у пропозиції *WHERE*. *HAVING* та *WHERE* можна сумісно використовувати у запиті. Спочатку пропозиція *WHERE* виключає з розрахунку рядки, які не відповідають її умовам пошуку. Далі працюють умови пошуку пропозиції *HAVING*. Вони відфільтровують з результуючого набору даних групи рядків, де підсумкові значення не відповідають їм.

Стандарт ISO вимагає, щоб імена стовпців, які використовуються у пропозиції *HAVING*, обов'язково були в списку *GROUP BY* або оброблялися агрегатними функціями. На практиці умови пошуку у пропозиції *HAVING* завжди включають, щонайменше, одну агрегатну функцію. Умови пошуку, що не містять агрегатних функцій, мають входити до пропозиції *WHERE* і застосовуватися для відбору окремих рядків. В умовах пошуку пропозиції *WHERE* застосовувати агрегатні функції заборонено.

ПРИКЛАД

Для кожного бібліотечного фонду, де зберігається більше шести книжок, визначіть їх кількість та сумарну вартість (табл. 6.22).

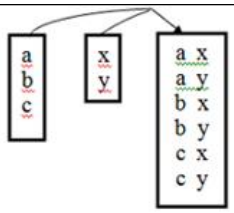
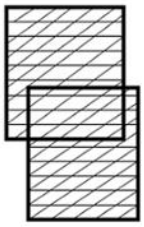
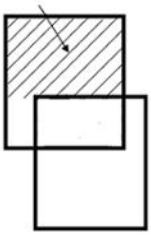
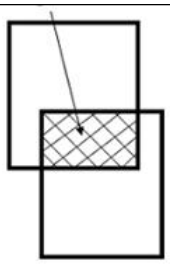
```
SELECT FundCode, COUNT(BookCode) AS cCount, SUM(Cost) AS cSum
  FROM BookInventoryNumbers
  GROUP BY FundCode
  HAVING COUNT(BookCode) > 6
  ORDER BY FundCode
```

Цей приклад аналогічний попередньому, але тут використовуються додаткові обмеження, що вказують на те, що нас цікавлять відомості тільки про ті фонди, у яких зберігається більше шести книжок. Подібна вимога накладається на групи, тому в запиті слід використовувати фразу *HAVING*.

Кількість та сумарна вартість книжок бібліотечного фонду, де зберігається більше шести примірників

FundCode	cCount	cSum
1	9	307.95

РЕЛЯЦІЙНА АЛГЕБРА
Оператори реляційної алгебри вам знайомі з теорії множин, тут приклади їх реалізації в SQL

Оператор декартового добутку		Реляційна алгебра: A Times B	Оператор SQL: SELECT A.Поле1, A.Поле2, ..., B.Поле1, B.Поле2, ... FROM A, B;
Оператор об'єднання		Реляційна алгебра: A Union B	Оператор SQL: SELECT * FROM A **UNION** SELECT * FROM B;
Оператор різниця		Реляційна алгебра: A Minus B	Оператор SQL: SELECT * FROM A **EXCEPT** SELECT * FROM B
Оператор перетину		Реляційна алгебра: A Intersect B	Оператор SQL: SELECT * FROM A **INTERSECT** SELECT * FROM B;

INTERSECT (перетин) - це оператор, який виводить одинакові рядки з першого, другого та наступних наборів даних. Використовують коли потрібно дізнатися, які рядки є і в першій і в другій таблиці (для прикладу, повтор даних).

Як і у оператора UNION, у INTERSECT є правила, наприклад, те, що кількість полів результуючих наборів має бути однакова, як і їх кількість.

EXCEPT (різниця) – використовують коли потрібно порівнювати дві таблиці та вивести тільки ті рядки першої таблиці, яких нема в іншій.

2. ПРАКТИЧНІ ЗАВДАННЯ ДЛЯ ВИКОНАННЯ

Групування	Перейти за посиланням https://sql.js.org/examples/GUI/index.html
	1. Для кожного department підрахувати кількість працівників
	2. Для кожного department показати сума всіх його зарплат

HAVING Оператор HAVING здійснє фільтрацію **груп.** Оператор **HAVING** дуже подібний до оператора **WHERE**, проте: **WHERE** фільтрує дані до того, як вони будуть згруповані, а **HAVING** - здійснює фільтрацію після групування.	
SELECT Product, SUM(Quantity) AS **Product _num** FROM **Sumproduct** GROUP BY **Product** HAVING **SUM(Quantity)**>4000	3. Вивести department, де кількість працівників перевищує 2 (підправити запит 1) 4. В яких department сумарна величина зарплат перевищує 100 тисяч (підправити запит 2)

Модифікація даних

ALTER TABLE Customers ADD Email varchar(255) default ‘A’;	1. Додати стовпчик country текстового типу з текстом по замовчуванні 'Ukraine'
ALTER TABLE Customers DROP COLUMN Email;	2. Знищити стовпчик hire_date
ALTER TABLE *table_name* RENAME COLUMN *old_name* to *new_name*;	3. Перейменувати стовпчик salary в money
UPDATE Customers SET ContactName = 'Alfred Schmidt', City= 'Frankfurt' WHERE CustomerID = 1;	4. Для працівника 'Dave Brown' поставити значення country ‘France’
	5. Змінити дані для всіх працівників за допомогою запиту: зменшити їх зарплату на 1000
DELETE FROM Customers WH	6. Знищити з таблиці всіх інженерів

ERE CustomerName='Alfreds Futterkiste';	

Реляційна лагебра:

1. Створити таблиці

```
DROP TABLE IF EXISTS employees;
CREATE TABLE employees( id  integer,  name    text,  designation text);

INSERT INTO employees VALUES
(1,'JOHNSON','ADMIN'),(2,'HARDING','MANAGER'),
(3,'TAFT','SALES I'),  (4,'HOOVER','SALES I'), (5,'LINCOLN','TECH'),
(6,'GARFIELD','MANAGER'), (7,'GOBLIN','MANAGER');

DROP TABLE IF EXISTS managers;
CREATE TABLE managers( id         integer,  name    text,
                       designation text,     manager integer,
                       hired_on    date,     salary  integer,
                       commission  float,    dept    integer);
  INSERT INTO managers VALUES (2,'HARDING','MANAGER',9,'1998-02-
02',52000,300,3);
  INSERT INTO managers VALUES (6,'GARFIELD','MANAGER',9,'1993-05-
01',54000,NULL,4);
  INSERT INTO managers VALUES (10,'FILLMORE','MANAGER',9,'1994-08-
09',56000,NULL,2);
```

2. Вивести прізвища всіх працівників з двох таблиць (їх об'єднання)

Union Union all	Яка різниця між командами

3. Скільки є таких працівників (написати через підзапит)

4. Хто згадується в обох таблицях? (написати двома способами: intersect та через підзапит)

SQLite, Postgresql **intersect**	**MySql** SELECT products.category_id FROM products WHERE products.category_id **IN** (SELECT inventory.category_id FROM inventory);

5. Про яких працівників нема інформації у першій таблиці, але є в другій (написати двома способами)

SQLite, Postgresql **except**	**MySql** SELECT File_Name FROM Words_DB

	WHERE File_Name **NOT IN** (SELECT File_Name FROM Files_DB)

Завдання для самостійного виконання

```
DROP TABLE IF EXISTS the_first;
CREATE TABLE the_first(
      id int NOT NULL,
      tip varchar(50) NULL,
      summa varchar(50) NULL);
insert into the_first values (1,'принтер',34);
insert into the_first values (2,'сканер',86);
insert into the_first values (3,'монітор',15);

DROP TABLE IF EXISTS the_second;
CREATE TABLE the_second(
      id int NOT NULL,
      tip varchar(50) NULL,
      summa varchar(50) NULL);
insert into the_second values (1,'принтер',34);
insert into the_second values (2,'сканер',23);
insert into the_second values (3,'монітор',15);
insert into the_second values (4,'системний блок',6);
insert into the_second values (5,'монітор',45);
```

1. Вивести рядки, що є у двох таблицях одночасно
2. Вивести назви товарів, що є в обох таблицях
3. Вивести назви товарів з другої таблиці, які відсутні в першій
4. Вивести рядки з першої таблиці,яких нема в другій
5. Вивести всі товари з двох таблиць без повторень
6. Вивести всі товари з двох таблиць з повтореннями
7. Покупець буде брати обов’язково один товар з однієї таблиці, а другий з другої таблиці. Скільки всього варіантів у нього може бути? Обчисліть за допомогою декартового добутку

ЛАБОРАТОРНА РОБОТА №6
ДАТА І ЧАС

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

MySQL.

Типи даних:

DATE — дата (YYYY-MM-DD)
DATETIME — дата та час (YYYY-MM-DD HH:MM:SS)
TIMESTAMP — дата та час з часовою зоною, автоматично оновлюється при зміні рядка (DEFAULT CURRENT_TIMESTAMP)
TIME — час (HH:MM:SS)
YEAR — рік (YYYY)

Приклади функцій:

```
-- Поточна дата/час
SELECT NOW();       -- дата та час
SELECT CURRENT_DATE;
SELECT CURRENT_TIME;

-- Витяг компонентів
SELECT YEAR(NOW()), MONTH(NOW()), DAY(NOW());
SELECT HOUR(NOW()), MINUTE(NOW()), SECOND(NOW());

-- Додавання/віднімання
SELECT DATE_ADD(NOW(), INTERVAL 7 DAY);
SELECT DATE_SUB(NOW(), INTERVAL 1 MONTH);

-- Форматування
SELECT DATE_FORMAT(NOW(), '%d.%m.%Y %H:%i:%s');
```

PostgreSQL

Типи даних:

DATE — дата
TIMESTAMP [WITHOUT TIME ZONE] — дата та час
TIMESTAMP WITH TIME ZONE — дата та час з часовою зоною
TIME [WITHOUT TIME ZONE] — час

Приклади функцій:

```
-- Поточна дата/час
SELECT NOW();         -- дата та час з часовою зоною
SELECT CURRENT_DATE;
SELECT CURRENT_TIME;

-- Витяг компонентів
SELECT EXTRACT(YEAR FROM NOW()), EXTRACT(MONTH FROM NOW()), EXTRACT(DAY FROM NOW());
```

```
-- Додавання/віднімання
SELECT NOW() + INTERVAL '7 days';
SELECT NOW() - INTERVAL '1 month';

-- Форматування
SELECT TO_CHAR(NOW(), 'DD.MM.YYYY HH24:MI:SS');
```

SQLite не має спеціальних типів DATE або DATETIME.
Дати та часи зберігаються як:

TEXT — у форматі 'YYYY-MM-DD HH:MM:SS'
REAL — число днів з плаваючою точкою від «епохи» 24 листопада 4714 до н.е.
INTEGER — UNIX timestamp (кількість секунд з 1970-01-01 00:00:00 UTC)

Основні функції для роботи з датою та часом

```
-- Поточна дата/час
SELECT DATE('now');          -- тільки дата YYYY-MM-DD
SELECT TIME('now');          -- тільки час HH:MM:SS
SELECT DATETIME('now');      -- дата і час YYYY-MM-DD HH:MM:SS
SELECT JULIANDAY('now');     -- число днів у вигляді REAL
SELECT STRFTIME('%d.%m.%Y %H:%M:%S', 'now'); -- форматування

-- Додавання / віднімання
SELECT DATE('now', '+7 days');        -- через 7 днів
SELECT DATETIME('now', '-1 month');   -- місяць тому

-- Витяг компонентів
SELECT STRFTIME('%Y', 'now') AS year;
SELECT STRFTIME('%m', 'now') AS month;
SELECT STRFTIME('%d', 'now') AS day;
```

2. ПРАКТИЧНІ ЗАВДАННЯ

```
drop table if exists book;
CREATE TABLE book( id integer, title char(40), author char(20), published date, stock integer);
INSERT INTO book VALUES (1,'Scion of Ikshvaku','Amish Tripathi','2015-06-22',2);
INSERT INTO book VALUES (2,'The Lost Symbol','Dan Brown','2010-07-20',3);
INSERT INTO book VALUES (3,'Who?','Robin Sharma','2006-06-15',4);
INSERT INTO book VALUES (4,'Inferno','Dan Brown','2014-05-05',3);
INSERT INTO book VALUES (5, 'The Fault in our Stars','John Green','2015-01-03',3);
```

INSERT INTO book VALUES (6, 'Baby','Mr Smith','2010-05-03',5);

РОБОТА З ДАТОЮ В PostgreSQL ***https://extendsclass.com/postgresql-online.html*** ***https://coderpad.io/languages/postgresql/*** ***Синтаксис*** ***EXTRACT(що витягнути FROM поле типу дата)*** ***що витягнути:*** SECOND, MINUTE, HOUR, DAY, MONTH, YEAR	
select now();	15. Вивести сьогоднішню дату
	16. Посортувати по даті виходу книг
EXTRACT(YEAR FROM поле_з_датою) =...	17. Вивести книги, які видали 2015 року
EXTRACT(month FROM date_m)	18. Скільки книг було опубліковані влітку
	19. Найменший рік видання книг
Підзапит	20. Яка книга (або книги) видана найраніше

РОБОТА З ДАТОЮ В MYSQL https://onecompiler.com/mysql/43yfrbfmc Або **https://extendsclass.com/mysql-online.html#** **або** **https://paiza.io/en/projects/new?language=mysql** ***Синтаксис*** ***MONTH(полу типу дата), YEAR(полу типу дата), MINUTE(полу типу дата) …***	
MONTH(), DAY(), Year() *–місць, день, рік зі стовпчика типу «дата»*	
select now();	21. Вивести сьогоднішню дату
year( )	22. Вивести назви книги та тільки рік їх видання
MONTH()	23. Вивести книги, які опубліковані взимку
Поле_дати>='2000-09-01'	24. Вивести книги, що появились після 2011 року

підзапит (max	25.Вивести дані про найновішу книгу

РОБОТА З ДАТОЮ В SQLITE

https://sql.js.org/examples/GUI/index.html або https://repl.it/languages/sqlite
використовуючи таблицю з попереднього сайту

Синтаксис

strftime(парметр, поле типу дата)

Параметри:

%d	day of month: 00	%s	seconds since 1970-01-01
%f	fractional secons SS.SSS	%S	seconds: 00-59
%H	hou: 00-24	%w	day of week 0-6 with Sunday==0
%j	day of year: 001-366	%W	week of year: 00-53
%J	Julian day number	%Y	year: 0000-9999
%m	month: 01-12		
%M	minute: 00-59		

SELECT datetime('now');	26.Вивести сьогоднішню дату
strftime('%m', datefield) …	27.Вивести працівників, яких прийняли на роботу у вересні
	28.Вивести працівників, які прийняті 1990 та 1994
	29.Якого менеджера прийняли 1998 року
підзапит	30.Вивести кого прийняли найшвидше

Виконати завдання над власною таблицею, що створена в лаб2:

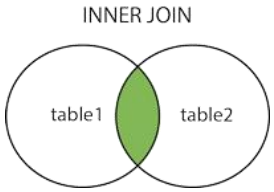
- Вивести товари від найстарішого до найновішого
- Вивести дані про найстаріший товар
- Скільки товарів мають цьогорічну дату
- Погрупувати по роках і обчислити кількість творів для кожного року

ЛАБОРАТОРНА РОБОТА №7.
З'ЄДНАННЯ ТАБЛИЦЬ

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Внутрішнє з'єднання (JOIN or INNER JOIN)

(INNER) JOIN — внутрішнє з'єднання, в результуючому наборі якого присутні тільки записи, значення пов'язаних полів у яких збігаються.



SELECT *column_name(s)*
FROM ***table1***
INNER JOIN ***table2***
ON ***table1.column_name = table2.column_name***;

ЗОВНІШНЄ ЗЄДНАННЯ (OUTER JOIN)

При зовнішньому з'єднанні таблиць є можливість вказати, яка з таблиць буде головною, а яка підлеглою.

Таблиці, які беруть участь у об'єднанні, вказуються ліворуч і праворуч від слова JOIN. Яка з двох таблиць буде головною визначає тип з'єднання:

LEFT — головна таблиця вказана зліва (У лівому зовнішньому з'єднанні результатом є всі рядки лівої таблиці, незалежно від того, чи мають вони відповідну пару в правій таблиці.)

RIGHT — головна таблиця вказана справа (У правому зовнішньому з'єднанні результатом є всі рядки правої таблиці, незалежно від того, чи мають вони відповідність в лівій таблиці.)

FULL OUTER - Повне зовнішнє з'єднання. У ньому результатом є рядки обох таблиць, незалежно від того, чи мають вони відповідності в іншій таблиці. Іншими словами, воно працює так само, як ліве і праве, але на цей раз повертаються значення обох таблиць.

2. ПРАКТИЧНІ ЗАВДАННЯ

1. Перейти в компілятор Mysql
https://onecompiler.com/mysql/43yfrbfmc
Або
https://www.mycompiler.io/new/mysql
Або
https://nextleap.app/online-compiler/sql-programming

2. Додати до існуючої таблиці ще одну таблицю rozdil;
Дано таблицю про працівників

```
DROP TABLE IF EXISTS employees;
CREATE TABLE employees (
  id INTEGER AUTO_INCREMENT PRIMARY KEY,
  name TEXT NOT NULL,
```

```
  dep_id integer,
  salary NUMERIC,
  hire_date DATE);
INSERT INTO employees (name, dep_id, salary, hire_date) VALUES
  ('Alice Smith', 1, 85000, '2020-01-15'),
  ('Bob Johnson', 2, 72000, '2019-03-20'),
  ('Carol Williams', 1, 92000, '2018-11-07'),
 ('Dave Brown', NULL, 115000, '2017-05-12'),
 ('Eve Davis', 1, 110000, '2021-08-30');
```

Дано інформацію про відділі, де вони працюють

```
DROP TABLE IF EXISTS department;
CREATE TABLE department( id integer, dep_name char(15), adress text);
INSERT INTO department VALUES (1,'Engineering','Nadvirna 34'),
(2, 'Marketing', 'Konovalcia 54'),
(3,'Modern','Chornovola 57'),
(4, 'Finance', 'Shechenco 107');
```

3. Об'єднати таблиці за dep_id з першої таблиці та id з другої таблиці й вивести дані з двох таблиць на основі об'єднання (чи всі працівників показав запит?)

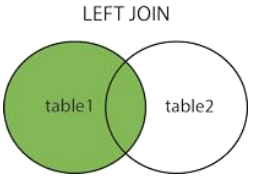
```
SELECT column_name(s)
FROM table1 JOIN table2
ON table1.column_name = table2.column_name;
```

4. Вивести **id працівника, прізвище працівника, назву відділу та адресу**

Зауваження: стовпчик id знаходиться в обох таблицях, то в select вказувати конкретно з якої таблиці обрати id - employees.id або department.id. Інші стовпці не повторюються, томі їх назви можна записувати без вказання таблиці

5. Вивести **ВСІ** прізвища (name) з першої таблиці employees та назви відділів з department

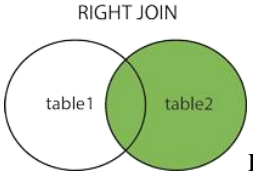
```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```



ВСІ дані з першої таблиці +спільні дані з двох

6. Вивести **ВСІ назви відділів** з другої таблиці та вказати для них прізвища працівників з першої

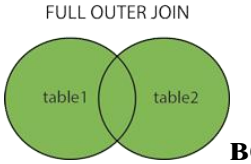
SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2* ON *table1.column_name = table2.column_name*;



всі дані з другої таблиці +спільні дані з двох

Зауваження: **Не всі СУБД підтримують Right I Full join.** Якщо в СУБД не працює right join, то Right з'єднання переписують через left, беручи другу таблицю початковою-першою

7. Задати повну зведену таблицю з id, name по двох таблицях



всі дані з першої та другої таблиць

Не всі СУБД підтримують дану команду, зокрема і Mysql. Записати дану команду через union

Full з'єднання переписують через left та об'єднюють – union	
SELECT * FROM dogs **FULL OUTER** JOIN cats ON dogs.color = cats.color;	SELECT type,color, FROM dogs **LEFT** JOIN cats dogs.color=cats.color **UNION** SELECT type,color, FROM cats **LEFT** JOIN dogs cats.color=dogs.color ;
Зауваження: Запити в мові SQL комбінуються за допомогою оператора UNION. Крапка з комою ставиться тільки після останнього запиту. Використовувати UNION можна тільки при об'єднанні запитів, що відповідають стовпчикам, які сумісні по об'єднанню (стовпці як мінімум повинні відноситись до одного типу)	

CROSS JOIN - Перехресне з'єднання. У цьому кожен рядок з однієї таблиці з'єднується з кожним рядком з іншої таблиці (декартовий добуток). Характерною рисою перехресного з'єднання є відсутність умови ON, як видно з наступного запиту:

SELECT A, B FROM R1 CROSS JOIN R2

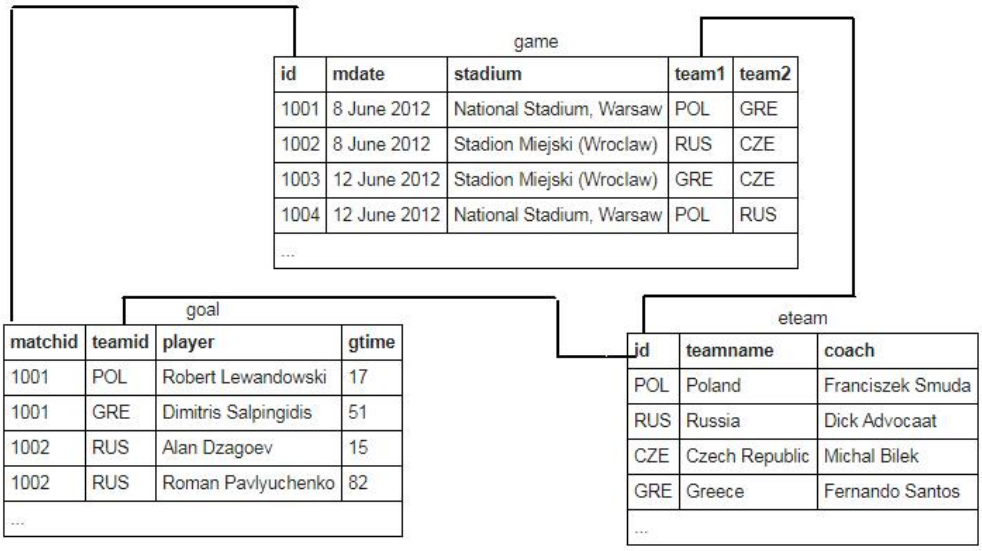
8. Виконати запити

1. Вивести прізвища всі інженерів

2. В якому відділі працює 'Eve Davis'
3. Скільки є працівників з відділу 'Marketing'
4. Скільки працівників працює в кожному відділі
5. В якому відділі найвища зарплата?

Набуття навичок:

https://sqlzoo.net/wiki/The_JOIN_operation



game

id	mdate	stadium	team1	team2
1001	8 June 2012	National Stadium, Warsaw	POL	GRE
1002	8 June 2012	Stadion Miejski (Wroclaw)	RUS	CZE
1003	12 June 2012	Stadion Miejski (Wroclaw)	GRE	CZE
1004	12 June 2012	National Stadium, Warsaw	POL	RUS
...				

goal

matchid	teamid	player	gtime
1001	POL	Robert Lewandowski	17
1001	GRE	Dimitris Salpingidis	51
1002	RUS	Alan Dzagoev	15
1002	RUS	Roman Pavlyuchenko	82
...			

eteam

id	teamname	coach
POL	Poland	Franciszek Smuda
RUS	Russia	Dick Advocaat
CZE	Czech Republic	Michal Bilek
GRE	Greece	Fernando Santos
...		

https://sqlzoo.net/wiki/The_JOIN_operation	1. Змініть запит так, щоб показати номер матчу та ім'я гравця всіх голів, забитих Німеччиною. Щоб визначити німецьких гравців: teamid = 'GER' 2. Покажіть id, stadium, team1, team2 для гри з номером 1012 3. Модифікуйте запит та покажіть player, teamid, stadium, mdate для кожного німецького гола 4. Показати team1, team2, player для кожного гола граця,що називається Mario player LIKE 'Mario%' 5. Показати player, teamid, coach, gtime д ля всіх голів забитих підчас перших 10 хвилин гри gtime<=10 6. Перелічіть дати матчів та назву команди, в якій «Фернандо Сантос» був тренером команди. 7. Вивести список гравців, що забили голна стадіоні **'National Stadium, Warsaw'**

Виконати запити	
SQL Lesson 6: Multi-table queries with JOINs https://sqlbolt.com/lesson/select_queries_with_joins	
Знайдіть для кожного фільму дані про продажі в країні та за кордоном Покажіть для кожного фільму дані продажів, якщо на міжнародному рівні прокат фільму більший ніж на внутрішньому ринку Перелічіть усі фільми з їх рейтингами в порядку спадання рейтингу	
SQL Lesson 7: OUTER JOINs https://sqlbolt.com/lesson/select_queries_with_outer_joins	
Знайдіть список всіх будівель, у яких є співробітники Знайдіть список всіх будівель та їх «capacity» Перерахуйте всі будівлі та посади (role) працівників у кожному будинку (включаючи порожні будівлі)	

ЛАБОРАТОРНА РОБОТА №8.
CASE. SELF JOIN.USING NULL

1. ТЕРЕТИЧНІ ВІДОМОСТІ

1. Визначення

Self JOIN — це приєднання таблиці до самої себе.

Ця операція дозволяє порівнювати рядки однієї таблиці між собою або отримувати пов'язані дані в межах однієї таблиці.

Іншими словами, ми створюємо **дві «віртуальні копії»** однієї таблиці та встановлюємо між ними зв'язок через JOIN.

2. Синтаксис

SELECT <поля>
FROM table_name AS alias1
JOIN table_name AS alias2
ON alias1.field = alias2.field;

alias1 і alias2 — псевдоніми (аліаси) для однієї таблиці.

ON — умова приєднання (JOIN), яка визначає, які рядки зв'язуються.

Приклад: співробітники та менеджери

Таблиця employees**:**

id name manager_id

1 John 3

2 Mary 3

3 Alice NULL

4 Bob 1

Поле manager_id посилається на id іншого співробітника.

Потрібно отримати **ім'я співробітника та ім'я його менеджера**.

3. **SQL-запит із Self JOIN:**

SELECT e.name AS employee, m.name AS manager
FROM employees e
LEFT JOIN employees m ON e.manager_id = m.id;

4. Пояснення прикладу

employees e — перша копія таблиці для рядків співробітників.

employees m — друга копія таблиці для рядків менеджерів.

Умова e.manager_id = m.id дозволяє зіставити кожного співробітника зі своїм менеджером.

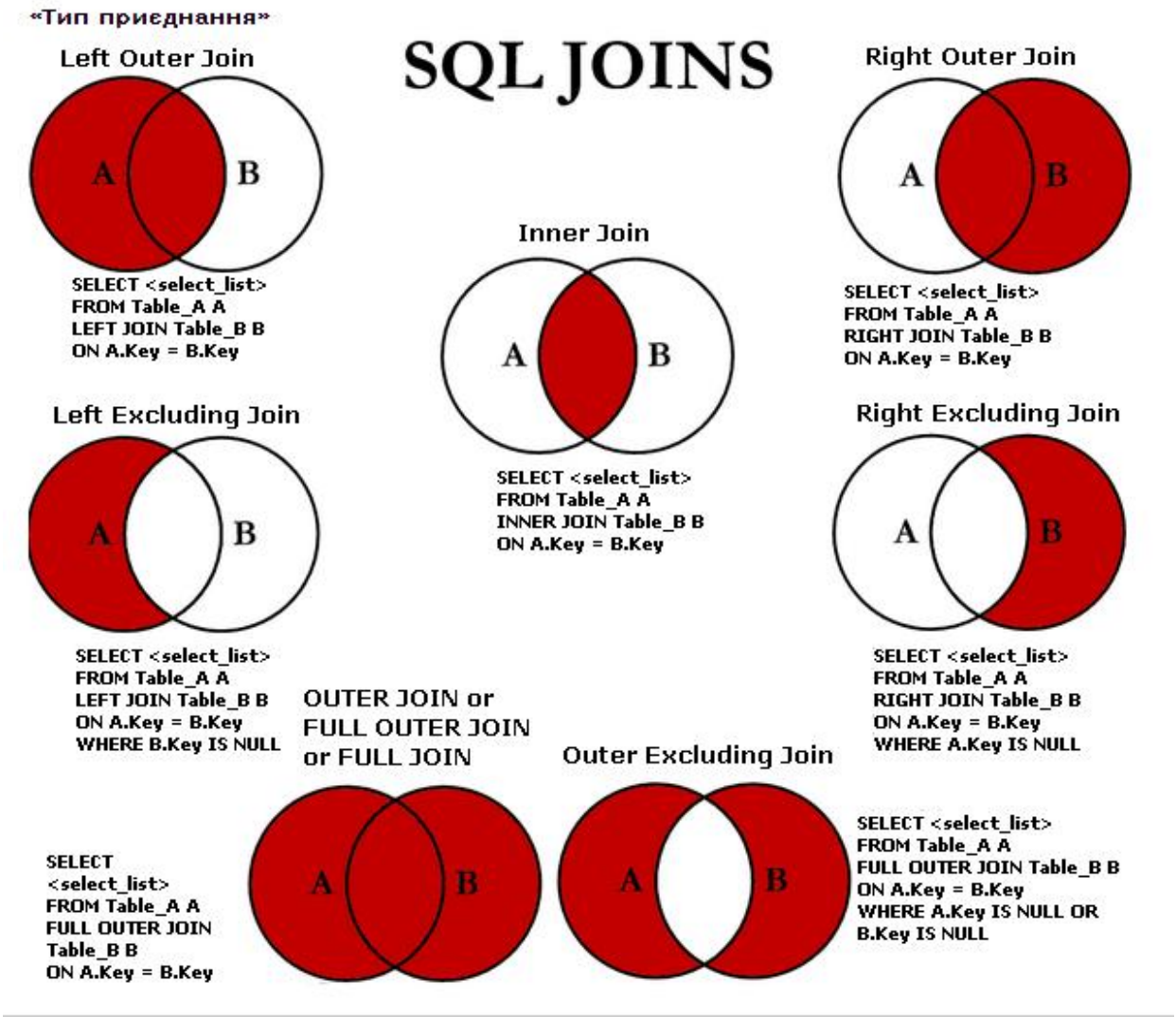
LEFT JOIN використано, щоб зберегти рядки, навіть якщо у співробітника немає менеджера (NULL).

5. Застосування Self JOIN

Побудова **ієрархічних структур** (наприклад, співробітники → менеджери → керівники відділів)

Виявлення **пов'язаних елементів всередині таблиці**

Порівняння або пошук **дублікатів/схожих записів**



ПРАКТИЧНІ ЗАВДАННЯ

```
DROP TABLE IF EXISTS employees;
CREATE TABLE employees( id          integer,  name    text,
                        designation text,     manager integer,
                        hired_on    date,     salary  integer,
                        commission  float,    dept    integer);

  INSERT INTO employees VALUES (1,'JOHNSON','ADMIN',6,'1990-12-
17',18000,NULL,4);
  INSERT INTO employees VALUES (2,'HARDING','MANAGER',9,'1998-02-
02',52000,300,3);
  INSERT INTO employees VALUES (3,'TAFT','SALES I',2,'1996-01-
02',25000,500,3);
  INSERT INTO employees VALUES (4,'HOOVER','SALES I',2,'1990-04-
02',27000,NULL,3);
  INSERT INTO employees VALUES (5,'LINCOLN','TECH',6,'1994-06-
23',22500,1400,4);
```

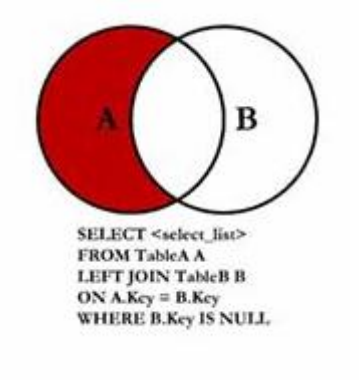
```
INSERT INTO employees VALUES (6,'GARFIELD','MANAGER',9,'1993-05-01',54000,NULL,4);
INSERT INTO employees VALUES (7,'POLK','TECH',6,'1997-09-22',25000,NULL,4);
INSERT INTO employees VALUES (8,'GRANT','ENGINEER',10,'1997-03-30',32000,NULL,2);
INSERT INTO employees VALUES (9,'JACKSON','CEO',NULL,'1990-01-01',75000,NULL,4);
INSERT INTO employees VALUES (10,'FILLMORE','MANAGER',9,'1994-08-09',56000,NULL,2);
INSERT INTO employees VALUES (11,'ADAMS','ENGINEER',10,'1996-03-15',34000,NULL,2);
INSERT INTO employees VALUES (12,'WASHINGTON','ADMIN',6,'1998-04-16',18000,NULL,4);
INSERT INTO employees VALUES (13,'MONROE','ENGINEER',10,'2000-12-03',30000,NULL,2);
INSERT INTO employees VALUES (14,'ROOSEVELT','CPA',9,'1995-10-12',35000,NULL,3);
```

https://onecompiler.com/mysql

CASE	
I варіант	
SELECT NAME, ***CASE*** ***WHEN CREDIT_LIMIT > 6000 THEN 'A'*** ***WHEN CREDIT_LIMIT > 4500 THEN 'B'*** ***WHEN CREDIT_LIMIT > 2500 THEN 'C'*** ***ELSE 'D' END*** *FROM …*	1. Для кожного працівника відобразити стовпчик зі значеннями «+», якщо зарплата більше 30 000, або «-» якщо умова не виконується
II варіант	
SELECT ProductNumber, **CASE ProductLine** **WHEN 'R' THEN 'Road'** **WHEN 'M' THEN 'Mountain'** **WHEN 'T' THEN 'Touring'** **WHEN 'S' THEN 'Other sale items'** **ELSE 'Not for sale' END,** Name FROM Production.Product ORDER BY ProductNumber;	2. Для кожного працівника вивести номер dept словесно: ‘one’, ‘two’ і т.д.

USING NULL

Додати таблицю

	**DROP TABLE IF EXISTS department;** **CREATE TABLE department(id integer, dep_name char(15), room integer);** **INSERT INTO department VALUES** **(1,'Engineering',318),** **(2, 'Marketing', NULL),** **(3,'Modern',232),** **(4, 'Finance', 310),** **(5, 'Commercial', NULL);**
select_____ **from table1 left join table2 on table1.col=table2.co2** **where ____ is null**	3. Вивести дані про працівників першої таблиці для яких не вказано номер кабінету
	4. Вивести назви відділень для яких не вказано працівників

SELF JOIN

SQL Self JOIN Examples

Problem: Match customers that are from the same city and country

CUSTOMER	
Id	PK
FirstName	
LastName	
City	
Country	
Phone	

CUSTOMER	
Id	PK
FirstName	
LastName	
City	
Country	
Phone	

```
SELECT B.FirstName AS FirstName1, B.LastName AS LastName1,
       A.FirstName AS FirstName2, A.LastName AS LastName2,
       B.City, B.Country
  FROM Customer A, Customer B
 WHERE A.Id <> B.Id
   AND A.City = B.City
   AND A.Country = B.Country
 ORDER BY A.Country
```

5. Зробити запит, де вказуються двоє працівників, що працюють в одному dept

Формуємо звідки беремо дані: **from employees a, employees b** (сприймаємо наче працюємо з таблицею та її копією)

Беремо дані з кожної таблиці: select **a.name, b.name, a.dept**

Запишемо умову, що працівники були з одного відділу: **where a.dept=b.dept**

6. Щоб відкинути рядки, де працівник відображається сам з собою допишемо додаткову умову: **and a.id<>b.id**
7. Щоб відкинути повторення

JOHNSON	LINCOLN

LINCOLN	JOHNSON

Підправимо команду and **a.id>b.id**

8. Вивести по двоє працівників, які працюють на тих самих посадах

9. Для **кожного працівника вивести його прізвище та прізвище його менеджера**

У таблиці employees для кожного працівника вказано номер id його менеджера

(1,'JOHNSON','ADMIN',**6**,'1990-12-17',18000,NULL,4);

Тобто для 'JOHNSON' менеджером є працівник під номером 6 - а це (**6**,'GARFIELD','MANAGER',9,'1993-05-01',54000,NULL,4)

Результат виведення буде:

```
+------------+---------+----------+
| name       | manager | name     |
+------------+---------+----------+
| HOOVER     |       2 | HARDING  |
| TAFT       |       2 | HARDING  |
| WASHINGTON |       6 | GARFIELD |
| POLK       |       6 | GARFIELD |
| LINCOLN    |       6 | GARFIELD |
| JOHNSON    |       6 | GARFIELD |
| ROOSEVELT  |       9 | JACKSON  |
| FILLMORE   |       9 | JACKSON  |
| GARFIELD   |       9 | JACKSON  |
| HARDING    |       9 | JACKSON  |
| MONROE     |      10 | FILLMORE |
| ADAMS      |      10 | FILLMORE |
| GRANT      |      10 | FILLMORE |
+------------+---------+----------+
```

Завдання для зарахування:

https://sqlbolt.com/lesson/select_queries_with_nulls

Скористатися таблицею, що створена в лабі 2

10. Додати стовпчик номер складу wh_id integer
11. Для ДЕЯКИХ стовпців поставити номер складу (наприклад, українським товарам задати склад 102, чи товарам з певної категорії встановити номер складу 103, чи поставити товарам 2025 року склад 101)

Залишити деякі товари без вказання номеру складу

Використати наступну таблицю

```
DROP TABLE IF EXISTS warehouse;
CREATE TABLE warehouse( id     integer primary key,  name
VARCHAR(25) NOT NULL,    address VARCHAR(45),
  volume DECIMAL(12,2),
  responsible_person VARCHAR(120),
  contacts VARCHAR(45));
```

```
INSERT INTO warehouse VALUES
(101, 'Центральний склад', 'м. Київ, вул. Промислова, 10', 25000.00,
'Олексій Іванов', '+380671234567, ivanov@company.ua'),
(102, 'Регіональний склад Львів', 'м. Львів, вул. Складська, 5', 12000.00,
'Марія Петренко', '+380932223344, petrenko@company.ua'),
(103,'Склад Одеса-Порт', NULL, 40000.00, 'Іван Сидоренко',
'+380502223355, sydorenko@company.ua');
```

12. Вивести об'єднання даних таблиць
13. Підравити запит, щоб відображались всі товари таблиці з лаби 2
14. Вивести тільки ті товари, в яких не вказано номер складу
15. Вивести номери складів, де нема товарів
16. Вивести по два товари з одного того ж складу, використовуючи self join

ЛАБОРАТОРНА РОБОТА №9.
Об'єднання трьох таблиць

ПРАКТИЧНІ ЗАВДАННЯ

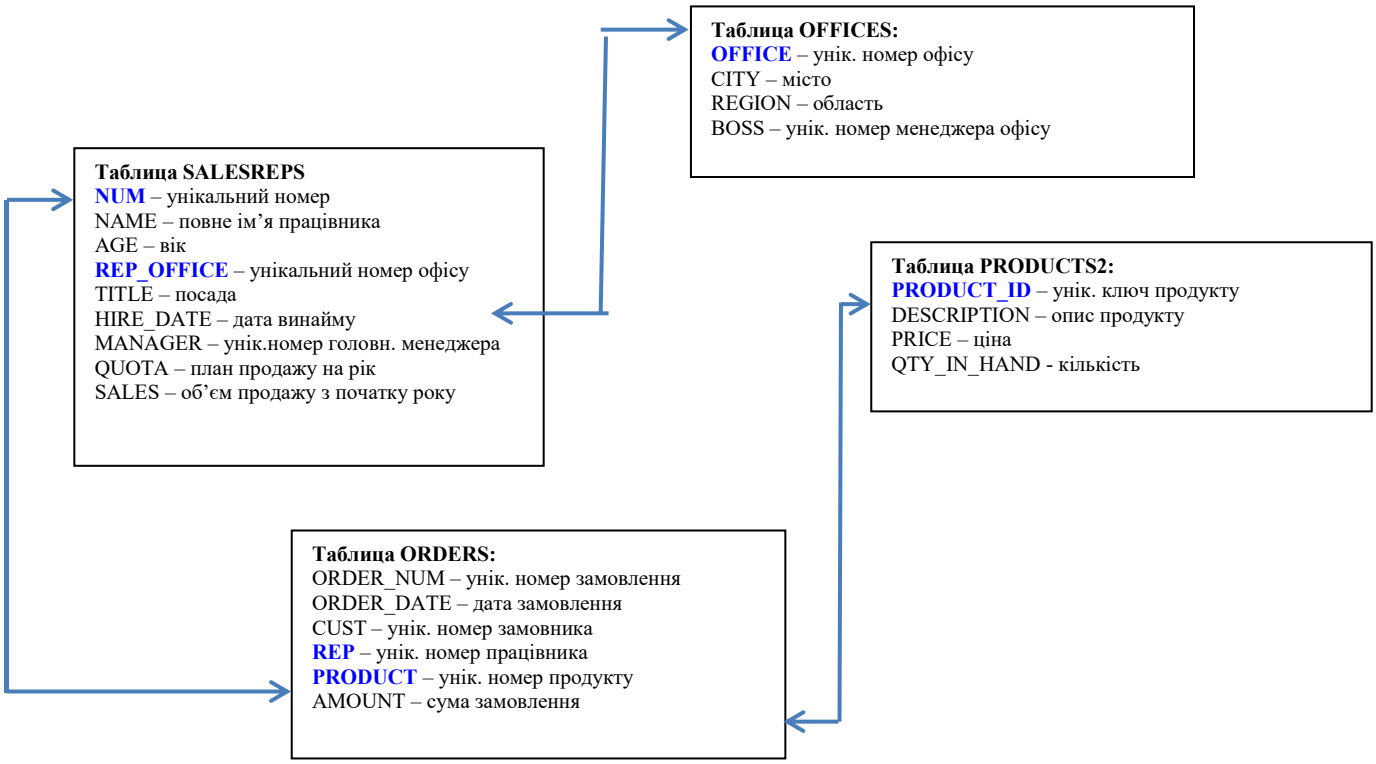
```
DROP TABLE IF EXISTS SALESREPS;
Create table SALESREPS(num integer, name char(40), age integer, rep_office integer, title char(20), hire_date date, manager integer, quota integer, sales integer);
insert into SALESREPS values (100,'Mary Jones',27,51,'East','2012-04-05',104,45000,36791);
insert into SALESREPS values(102,'Sam Clark',22,45,'West','2016-04-08',108,30000,39272);
insert into SALESREPS values(103,'Ymu',34,51,'West','2016-04-08',108,10000,9000);
insert into SALESREPS values(104,'IRen',29,null,'West','2018-04-08', null, 100000, 56000);

drop table if exists PRODUCTS;
create table PRODUCTS(product_id integer, description char(40), price float, qty_in_hand integer);
insert into PRODUCTS values(4, 'Nokia',7600,2);
insert into PRODUCTS values(16, 'Samsung',10300,5);
insert into PRODUCTS values(34, 'Lenovo',1300,7);
insert into PRODUCTS values(83, 'Apple',10000,5);

drop table if exists ORDERS;
create table ORDERS(order_num integer, order_date date, cust integer, rep integer, product integer, amount  integer);
insert into ORDERS values(1,'2024-12-09',123,104, 16, 3000);
insert into ORDERS values(2,'2024-06-13',14,100, 4,2111);
insert into ORDERS values(3,'2024-07-01',56,103, 4,5600);
insert into ORDERS values(4,'2025-03-01',31,103, 16,20500);
insert into ORDERS values(5,'2025-04-28',14,102, 34,1340);

drop table if exists OFFICES;
create table OFFICES(office integer, city char(25), region char(40), boss integer);
insert into OFFICES values(45,'Lviv','Lvivska',102);
insert into OFFICES values(51,'Kiev','Obolon',100);
insert into OFFICES values(78,'Odesa','Kark',100);
```

https://onecompiler.com/mysql



Виконати запити:

1. Коли було здійснено останнє замовлення (order_date)	
2. Для працівників показати місце знаходження їх офісів	+------------+------+---------+ | name | city | region | +------------+------+---------+ | Mary Jones | Kiev | Obolon | | Sam Clark | Lviv | Lvivska | | Ymu | Kiev | Obolon | +------------+------+---------+
3. Скільки є працівників у Києві	
4. Вивести прізвища працівників, які прийняли замовлення цього року	+-----------+ | name | +-----------+ | Ymu | | Sam Clark | +-----------+

5. Для кожного працівника вивести кількість замовлень, що він здійснив	+------+-----------+----------+ | num | name | count(*) | +------+-----------+----------+ | 104 | IRen | 1 | | 100 | Mary Jones | 1 | | 103 | Ymu | 2 | | 102 | Sam Clark | 1 | +------+-----------+----------+
select num, name, … group by num, name;	
6. Вивести ціни для товарів, що є в замовленнях	+-------------+-------+ | description | price | +-------------+-------+ | Samsung | 10300 | | Nokia | 7600 | | Lenovo | 1300 | +-------------+-------+
7. Вивести всі дані про працівника, що оформив найдорожче замовлення	+------+------+--------+ | num | name | amount | +------+------+--------+ | 103 | Ymu | 20500 | +------+------+--------+
8. Якого товару нема в замовленнях	+-------------+ | description | +-------------+ | Apple | +-------------+
9. В якому офісі більше одного працівника	+--------+------+--------+----------+ | office | city | region | count(*) | +--------+------+--------+----------+ | 51 | Kiev | Obolon | 2 | +--------+------+--------+----------+
10. Яка загальна сума всіх замовлень працівника Ymu?	+-------+ | TOTAL | +-------+ | 26100 | +-------+
11. В якому офісі нема працівників	+-------+--------+ | city | region | +-------+--------+ | Odesa | Kark | +-------+--------+

12. Вивести описи тих продуктів (description) в замовленнях, які прийняла 'Mary Jones'	+------------+-------------+ | name | description | +------------+-------------+ | Mary Jones | Nokia | +------------+-------------+
13. Коли та на яку суму були прийняті замовлення у Львівському офісі	+------------+--------+ | order_date | amount | +------------+--------+ | 2025-04-28 | 1340 | +------------+--------+
14. Вивести прізвище працівника, що оформив замовлення з найдешевшим товаром з таблиці товарів	+-----------+ | name | +-----------+ | Sam Clark | +-----------+
15. Для кожного року погрупувати по містах та показати кількість замовлень group by _____, _____;	+------------------+------+----------+ | year(order_date) | city | count(*) | +------------------+------+----------+ | 2024 | Kiev | 2 | | 2025 | Kiev | 1 | | 2025 | Lviv | 1 | +------------------+------+----------+

ЛАБОРАТОРНА РОБОТА №10
ТРИГЕРИ

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Кожен тригер пов’язаний з визначеною таблицею і автоматично запускається при виконанні одного з DML-операторів (INSERT, DELETE, UPDATE) або їх сукупності над цією таблицею.

Конструкція trigger_time вказує момент виконання тригера і може приймати два значення:

1. before-дії тригера проводяться до виконання операції зміни таблиці;
2. AFTER-дії тригера проводяться після виконання операції зміни таблиці.

Конструкція trigger_event показує, на яке з подій повинен реагувати тригер, і може приймати три значення:

1. INSERT - тригер прив'язаний до події вставки нового запису в таблицю;
2. UPDATE - тригер прив'язаний до події оновлення запису таблиці;
3. DELETE - тригер прив'язанй до події видалення записів таблиці.

```
CREATE  TRIGGER trigger_name [BEFORE|AFTER] event_name
ON table_name
BEGIN
 -- Trigger logic goes here....
END;
```

INSERT	NEW references are valid
UPDATE	NEW and OLD references are valid
DELETE	OLD references are valid

2. ПРАКТИЧНІ ЗАВДАННЯ

Задати таблицю в https://onecompiler.com/mysql:

```
DROP TABLE IF EXISTS employees;
CREATE TABLE employees( id          integer AUTO_INCREMENT primary
key,  name    text,
                    designation text,     manager integer,
                    hired_on    date,     salary  integer,
                    commission  float,    dept    integer);
```

```
INSERT INTO employees VALUES (1,'JOHNSON','ADMIN',6,'1990-12-17',18000,NULL,4);
INSERT INTO employees VALUES (2,'HARDING','MANAGER',9,'1998-02-02',52000,300,3);
INSERT INTO employees VALUES (3,'TAFT','SALES I',2,'1996-01-02',25000,500,3);
INSERT INTO employees VALUES (4,'HOOVER','SALES I',2,'1990-04-02',27000,NULL,3);
INSERT INTO employees VALUES (5,'LINCOLN','TECH',6,'1994-06-23',22500,1400,4);
INSERT INTO employees VALUES (6,'GARFIELD','MANAGER',9,'1993-05-01',54000,NULL,4);
INSERT INTO employees VALUES (7,'POLK','TECH',6,'1997-09-22',25000,NULL,4);
INSERT INTO employees VALUES (8,'GRANT','ENGINEER',10,'1997-03-30',32000,NULL,2);
INSERT INTO employees VALUES (9,'JACKSON','CEO',NULL,'1990-01-01',75000,NULL,4);
INSERT INTO employees VALUES (10,'FILLMORE','MANAGER',9,'1994-08-09',56000,NULL,2);
INSERT INTO employees VALUES (11,'ADAMS','ENGINEER',10,'1996-03-15',34000,NULL,2);
INSERT INTO employees VALUES (12,'WASHINGTON','ADMIN',6,'1998-04-16',18000,NULL,4);
INSERT INTO employees VALUES (13,'MONROE','ENGINEER',10,'2000-12-03',30000,NULL,2);
INSERT INTO employees VALUES (14,'ROOSEVELT','CPA',9,'1995-10-12',35000,NULL,3);
```

Створити тригер, що буде при оновленні даних про працівника змінювати його дати прийому на сьогоднішню

1. Створимо тригер та задамо йому назву

```
create trigger hired_on_change
```

2. Тригер буде запускатись перед оновлення даних для працівника **employees**

```
before update on employees
for each row
```

3. У блоці **begin … end** вказуємо команду: поставити для **нового** hired_on сьогоднішню дату

```
set new.hired_on = current_date();
```

4. Змінити роздільник команд: поставити “DELIMITER $$” перед тригером і “end $$ DELIMITER ;” після тригера

```
DELIMITER $$
create trigger hired_on_change
before update on employees
for each row
begin
set new.hired_on = current_date();
end $$
DELIMITER ;
```

5. Перевіримо роботу тригера. Задати оновлення даних для одного з працівників - поставити йому інший номер dept або іншу посаду. Вивести таблицю

Створимо тригери для аудиту дій INSERT та DELETE у таблиці employees - створимо таблицю для фіксації дій прийняття нового працівника та відрахування з роботи працівника з вказанням часу виконання цих дій

Створимо таблицю для аудиту дій

```
DROP TABLE IF EXISTS employees_audit;
CREATE TABLE employees_audit (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    emp_id INT,
    emp_name VARCHAR(25),
    action VARCHAR(10),
    action_time DATETIME DEFAULT CURRENT_TIMESTAMP);
```

А) Створимо спочатку тригер emp_insert, що фіксує в таблиці employees_audit додавання нового працівника

1. Задамо тригер, що буде виконуватись після додаванням нового рядка в employees

```
CREATE trigger emp_insert
after insert on employees
for each row
```

2. У блоці **begin … end** вказуємо команду: у таблицю employees_audit додати новий рядок, де буде вказано id та name нового працівника, слово 'INSERT' у полі **action** і час додавання

```
insert into employees_audit (emp_id, emp_name, action, action_time)
values (new.id, new.name, 'INSERT', now());
```

3. Змінити роздільник команд: поставити "DELIMITER $$" перед тригером і "end $$ DELIMITER ;" після тригера

```
DELIMITER $$
CREATE trigger emp_insert
after insert on employees
for each row
begin
  insert into employees_audit (emp_id, emp_name, action, action_time)
  values (new.id, new.name, 'INSERT', now());
end $$
DELIMITER ;
```

4. Додати до таблиці employees новий рядок з даними. Переглянути employees чи змінилась і переглянути таблицю employees_audit чи дана дія була в ній зафіксована

Б) Створимо спочатку тригер emp_delete, що фіксує в таблиці employees_audit знищення працівника

1. Задамо тригер emp_delete, що буде виконуватись після знищенняо рядка в employees
after delete

2. У блоці **begin … end** вказуємо команду: у таблицю employees_audit додати новий рядок, де буде вказано id та name "старого" працівника, слово 'DELETE' у полі **action** і час знищення

```
insert into employees_audit (emp_id, emp_name, action, action_time)
values (old.id, old.name, 'DELETE', now());
```

3. Змінити роздільник команд: поставити "DELIMITER $$" перед тригером і "end $$ DELIMITER ;" після тригера

4. Знищити в таблиці employees рядок з даними. Переглянути employees чи змінилась і переглянути таблицю employees_audit чи дана дія була в ній зафіксована

Задання для зарахування:

Створити тригер, що фіксуватиме в окремій таблиці переходи працівників від одного dept до іншого

1. Спочатку створимо окрему таблицю-журнал для фіксації переходів працівника. В ній запишемо name працівника, який переходить та номер dept звідки йде та dept куди йде

```
drop table if exists moving;
create table moving (name text, move_from integer, move_to integer);
```

2. Задамо тригер, що буде запускатись перед зміною номеру dept для працівника таблиці

… **before update on employees**…

3. У блоці **begin … end** вказуємо команду записати-створити новий рядок у moving для збереження даних

```
Create trigger dept_change before update on employees
begin
insert into moving values(old.name, old.dept, new.dept);
end;
```

4. Перевіряємо роботу тригера: задаємо запити на зміну dept (наприклад: update employees set dept=4 where id=2; і т.д.).
5. Виводимо таблицю moving

Створити тригер, що фіксуватиме в окремій таблиці з якої посади звільнили працівника і коли

6. Створити таблицю журнал для збереження даних after delete: з якої посади вигнали працівника (знищили рядок) та дату цього звільнення-знищення;

```
old.designation, datetime()
```

7. Перевірити за командою delete from employees where id=5 or id=8

Створити тригер який для таблиці employees при додаванні нових працівників (…after insert on employees…) з їх зарплатами, надає новим менеджерам надбавку 10 процентів до зарплати, та новим інженерам 15 процентів

```
update employees
set salary=salary*1.1
where new.designation='MANAGER' and id=new.id;
```

Лабораторна робота №11
Додаткові функції. Цілісність даних

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Цілісність сутностей

Це обмеження цілісності торкається первинних ключів базових таблиць. За визначенням, первинний ключ - мінімальний ідентифікатор (одно або декілька полів), який використовується для унікальної ідентифікації записів в таблиці. Таким чином, ніяка підмножина первісного ключа не може бути достатньою для унікальної ідентифікації записів.

Цілісність сутностей визначає, що у базовій таблиці жодне поле первинного ключа не може містити відсутніх значень, позначених NULL.

Якщо допустити присутність визначника NULL у будь-якій частині первинного ключа, це рівносильно твердженню, що не усі його поля потрібні для унікальної ідентифікації записів, і суперечить визначенню первинного ключа.

Цілісність посилання

Вказане обмеження цілісності торкається **зовнішніх ключів**. Зовнішній ключ - це поле (чи безліч полів) однієї таблиці, що є ключем іншої (чи тій же самій) таблиці. Зовнішні ключі використовуються для встановлення логічних зв'язків між таблицями. Зв'язок встановлюється шляхом привласнення значень зовнішнього ключа однієї таблиці значенням ключа інший.

Між двома або більше таблицями бази даних можуть існувати стосунки підлеглості, які визначають, що для кожного запису головної таблиці (званою ще батьківською) може існувати одна або декілька записів в підпорядкованій таблиці (званою так же дочірньої).

Існує три різновиди зв'язку між таблицями бази даних :

- **"один-до-багатьох";**
- **"один-до-одного";**
- **"багато-до-багатьох".**

Відношення "один-до-одного" має місце, коли одному запису батьківської таблиці може відповідати декілька записів дочірньої. Зв'язок "один-до-багатьох" іноді називають зв'язком "багато-до-одного". І у тому, і в іншому випадку суть зв'язку між таблицями залишається незмінною.

Зв'язок "один-до-багатьох" найбільш поширений для реляційних баз даних. Вона дозволяє моделювати також ієрархічні структури даних.

Відношення "один-до-одного" має місце, коли одному запису у батьківській таблиці відповідає один запис в дочірній. Це відношення зустрічається набагато рідше, ніж відношення "один-до-багатьох". Його використовують, якщо не хочуть, щоб таблиця БД "розпухала" від другорядної інформації. Використання зв'язку "один-до-одного" призводить до того, що для читання пов'язаної інформації в декількох таблицях

доводиться робити декілька операцій читання замість однієї, коли дані зберігаються в одній таблиці.

Відношення "багато-до-багатьох" має місце в наступних випадках:

• одному запису у батьківській таблиці відповідає більше за один запис в дочірній таблиці ;

• одному запису в дочірній таблиці відповідає більше за один запис у батьківській таблиці.

Вважається, що всякий зв'язок "багато-до-багатьох" може бути замінений на зв'язок "один-до-багатьох" (один або декілька).

Часто зв'язок між таблицями встановлюється по первинному ключу, тобто значення зовнішнього ключа однієї таблиці привласнюються значенню первісного ключа іншої. Проте це не є обов'язковим - в загальному випадку зв'язок може встановлюватися і за допомогою вторинних ключів. Крім того, при встановленні зв'язків між таблицями не потрібно неодмінну унікальність ключа, що забезпечує встановлення зв'язку. Поля зовнішнього ключа не зобов'язані мати тих же імен, що і імена ключів, яким вони відповідають. Зовнішній ключ може посилатися на свою власну таблицю - у такому разі зовнішній ключ називається рекурсивним.

Посилальна цілісність визначає: якщо в таблиці існує зовнішній ключ, то його значення повинне або відповідати значенню первісного ключа деякого запису у базовій таблиці, або задаватися визначником NULL.

Існує декілька важливих моментів, пов'язаних із зовнішніми ключами. По-перше, слід проаналізувати, чи допустиме використання в зовнішніх ключах порожніх значень. У загальному випадку, якщо участь дочірньої таблиці в зв'язку є обов'язковою, то рекомендується забороняти застосування порожніх значень у відповідному зовнішньому ключі. В той же час, якщо має місце часткова участь дочірньої таблиці в зв'язку, то приміщення порожніх значень в поле зовнішнього ключа має бути дозволене. Наприклад, якщо в операції фіксації угод деякої торгової фірми необхідно вказати покупця, то поле КодКлієнта повинне мати атрибут NOT NULL. Якщо допускається продаж або купівля товару без вказівки клієнта, то для поля КодКлієнта можна вказати атрибут NULL.

Наступна проблема пов'язана з організацією підтримки посилальної цілісності при виконанні операцій модифікації даних у базі. Тут можливі наступні ситуації:

1. Вставка нового рядка в дочірню таблицю. Для забезпечення посилальної цілісності необхідно переконатися, що значення зовнішнього ключа нового рядка дочірньої таблиці рівно порожньому значенню або деякому конкретному значенню, присутньому в поле первісного ключа одного з рядків батьківської таблиці.

2. Видалення рядка з дочірньої таблиці. Ніяких порушень посилальної цілісності не відбувається.

3. Оновлення зовнішнього ключа в рядку дочірньої таблиці. Цей випадок подібний до описаної вище першої ситуації. Для

збереження цілісності посилання необхідно переконатися, що значення зовнішнього ключа в оновленому рядку дочірньої таблиці рівно порожньому значенню або деякому конкретному значенню, присутньому в полі первинного ключа одного з рядків батьківської таблиці.

4. Вставка рядка у батьківську таблицю. Така вставка не може викликати порушення посилальної цілісності. Доданий рядок просто стає батьківським об'єктом, що не має дочірніх об'єктів.

5. Видалення рядка з батьківської таблиці. Посилальна цілісність виявиться порушеною, якщо в дочірній таблиці існуватимуть рядки, що посилаються на видалений рядок батьківської таблиці. В цьому випадку може використовуватися одна з наступних стратегій :

- **NO ACTION**. Видалення рядка з батьківської таблиці забороняється, якщо в дочірній таблиці існує хоч би один рядок, що посилається на неї.
- **CASCADE**. При видаленні рядка з батьківської таблиці автоматично видаляються усі рядки дочірньої таблиці, що посилаються на неї. Якщо будь-який з рядків дочірньої таблиці, що видаляються, виступає батьківською стороною в якому-небудь іншому зв'язку, то операція видалення застосовується до усіх рядків дочірньої таблиці цього зв'язку і так далі. Іншими словами, видалення рядка батьківської таблиці автоматично поширюється на будь-які дочірні таблиці.
- **SET NULL**. При видаленні рядка з батьківської таблиці в усіх рядках дочірнього відношення, що посилаються на неї, в поле зовнішнього ключа, що відповідає первинному ключу видаленого рядка, записується порожнє значення. Отже, видалення рядків з батьківської таблиці викличе занесення порожнього значення у відповідне поле дочірньої таблиці. Ця стратегія може використовуватися, тільки коли в полі зовнішнього ключа дочірньої таблиці дозволяється поміщати порожні значення.
- **SET DEFAULT**. При видаленні рядка з батьківської таблиці в поле зовнішнього ключа усіх рядків дочірньої таблиці, що посилаються на неї, автоматично поміщається значення, вказане для цього поля як значення за умовчанням. Таким чином, видалення рядка з батьківської таблиці викликає приміщення значення, що набуває за умовчанням, в поле зовнішнього ключа усіх рядків дочірньої таблиці, що посилаються на видалений рядок. Ця стратегія застосовна лише в тих випадках, коли полю зовнішнього ключа дочірньої таблиці призначено деяке значення, що приймається за умовчанням.
- **NO CHECK**. При видаленні рядка з батьківської таблиці ніяких дій зі збереження посилальної цілісності даних не робиться.

6. Оновлення первинного ключа в рядку батьківської таблиці. Якщо значення первинного ключа деякого рядка батьківської таблиці буде оновлено, порушення посилальної цілісності станеться за тієї умови, що в дочірньому відношенні існують рядки, що посилаються на початкове

значення первинного ключа. Для збереження посилальної цілісності може застосовуватися будь-яка з описаних вище стратегій. При використанні стратегії CASCADE оновлення значення первинного ключа в рядку батьківської таблиці буде відображене у будь-якому рядку дочірньої таблиці, що посилається на цей рядок.

2. ПРАКТИЧНІ ЗАВДАННЯ

```
drop table if exists students;
CREATE TABLE students (
   id INT AUTO_INCREMENT PRIMARY KEY,
   first_name VARCHAR(50) NOT NULL,
   last_name VARCHAR(50) NOT NULL,
   average_grade DECIMAL(4,2),
   birth_date DATE
);
INSERT INTO students (first_name, last_name, average_grade, birth_date)
VALUES
('Олександр', 'Коваленко', 87.75, '2003-11-22'),
('Ірина', 'Гнатюк', 95.20, '2005-01-09'),
('Владислав', 'Петренко', 89.10, '2004-09-30'),
('Олена', 'Мельник', 91.80, '2003-06-12');
```

Додаткові функції

Об’єднання стрічок: **CONCAT**(str1 string, str2 string,...)	1. Об’єднати прізвище та ім’я та вивести під псевдонімом full_name
	2. Додати пропуск між прізвищем та ім'ям
LEFT(str, n), яка повертає n кількість символів з початку рядка,	3. Підправити, щоб при об'єднанні в імені взяти тільки першу букву і після цього поставити крапку
Зміна регістрів символів - **LOWER, UPPER**	4. Підправити: прізвище вивести великими буквами
CONCAT_WS(separator, str1, str2, str3, ...)	5. Об’єднати прізвище та повне ім’я з CONCAT_WS. Використати пропуск як сепаратор між прізвищем та ім'ям
ROUND(x, d)Округлює до d знаків (звичайне математичне округлення) Округлення до цілого: ROUND(x)	6. Округлити average_grade до цілої частиини
TIMESTAMPDIFF(unit,	7. Порахувати скільки років кожному

datetime1, datetime2) — повертає різницю між двома датами у вибраному unit (тут YEAR). CURDATE() — поточна дата	студенту

Цілісність даних

9. Задано таблиці:

```
DROP TABLE IF EXISTS Teachers;
DROP TABLE IF EXISTS Departments;
DROP TABLE IF EXISTS Faculties;

CREATE TABLE Faculties(id INT, fac_name VARCHAR(45) NOT NULL,
dekan INT);
  INSERT INTO Faculties VALUES (1,'Faculty of Mathematics and Computer
Science',22);
  INSERT INTO Faculties VALUES (2,'Faculty of Pedagogy',24);
  INSERT INTO Faculties VALUES (3,'Faculty of Psychology',25);

CREATE TABLE Departments (id INT, dep_name VARCHAR(100) NOT NULL,
facult INT, zav INT);
  INSERT INTO Departments VALUES (11,'Department of Computer Science and
Information Systems',1,23);
  INSERT INTO Departments VALUES (12,'Department of Algebra and
Geometry',1,29);
  INSERT INTO Departments VALUES (13,'Department of Primary Education
Pedagogy',2,25);
  INSERT INTO Departments VALUES (14,'Department of Philosophy, Sociology,
and Religious Studies',3,28);

CREATE TABLE Teachers (id INT, teacher_name VARCHAR(100) NOT NULL,
depat INT);
  INSERT INTO Teachers VALUES (21,'Sharyn',12);
  INSERT INTO Teachers VALUES (22,'Pylypiv',12);
  INSERT INTO Teachers VALUES (23,'Petryshyn',11);
  INSERT INTO Teachers VALUES (24,'Vynnychuk',14);
  INSERT INTO Teachers VALUES (25,'Ivanenko',13);
  INSERT INTO Teachers VALUES (26,'Ilyash',11);
  INSERT INTO Teachers VALUES (27,'Solomko',12);
  INSERT INTO Teachers VALUES (28,'Kozlenko',14);
  INSERT INTO Teachers VALUES (29,'Nykyforchyn',12);
```

Підправити існуючий код для коректної роботи

10. Для таблиць задати первинні ключі
11. Задати для всіх первинних ключів AUTO_INCREMENT

12. Оскільки імена **первинних ключів співпадають**, тому задайте їм унікальні імена, щоб **не було конфліктів** у JOIN або VIEW (**fac_id, dep_id, teacher_id** для відповідних таблиць)

13. Задати запит об'єднавши таблиці **Faculties та Departments**
14. Для кожного факультету порахувати кількість кафедр (використовуючи попередній запит)
15. Об'єднати всі три таблиці
16. Створити view на основі запиту об'єднання трьох таблиць та відібравши тільки поля з назвами: fac_name, dep_name, teacher_name

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name;
```

17. Вивести view
18. Порахувати скільки він містить рядків
19. Використати view для підрахунку кількості працівників на кожному факультету

20. Видалити через команду з таблиці Departments рядок з кафедрою алгебри.
21. Вивести таблицю Departments та view. Скільки тепер у view рядків
22. Перевірити хто з викладачів тепер без кафедри (у view додати **right** перед join Teachers)
23. Вивести таблицю Teachers

Зауваження: у таблиці Teachers залишились працівники кафедри алгебри, для яких вказано dep_id кафедри 12, тобто не забезпечується цілісність даних (зміни в одній таблиці не впливають на інші таблиці)

24. **Закоментувати** запит на знищення кафедри
25. Зв'яжемо таблиці **Departments та Teachers:**, задати зовнішній ключ у таблиці Teachers, тобто вказуємо, що зміни які відбуваються з первинним ключем Departments(dep_id) впливають на поле depat:

```
foreign key (depat) references Departments(dep_id));
```

26. **Розкоментувати** запит на знищення кафедри алгебри. Зовнійшній ключ не дасть знищити пов'язані дані: Cannot delete or update a parent row: a foreign key constraint fails

27. Додамо конкретні дії при знищенні рядка з первинним ключем в таблиці **Departments** у випадку існування зовнішнього ключа:

У таблиці Teachers підправимо команду зовнішнього ключа, дописавши **on delete задавши один з параметрів CASCADE або SET NULL або NO ACTION**

1 варіант: При знищення кафедри нехай всі працівники з цієї кафедри теж будуть “знищенні” - звільнені

```
depat INT, foreign key (depat) references Departments(dep_id) on delete cascade);
```

Перевірити чи при відображенні таблиці **Teachers** працівники кафедри алгебри відсутні

2 варіант: При знищення кафедри нехай всім працівники з цієї кафедри забрати номер кафедри та поставити NULL

```
references Departments(dep_id) on delete set NULL
```

Перевірити чи при відображенні таблиці **Teachers** працівникам кафедри алгебри поставлено NULL

Проведемо аналогічні дії з таблицею **Departments:**

28. Зв’яжемо таблиці **Faculties та Departments**: задати зовнішній ключ у таблиці Departments:

CREATE TABLE Departments (dep_id INT …, zav INT,

```
foreign key (facult) references Faculties(fac_id));
```

Зміни з **Faculties** (fac_id) будуть впливати на **Departments(**facult**)**

29. Дописати on delete set NULL до зовнішнього ключа
30. Знищити факультет Psychology

31. Закоментувати запити на знищення, переходимо до **оновлення даних**
32. Змінити значення первинного ключа для кафедри 'Department of Primary Education Pedagogy' з 13 на 15
33. Щоб команда виконалась підправити команди зовнішніх ключів, додавши **on update cascade** (можна зразу в обох)

```
on delete set NULL on update cascade
```

34. Перевірити дані для 'Ivanenko'
35. Вивести таблицю Teachers
36. Вивести view, що об’єднював всі три таблиці

ЛАБОРАТОРНА РОБОТА №12
ПРЕДСТАВЛЕННЯ (ПОДАННЯ). ОПЕРАТОРИ ANY, ALL У ВКЛАДЕНИХ ЗАПИТАХ

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Представлення (Подання).

Базові таблиці (TABLE): фізичні об'єкти БД, які містять дані і зберігаються в пам'яті комп'ютера на жорсткому диску**. Запит на вибірку даних** до базових таблиць є тимчасова результатна таблиця, доступна лише тому, хто виконав запит.

Віртуальні таблиці (VIEW): не є фізичними об'єктами зберігання даних, вони дозволяють повертати певні поля таблиць у вигляді зв'язаного набору даних, які можуть бути доступні багатьом користувачам та існують в базі даних до тих пір, поки не будуть спеціально видалені

Подання, представлення(VIEW) – об'єкт бази даних, що є результатом виконання запиту до бази даних, визначеного за допомогою оператора SELECT, в момент звернення до подання.

Представлення іноді називають "віртуальними таблицями". Така назва пов'язана з тим, що подання доступно для користувача як таблиця, але саме воно не містить даних, а витягує їх з таблиць в момент звертання до нього. Подання може містити дані з декількох з'єднаних таблиць.

Подання називається оновлюваним (модифіцированим), якщо до нього можуть бути застосовні оператори UPDATE і DELETE для зміни даних в таблицях, на яких засновано уявлення.

https://www.slideshare.net/mehalyna/sql-view

Практичні завдання:

```
CREATE TABLE teacher( id integer,  kafedra    char(20),
                      name char(20), chief_id integer,
                      staz integer);
INSERT INTO teacher VALUES (1,'matem','Pylypiv', 0, 27);
INSERT INTO teacher VALUES (2,'matem','Nykyforchyn', 1,18);
INSERT INTO teacher VALUES (3,'inform','Petryshyn',1,21);
INSERT INTO teacher VALUES (4,'inform','Ilyash',3,10);
INSERT INTO teacher VALUES (5,'inform','Malko',3,24);
INSERT INTO teacher VALUES (6,'matem','Zatorskuj',1,34);
INSERT INTO teacher VALUES (7,'matem','Solomko',1,12);

INSERT INTO teacher VALUES (8,'inform','Rovinsk',3,18);
```

```
INSERT INTO teacher VALUES (9,'inform','Gorelov',3,15);
```

1. Створити представлення komp з викладачами кафедри інформатики. Стовпчик з назвою кафедри вже не буде потрібен
2. Змінити дані в основній таблиці через update, перевірити чи змінюється дані після цього в komp
3. Створити представлення, що для кожної кафедри обчислює середній стаж
4. Знищити представлення

Оператори Any, All у вкладених запитах

> ANY, > = ANY істинні коли значення більше або більше-рівне довільного значення внутрішнього запиту.

< ANY, < = ANY істинні коли значення менше або менше -рівне довільного значення внутрішнього запиту

= ALL істинний, коли значення рівне всім значенням внутрішнього запиту. Така ситуація може спостерігатися, коли внутрішній запит повертає лише одне значення.

<> ALL істинний, коли значення не рівне жодному значенню внутрішнього запиту.

> ALL, > = ALL істинні, коли значення більші або більші-рівні всіх значень внутрішнього запиту

< ALL, < = ALL істинні, коли значення менші або менші-рівні всіх значень внутрішнього запиту

Слід враховувати, що коли результат внутрішнього запиту пустий, то оператори (=, >, <) ALL приймають істинне значення, а аналогічні оператори із ANY – хибне.

Нехай потрібно вибрати дані про студентів, які проживають в містах де розташований університет, в якому вони навчаються

```
SELECT  *
  FROM student s
  WHERE city IN
   (SELECT city
     FROM university u
       WHERE u.univ_id = s.univ_id);
```

Запит, що вибирає назви університетів з рейтингом вищим, ніж рейтинг довільного університету в Тернополі

```
SELECT *
  FROM university
  WHERE rating > ALL
 ( SELECT rating
   FROM university
```

WHERE city = ‘Тернопіль’);

2. **ПРАКТИЧНІ ЗАВДАННЯ**

1. Вивести дані про менеджера з найбільшою зарплатою
2. Вивести дані про працівників, які працюють у відділі де є менеджер
3. **В кожному відділі** вивести дані про працівника, що має найбільшу зарплату

з оператором ALL	**без оператора ALL (в Sqlite)**
select a.name, a.salary, a.dept from employees a where a.salary>=**All**(select b.salary from employees b where a.dept=b.dept);	select a.name, a.salary, a.dept from employees a where a.salary>=(select **max**(b.salary) from employees b where a.dept=b.dept);

Використання реляційної алгебри

	1. Вивести список без повторень номерів всіх відділів з першої таблиці
	2. Вивести номер відділу, де є працівники на посаді ADMIN
	3. Вивести номер відділу, де нема ADMIN (зі списку всіх відділі з 5 запиту відкинути відділі де є ADMIN з 6 запиту)
SQLite, Postgresql **except**	**MySql** SELECT File_Name FROM Words_DB WHERE File_Name **NOT IN** (SELECT File_Name FROM Files_DB)
	4. Яких посад нема у 4 відділі
Особливості MySQL	Якщо у підзапиті є **NULL**, тоді NOT IN може **повернути порожній результат**, бо MySQL не знає, чи порівнювати id з NULL (невизначене значення). Безпечніший варіант — використати NOT EXISTS:
На основі двох таблиць	SELECT c.customer_id, c.name FROM Customers c WHERE NOT EXISTS (SELECT o.customer_id

	FROM Orders o WHERE o.customer_id = c.customer_id);
	5. Вивести співробітників, які не є менеджерами для інших співробітників. Вивести всі номери, що задані в стовпці manager (він буде підзапитом) Задати в основному запиті виведеннч id з першої “копії” таблиці **from employees a** У підзапиті використати позначення from employees b та умову **where a.id=b.manager**
	6. Оновити інформацію про 'GRANT' і 'TAFT', задати dept для них 4 7. Дані про працівника ROOSEVELT видалити
	8. Вивести відділи, де є всі посади, о згадуються в таблиці А) вивести всі посади з таблиці без повторень Б) вивести кількість посад без повторень В) для кожного відділу порахувати кількість посад Г) через фільтрацію в групах задати умову COUNT(кількість посад без повторень) = (підзапит, що рахує кількість посад без повторень у всіх таблиці);

ЛАБОРАТОРНА РОБОТА №13

ПРАКТИЧНІ ЗАВДАННЯ

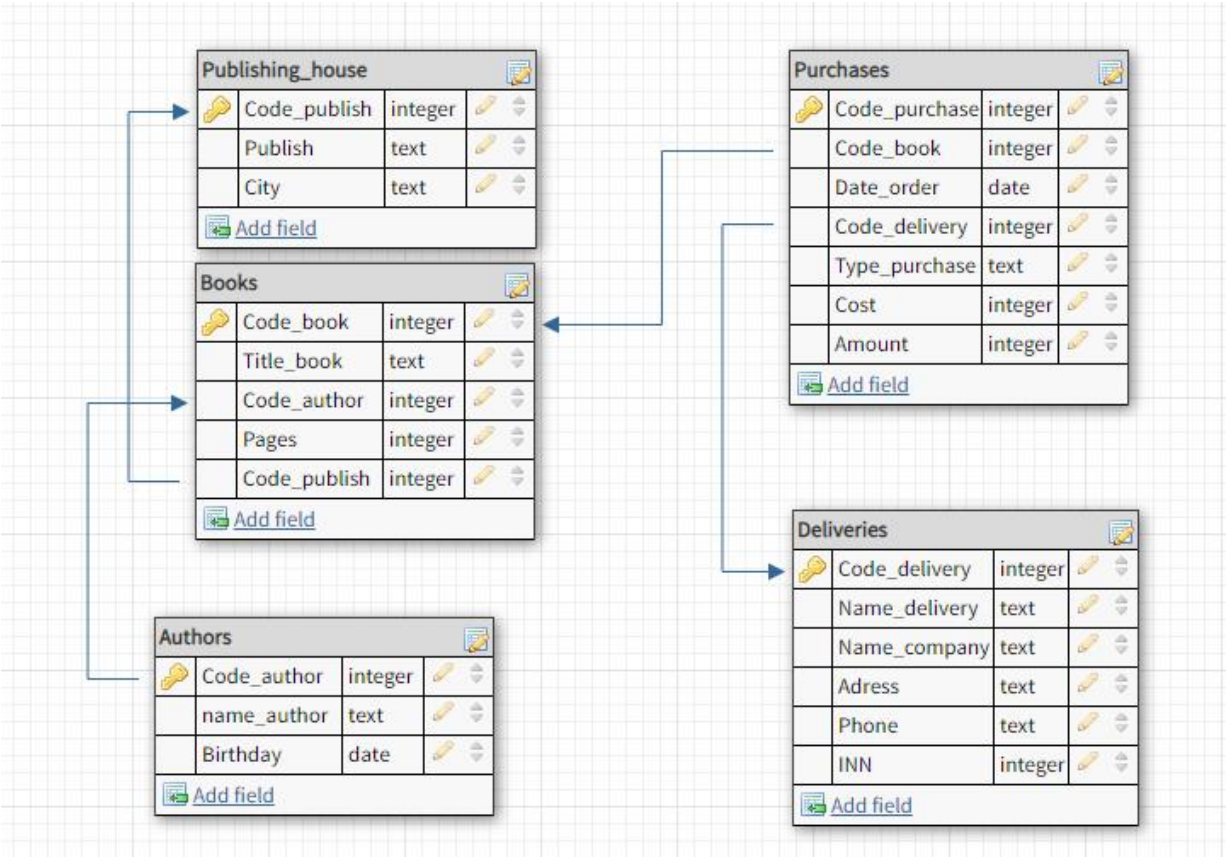
<table>
<tr><td>1. Створити таблицю</td><td>Подано маршрут 26 автобусу від Сільпо до вокзалу
drop table if exists bus_26;
CREATE TABLE bus_26(id integer primary key,
name char(20), timefloat);
INSERT INTO bus_26 VALUES (1,'Silpo',0);
INSERT INTO bus_26 VALUES (2,'Dovzenko',6.3);
INSERT INTO bus_26 VALUES (3,'Zaravshan',3.7);
INSERT INTO bus_26 VALUES (4,'Kolco',5);
INSERT INTO bus_26 VALUES (5,'Park',4);
INSERT INTO bus_26 VALUES (6,'Universytet',2.4);
INSERT INTO bus_26 VALUES (7,'Chornovola',3.2);
INSERT INTO bus_26 VALUES (8,'Poshta',5.2);
INSERT INTO bus_26 VALUES (9,'Vokzal',4);</td></tr>
<tr><td colspan="2">2. Які зупинки перед Парком
3. Яка остання зупинка перед Парком</td></tr>
<tr><td colspan="2">4. Скільки зупинок треба проїхати щоб добратись від зупинки Парк до Пошти
5. Скільки часу триватиме поїздка від зупинки Університет до Вокзалу</td></tr>
<tr><td colspan="2">6. Вивести назви двох зупинок: одну перед зупинкою Парк і ту, що є наступною після зупинки Парк</td></tr>
<tr><td colspan="2">7. Для яких зупинок тривалість однакова
<table><tr><th>name</th><th>name</th><th>time</th></tr><tr><td>Vokzal</td><td>Park</td><td>4</td></tr></table>
8. Вивести дві назви зупинок між якими найдовше їхати</td></tr>
<tr><td>Створити нову таблицю</td><td>DROP TABLE IF EXISTS bus;
CREATE TABLE bus(num_bus integer, id integer,
name char(20), timeinteger);

INSERT INTO bus VALUES (26, 1,'Silpo',0);
INSERT INTO bus VALUES (26, 2,'Dovzenko',5);
INSERT INTO bus VALUES (26, 3,'Zaravshan',4);
INSERT INTO bus VALUES (26, 4,'Kolco',3);
INSERT INTO bus VALUES (26, 5,'Park',3);
INSERT INTO bus VALUES (26, 6,'Universytet',2);
INSERT INTO bus VALUES (26, 7,'Chornovola',4);
INSERT INTO bus VALUES (26, 8,'Poshta',5);
INSERT INTO bus VALUES (26, 9,'Vokzal',7);
INSERT INTO bus VALUES (25, 1,'Konovalca',0);
INSERT INTO bus VALUES (25, 2,'Likarnya',4);</td></tr>
</table>

	INSERT INTO bus VALUES (25, 3,'Universytet',4); INSERT INTO bus VALUES (25, 4,'Poshta',4); INSERT INTO bus VALUES (25, 5,'Bazar',6); INSERT INTO bus VALUES (25, 6,'Prykarpattya',4); INSERT INTO bus VALUES (22, 1,'Chotkevucha',0); INSERT INTO bus VALUES (22, 2,'Vokzal',5); INSERT INTO bus VALUES (22, 3,'Nadia',6); INSERT INTO bus VALUES (22, 4,'Prykarpattya',10);
4. Які автобуси зупиняються на зупинці «університет»	
5. Скільки зупинок на кожному рейсі автобуса	
6. Тривалість повної поїздки для 25 автобуса	
7. Вивести найбільші тривалості часу між зупинками на кожному маршруті	
8. на який маршрутах кількість зупинок більша 7 9. на якому маршруті тривалість повного рейсу більше 30	
10. на кожному рейсі вивести середню тривалість часу між зупинками	
11. на рейсі 22 знайти найкоротший інтервал між зупинками (крім нуля)	
select _____ from bus a, bus b where __	12. Яким автобусом можна доїхати від університету до вокзалу без пересадки 13. Яким можна доїхати від Хоткевича до Надії

Підготовка до контрольної роботи

Виконати запити над заданими таблицями



```
DROP TABLE IF EXISTS Publishing_house;
DROP TABLE IF EXISTS Books;
DROP TABLE IF EXISTS Purchases;
DROP TABLE IF EXISTS Authors;
DROP TABLE IF EXISTS Deliveries;

CREATE TABLE Publishing_house (
	Code_publish integer PRIMARY KEY AUTOINCREMENT,
	Publish text,
	City text
);
insert into Publishing_house values (1, 'Machaon', 'Kiev');
insert into Publishing_house values (2, 'Lev', 'Lviv');

CREATE TABLE Books (
	Code_book integer PRIMARY KEY AUTOINCREMENT,
	Title_book text,
	Code_author integer,
	Pages integer,
	Code_publish integer
);
insert into Books values (1,'Kod da vinchi', 2,345,2);
insert into Books values (2,'Bashnja', 1,652,1);
insert into Books values (3,'Angel', 2,444,1);
insert into Books values (4,'Demon', 2,234,1);

CREATE TABLE Purchases (
	Code_purchase integer PRIMARY KEY AUTOINCREMENT,
	Code_book integer,
	Date_order date,
	Code_delivery integer,
	Type_purchase text,
	Cost integer,
	Amount integer
);
insert into Purchases values (1,3,'2019-01-03',1, 'opt',34,10);
insert into Purchases values(2,2,'2019-01-18',2, 'rozrib',4,45);
insert into Purchases values(3,2,'2019-02-11',1, 'opt',67,23);
insert into Purchases values(4,1,'2019-02-24',2, 'opt',345,2);
insert into Purchases values(5,3,'2019-03-01',1, 'opt',45,56);

CREATE TABLE Authors (
	Code_author integer PRIMARY KEY AUTOINCREMENT,
	name_author text,
```

```
    Birthday date
);
insert into Authors values (1,'King','1975-10-04');
insert into Authors values (2,'Den Draun','1988-04-11');

CREATE TABLE Deliveries (
    Code_delivery integer PRIMARY KEY AUTOINCREMENT,
    Name_delivery text,
    Name_company text,
    Adress text,
    Phone text,
    INN integer
);
insert into Deliveries values (1,'Vat Inger', 'Ivodent','Ternopil,Bandera 34','2-45-
56',3453463463);
insert into Deliveries values (2,'TNZO', 'Krak','Odesa, Lenina 124','44-7-
20',45626262);
insert into Deliveries values (3,'fovat', 'Mameno','Ivano-Frankivsc,
Chornovola','55-78-90',24654745);
```

Завдання для виконання

Багатотабличні запити (вибірка з двох таблиць, вибірка з трьох таблиць з використанням JOIN)

1. Вивести список назв компаній-постачальників (поле Name_company) і назви книг (поле Title_book), які вони постачали в період з 01.01.2016 по 31.12.2016 (умова по полю Date_order).
2. Вивести список авторів (поле Name_author), книги яких були випущені у видавництві “Мир” (умова по полю Publish).
3. Вивести список постачальників (поле Name_company), які постачають книги видавництва “Махаон” (умова по полю Publish).
4. Вивести список авторів (поле Name_author) і назви книг (поле Title_book), які були поставлені постачальником “ВАТ Книготорг” (умова по полю Name_company).

Обчислення

5. Вивести сумарну вартість партії однойменних книг (використовуючи поля Amount і Cost) і назву книги (поле Title_book) в кожній поставці.
6. Вивести вартість однієї друкованої сторінки кожної книги (використовувати поля Cost і Pages) і назви відповідних книг (поле Title_book).

7. Вивести кількість років з моменту народження авторів (використовувати поле Birthday) і імена відповідних авторів (поле Name_author).

Обчислення підсумкових значень з використанням агрегатних функцій

8. Вивести загальну суму поставок книг (використовувати поле Cost), виконаних “ЗАТ Оптторг” (умова по полю Name_company).
9. Вивести загальну кількість всіх поставок (використовувати будь-яке поле з таблиці Purchases), виконаних в період з 01.01.2016 по 01.02.2016 (умова по полю Date_order).
10. Вивести середню вартість (використати поле Cost) і середню кількість екземплярів книг (використати поле Amount) в одній поставці, де автором книги є “Олесь Гончар” (поле Name_author).
11. Вивести всі відомості про постачання (всі поля таблиці Purchases), а також назву книги (поле Title_book) з мінімальною загальною вартістю (використовувати поля Cost і Amount).
12. Вивести всі відомості про постачання (всі поля таблиці Purchases), а також назву книги (поле Title_book) з максимальною загальною вартістю (використовувати поля Cost і Amount).

Зміна найменувань полів

13. Вивести назву книги (поле Title_book), сумарну вартість партії однойменних книг (використовувати поля Amount і Cost), помістивши результат в поле з назвою Itogo, поставки за період з 01.01.2016 по 01.06.2016 (умова по полю Date_order).
14. Вивести вартість однієї друкованої сторінки кожної книги (використати поля Cost і Pages), помістивши результат в поле з назвою One_page, і назви відповідних книг (поле Title_book).
15. Вивести загальну суму поставок книг (використовувати поле Cost) і помістити результат в поле з назвою Sum_cost, виконаних “ВАТ Луч” (умова по полю Name_company).

Використання змінних в умові

16. Вивести список угод (всі поля з таблиці Purchases) за останній місяць (умова з використанням поля Date_order).
17. Вивести список авторів (поле Name_author), вік яких менше заданого користувачем (умова з використанням поля Birthday).
18. Вивести список книг (поле Title_book), яких закуплено менше, ніж зазначено в запиті користувача (умова з використанням поля Amount).

Використання змінних замість назв таблиць

19. Вивести список назв компаній-постачальників (поле Name_company) і назви книг (поле Title_book), які вони поставили.
20. Вивести список авторів (поле Name_author), книги яких були випущені у видавництвах “СВІТ”, “НАУКА” (умова по полю Publish).
21. Вивести список видавництв (поле Name_company), книги, що були продані за ціною 350 грн. (Поле Cost).

Використання функцій спільно з підзапитом

22. Вивести список книг (поле Title_book), у яких кількість сторінок (поле Pages) більше середньої кількості сторінок усіх книг в таблиці.
23. Вивести список авторів (поле Name_author), вік яких менше середнього віку всіх авторів в таблиці (умова по полю Birthday).
24. Вивести список книг (поле Title_book), у яких кількість сторінок (поле Pages) дорівнює мінімальній кількості сторінок книг, представлених в таблиці.

Використання квантора існування в запитах

25. Вивести список видавництв (поле Publish), книги яких були придбані оптом (“опт” з поля Type_Purchase).
26. Вивести список авторів (поле Name_author), книг яких немає в таблиці Books.
27. Вивести список книг (поле Title_book), які були поставлені постачальником “ЗАТ Квантор” (умова по полю Name_company).

Оператор обробки даних Update

28. Змінити в таблиці Books вміст поля Pages на 300, якщо код автора рівний 56 (поле Code_author) і назва книги (поле Title_book) – “Мемуари”.
29. Змінити в таблиці Deliveries вміст поля Address на “немає відомостей”, якщо значення поля є порожнім.
30. Збільшити в таблиці Purchases ціну (поле Cost) на 20 відсотків, якщо замовлення було оформлено протягом останнього місяця (умова по полю Date_order).

Оператор обробки даних Insert

31. Додати в таблицю Purchases новий запис, причому так, щоб код покупки (поле Code_purchase) було автоматично збільшено на одиницю, а в тип закупівлі (поле Type_purchase) внести значення “опт”.
32. Додати в таблицю Books новий запис, причому замість ключового поля поставити код (поле Code_book), автоматично збільшений на одиницю

від максимального коду в таблиці, замість назви книги (поле Title_book) написати “Наука. Техніка. Інновації”.

33. Додати в таблицю Publish_house новий запис, причому замість ключового поля поставити код (поле Code_publish), автоматично збільшений на одиницю від максимального коду в таблиці, замість назви міста – “Київ” (поле City), замість видавництва – “Наука” (поле Publish).

Оператор обробки даних Delete

34. Видалити з таблиці Purchases всі записи, у яких кількість книг в замовленні (поле Amount) = 0.
35. Видалити з таблиці Authors всі записи, у яких немає імені автора в полі Name_Author.
36. Видалити з таблиці Deliveries всі записи, у яких не зазначено ІПН (поле INN порожнє).

Проектування реляційної бази даних

1 ЕТАП

На першому етапі проектування бази даних необхідно **визначити мету** створення бази даних, **основні її функції** та **дані**, яку вона повинна містити. Після чого слід починати опис предметної області. Такий опис повинен охоплювати весь клас сутностей предметної області (реальні об'єкти, процеси і явища), інформація про яких повинна міститися в БД і забезпечувати реалізацію можливих запитів до БД і рішення задач. Проектування передбачає етапи створення проекту бази даних від концепції до реального втілення.

Етапи проектування бази даних:

- дослідження предметної області;
- аналіз даних;
- визначення відносин між елементами даних. Визначення первинних і вторинних (зовнішніх) ключів відносин. Реальні наочні області є системами взаємозв'язаних об'єктів, наприклад облік товарів, які реалізовані різним фірмам з декількох складів. Проектуючи таку базу даних, користувач повинен передбачити, яка інформація йому може знадобитися в майбутньому.

Етапи розробки бази даних:

1. **Інфологічна модель**
2. **Даталогічна модель**
3. **Фізична модель**

Інфологічна модель – відображає інформацію про предметну область у вигляді незалежному від СУБД, що використовується. Ця модель відображає інформаційно-логічний рівень абстрагування, який пов'язаний з описом об'єктів предметної області, їх властивостей і взаємозв'язків. Часто ці моделі ототожнюють з концептуальними моделями предметної області і називають концептуальними інфологічними моделями (внутрішня і зовнішня концептуальні інфологічні моделі).

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

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

Модель "сутність-зв'язок" (ER-модель) – описує модель предметної області і складається з множини сутностей, множини звязків між сутностями, а також з атрибутів сутностей і зв'язків.

Представлення даних на різних фазах проектування

Концептуальний рівень (інфологічна модель)

• Сутності	Представлення аналітика
• Атрибути	
• Зв’язки	
Логічний рівень (даталогічна модель)	
• Записи	Представлення програміста
• Елементи даних	
• Зв’язки між записами	
Фізичний рівень (фізична модель)	
• Групування	Представлення адміністратора
• Індекси	
• Методи доступу	

Розглянемо більш докладно **інфологічний** етап проектування.

В предметній області існують *сутності*, інформацію про які необхідно зберігати, і ці сутності пов'язані одна з одною певними зв’язками.

Перед початком проектування бази даних необхідно визначити інформаційні об’єкти (сутності) і взаємозв’язки, що існують між ними.

Взаємовідношення між даними не залежать від моделі даних і СУБД, яка застосовується. А модель даних – навпаки — визначається системою управління базою даних.

Сутності і їх атрибути. **Сутність** (entity) — це щось, про що зберігається інформація. Клієнт — це сутність, як і товар, що зберігається на складі. Зовсім не обов'язково, щоб сутності були матеріальні. Так, деяка подія, наприклад концерт, є сутність; звернення до лікаря — теж сутність.

Сутність описують даними, званими **атрибутами** (attributes). Якщо сутність є економічним об’єктом, то атрибутами є його реквізити. Наприклад, клієнт як *сутність* звичайно описується номером, ім'ям, прізвищем, вулицею, містом, країною, поштовим індексом і номером телефону. *сутність* концертів можна описати назвою, датою, місцем проведення і ім'ям виконавця.

При представленні сутності в базі даних фактично зберігаються тільки атрибути. Кожна група атрибутів, що описують один реальний прояв сутності, є **екземпляром** сутність.

Ідентифікатори сутностей. Єдиною метою розміщення в базі даних інформації, що описує сутність, є подальше зчитування цієї інформації. Отже, необхідно мати певний метод, що дозволяє відрізняти одну сутність від іншої і тим самим забезпечувати зчитування потрібної сутності. Для цього кожній сутності привласнюються певні значення атрибутів, що відрізняють її від будь–якої іншої сутності бази даних; сукупність значень називається **ідентифікатором** сутності (entity identifier).

Припустимо, що у фірми є два клієнти з ім'ям Петро Сидоров. Якщо службовець шукає пункти замовлення Петра Сидорова, відомості про якого з Петрів Сидорових вибере СУБД? В даному випадку — про обох. Оскільки не існує способу розрізнити двох клієнтів, результати запиту будуть неточні.

Ця проблема може бути розв'язана створенням унікальних кодів (номерів) клієнтів. Це вельми поширений спосіб ідентифікації екземплярів сутностей, коли у даних не передбачений який–небудь простий унікальний ідентифікатор.

Іншим рішенням може стати об'єднання імені і прізвища клієнта з його телефонним номером. Подібна комбінація (складений ідентифікатор) також однозначно визначає кожного клієнта. Проте тут є два недоліки. По–перше, такий ідентифікатор довгий і громіздкий, і при введенні будь–якого з його компонентів велика вірогідність помилки. По–друге, при зміні номера телефону клієнта ідентифікатор теж повинен змінитися. Зміна ідентифікатора сутності може привести до виникнення серйозних проблем в базі даних.

У деяких сутностей, наприклад у *рахунків фактур*, є природні ідентифікатори (номери рахунків–фактур). Іншим сутностям — головним чином людям, населеним пунктам і предметам — привласнюються унікальні номери без певного значення. Для третіх необхідні складені ідентифікатори.

При збереженні екземпляра сутності в базі даних потрібно, щоб СУБД забезпечувала наявність у нового екземпляра унікального ідентифікатора. Це є прикладом обмеження в базі даних — правила, якому повинні відповідати дані. Реалізація різних обмежень допомагає підтримувати узгодженість і точність інформації.

*Однозначні і багатозначні атрибути.*У даному випадку кінцевим висновком процесу проектування є створення реляційної бази даних, тому атрибути нашої моделі даних повинні бути однозначними. Іншими словами, у кожного атрибуту конкретного екземпляра сутності може бути тільки одне значення. Наприклад, сутність Клієнт допускає наявність тільки одного телефонного номера для кожного клієнта. Якщо у клієнта декілька телефонних номерів, які потрібно включити в базу даних, то сутність Клієнт не зможе вміщати їх всі.

Хоча вірним є те, що модель взаємовідношення сутностей в бази даних не залежить від формальної моделі даних, що використовується для відображення структури даних в СУБД, часто рішення про моделювання даних ухвалюються виходячи з вимог тієї формальної моделі, яка застосовуватиметься згодом. Одне з подібних рішень — видалення багатозначних атрибутів. Аналогічний приклад буде приведений при розгляді взаємостосунків "багато-до багатьох" між сутностями.

Наявність декількох телефонних номерів перетворює атрибут телефонного номера в багатозначний. Оскільки в реляційній базі даних

сутність не може мати багатозначні атрибути, необхідно упорядкувати їх, створивши сутність для їх зберігання.

У даній ситуації можна створити сутність телефонних номерів. В кожен екземпляр цієї сутності потрібно включити номер клієнта, якому належить телефонний номер, і сам телефонний номер. Якщо у клієнта три телефонні номери, то для нього існуватимуть три екземпляри сутності телефонних номерів. Ідентифікатор такої сутності потрібно складати з номера клієнта і з телефонного номера. Уникнути використання телефонного номера як елемента ідентифікатора сутності телефонних номерів неможливо.

Як правило, при зустрічі з багатозначним атрибутом рекомендується створити ще один атрибут. Єдиним способом роботи з декількома значеннями одного атрибуту є створення сутності, декілька екземплярів якого можна берегти в базі даних, поодинці для кожного значення атрибуту.

*Про сукупність сутностей.*На початку роботи з сутностями може здатися, що природа сутності досить туманна. Розглянемо, наприклад, запас товарів. Чи є "запас товарів" сутністю? Ні. Цей запас складається з окремих товарів, з якими працює магазин. Справжньою сутністю є окремий товар. Товарний запас визначається переглядом усіх екземплярів сутності окремих товарів як єдиного цілого.

Щоб прояснити ситуацію, розглянемо атрибути, які необхідні у разі створення сутності запасу товарів: число товарів, назва товару, кількість на складі, роздрібна ціна і т.д. Оскільки весь запас товарів ми намагаємося описати за допомогою однієї сутності, для кожного з цих атрибутів потрібна по декілька значень. Проте, як було показано вище, атрибути не можуть бути багатозначними, тобто запас товарів сутністю бути не може. Його необхідно представляти у вигляді сукупності екземплярів сутності окремих товарів.

*Документування сутностей і атрибутів.*Діаграми взаємозв'язків сутностей дозволяють документувати сутності в базі даних, а також атрибути, що описують ці сутності. Є декілька типів **ER–діаграм**. Найчастіше використовуються діаграми Чена (Chen) (на ім'я винахідника ER–моделювання) і діаграми інформаційного проектування (Information Engineering). Дозволяється застосовувати діаграми будь–якого типу, головне, щоб той, хто використовує діаграму, розумів її символіку.

У обох моделях для представлення сутностей застосовуються прямокутники. Ім'я кожної сутності вказується у прямокутнику і ставиться в однині, як показано на рис.1.

Рис.1. Зображення сутності Клієнт.

У початковій ER–моделі (entity relationship) Чена не передбачена можливість відображення атрибутів безпосередньо на ER–діаграмі. Проте багато хто розширює цю модель, вносячи атрибути і укладаючи їх в овали, наприклад як на рис.2.

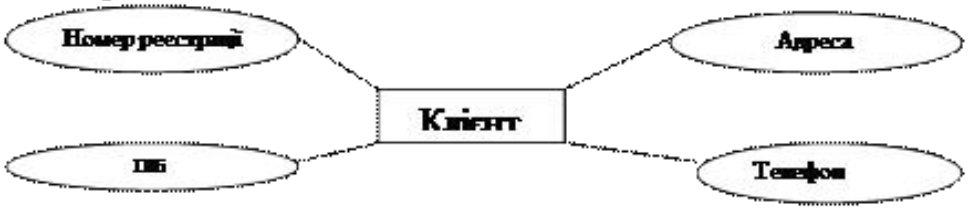


Рис.2. Зображення сутності Клієнт та її атрибутів.

Основними поняттями ER–моделі є с*утність*, *зв'язок* та *атрибут*. Кожна з частин такої діаграми повідомляє дещо про структуру даних або про те,, як ці дані співвідносяться з іншими

У методі Чена для зображення взаємозв’язків використовуються ромби, а для зображення типу взаємовідношення між сутностями — лінії із стрілками. Наприклад, на рис.3. представлено взаємовідношення між фірмою–клієнтом і замовленням.

Рис.3. Зображення взаємозв’язків між сутностями у методі Чена.

Одиночна стрілка, направлена на сутність Фірма–клієнт, указує на те, що замовлення належить максимум одному клієнту, подвійна стрілка, направлена на сутність Замовлення, означає, що клієнт може зробити один або декілька замовлень.

У рамках методу Чена існують два альтернативні способи представлення взаємозв’язків. Перший (див. мал. 4.а) припускає використання стрілок з цифрами і буквами. Тут "1" указує на те, що замовлення поступає від одного клієнта, а "М" — на те, що клієнт може робити багато замовлень.

Рис. 4.а. Використання стрілок з цифрою і буквою для зображення взаємовідношень між сутностями на ER–діаграмі.

При використанні другого способу усувається недолік, пов'язаний з легкістю для читання взаємовідношення в обох напрямах, коли ім'я взаємовідношення знаходиться усередині ромба. Вираз «Клієнт робить замовлення» має цілком певний сенс, але «Замовлення робить клієнта» — нісенітниця. Для вирішення цієї задачі ім'я взаємовідношення видаляється

з ромба і додається як до взаємовідношення, так і до його інверсії на діаграмі (див. рис.4.б). Цей варіант діаграми можна читати в будь–якому напрямі: «Клієнт робить багато замовлень» і «Замовлення робиться одним клієнтом».

Рис. 4.б. Винесення імені взаємовідношення за межі ромба на ER–діаграмі для зображення взаємовідношень між сутностями.

На етапі **даталогічного** проектування здійснюється перехід від інфологічної моделі ПО до логічної (даталогічної) моделі, яка підтримується засобами конкретної СУБД. Процес переходу від інфологічної до даталогічної моделі називається відображенням.

Даталогічна модель являє собою базу даних, структуровану на логічному рівні та орієнтовану на конкретну СУБД. Перш ніж виконати даталогічне проектування, необхідно вибрати СУБД. Кожна конкретна система накладає ряд обмежень на побудову логічної моделі даних, тому насамперед необхідно вивчити специфіку та особливості СУБД, виявити всі фактори, які можуть вплинути на логічну модель БД.

Основними факторами, що впливають на даталогічне проектування з боку СУБД, є:

1. Тип логічної моделі, що його підтримує вибрана СУБД. Зараз найпоширенішими на ринку програмних засобів і в практиці автоматизації економічних розрахунків є реляційні СУБД. Крім реляційних моделей, існують ієрархічні та мережеві моделі баз даних.

2. Особливості фізичної організації даних у середовищі вибраної СУБД. Наприклад, у СУБД Paradox чи dBASE-системах база даних організована у вигляді набору взаємозв'язаних файлів форматів DT і DBF. Усі інші об'єкти, такі як форми та звіти, також зберігаються в окремих файлах. У середовищі СУБД Microsoft Access усі дані та інструментальні засоби роботи з ними зберігаються в єдиному файлі бази даних. Тому при проектуванні БД потрібно знати не лише правила побудови логічної, а й особливості фізичної організації бази даних.

3. Кількісні обмеження, які накладає СУБД (наприклад, кількість рівнів ієрархії в ієрархічних моделях, можлива кількість полів, записів, файлів тощо). Усе ще не знайдено формалізованих методів, які б давали змогу однозначно виконати даталогічне проектування. Тому його результат багато в чому залежить від уміння та рівня кваліфікації спеціалістів, які здійснюють проектні розробки.

У результаті даталогічного проектування можна отримати кілька варіантів побудови логічної моделі даних. Тому важливим моментом є оцінка отриманих моделей і вибір найоптимальнішого варіанта.

Отриманий результат передусім потрібно оцінити з позицій відповідності наявним машинним ресурсам. У разі невідповідності цим обмеженням потрібно здійснити перепроектування БД. Крім того, на отриманій моделі необхідно перевірити умови виконання всіх запитів користувачів і вимог прикладних програм, тобто умову адекватності логічної моделі інформаційній моделі предметної області.

2 ЕТАП

Правила перетворення ER-діаграм у реляційні таблиці

Правило 1: Якщо зв'язок типу 1:1 та клас приналежності обох сутностей є обов'язковим, то потрібна лише одна таблиця. Первинним ключем цієї таблиці може бути первинний ключ будь-якої з двох сутностей.

Правило 2: Якщо зв'язок типу 1:1 і клас приналежності однієї сутності є обов'язковим, а інший - необов'язковим, необхідно побудувати таблицю для кожної сутності. Первинний ключ сутності має бути первинним ключем відповідної таблиці. Первинний ключ сутності, для якої клас приналежності є необов'язковим, додається як атрибут до таблиці сутності з обов'язковим класом приналежності .

Правило 3: Якщо зв'язок типу 1:1 і клас приналежності обох сутностей необов'язковий, необхідно побудувати три таблиці - по одній кожної сутності і одну для зв'язку. Первинний ключ сутності має бути первинним ключем відповідної таблиці. Таблиця зв'язку серед своїх атрибутів повинна мати ключі обох сутностей.

Правило 4: Якщо зв'язок типу 1:Б та клас приналежності сутності на боці Б є обов'язковим, то необхідно побудувати таблицю для кожної сутності. Первинний ключ сутності має бути первинним ключем відповідної таблиці. Первинний ключ сутності за 1 додається як атрибут в таблицю для сутності за Б.

Правило 5: Якщо зв'язок типу 1:Б та клас приналежності сутності на боці Б є необов'язковим, то необхідно побудувати три таблиці - по одній для кожної сутності та одну для зв'язку. Первинний ключ сутності має бути первинним ключем відповідної таблиці. Таблиця зв'язку серед своїх атрибутів повинна мати ключі обох сутностей.

Правило 6: Якщо зв'язок типу Б:Б, необхідно побудувати три таблиці - по одній для кожної сутності і одну для зв'язку. Первинний ключ сутності має бути первинним ключем відповідної таблиці. Таблиця зв'язку серед своїх атрибутів повинна мати ключі обох сутностей.

3 ЕТАП

Нормалізація БД

Нормалізація - це розбиття таблиці на дві або більше, які мають кращі властивості при внесенні, зміні та видаленні даних. Остаточна мета нормалізації зводиться отримання такого проекту бази даних, у якому виключено надмірність інформації. Це робиться не стільки з метою економії пам'яті, скільки для виключення можливої суперечливості даних, що зберігаються.

Перша нормальна форма (1НФ)

Перша нормальна форма наказує, що всі дані, які є у таблиці, мають бути атомарними (неподільними). Перелік відповідних атомарних типів даних визначається СУБД. Вимога 1НФ цілком природна. Воно означає, що в кожному полі кожного запису повинна бути лише одна величина, але не масив і не якась інша структура даних.

Друга нормальна форма (2НФ)

Кажуть, що таблиця знаходиться в другій нормальній формі, якщо вона знаходиться в 1НФ і кожен ключовий стовпець повністю залежить від первинного ключа. Інакше кажучи, значення кожного поля має повністю визначатися значенням первинного ключа. Важливо, що залежність від первинного ключа розуміється як залежність від ключа цілком, а чи не від окремої його складової (у разі складеного ключа).

Щоб перейти від першої нормальної форми до другої, потрібно виконати таку послідовність дій:

Визначити, на які частини можна розбити первинний ключ, так щоб деякі з неключових полів залежали від однієї з цих частин (ці частини не повинні складатися з однієї колонки).

Створити нову таблицю для кожної такої частини ключа та групи, що залежать від її полів та перемістити їх у цю таблицю. Частина колишнього первинного ключа стане первинним ключем нової таблиці.

Видалити з вихідної таблиці поля, переміщені до інших таблиць, крім тих, які стануть зовнішніми ключами.

Третя нормальна форма (3НФ)

Кажуть, що таблиця знаходиться в 3НФ, якщо вона відповідає 2НФ і всі ключові стовпці взаємно незалежні.

Взаємну залежність стовпців зручно розуміти так: стовпці є взаємно залежними, якщо не можна змінити одне із них, не змінюючи інший.

Щоб перейти від другої нормальної форми до третьої, потрібно виконати таку послідовність дій:

Визначити всі поля, від яких залежить інші поля.

Створити нову таблицю для кожного такого поля або групи полів, і перемістити їх в цю таблицю. Поле або група полів, від якого залежать переміщені поля, стане первинним ключем нової таблиці.

Видалити з вихідної таблиці поля, переміщені до інших таблиць, крім тих, які стануть зовнішніми ключами.

Вищі нормальні форми

Теоретично для реляційних баз даних розглядаються і форми вищих порядків - нормальна форма Бойса - Кодда, 4НФ, 5НФ і навіть вище. Великого практичного значення ці форми немають, і розробники, зазвичай, завжди зупиняються на 3НФ.

ПЕРЕЛІК РЕКОМЕНДОВАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Пасічник В.В., Резніченко В.А. Організація баз даних та знань. Підручник для вищих навчальних закладів. – К.: Видавнича група BHV, 2006. – 384 с.
2. Трофименко О.Г., Прокоп Ю.В., Логінова Н.І., Копитчук І.М. Організація баз даних. Навчальний посібник. 2-ге видання. – Одеса: Фенікс, 2019. – 246 с.
3. Гайна Г.А. Основи проектування баз даних: Навчальний посібник. – К.; КНУБА, 2005. – 204 с
4. Гайна Г.А. Організація баз даних і знань. Мови баз даних: Конспект лекцій.−К.:КНУБА, 2002. − 64 с
5. Гайна Г.А., Попович Н.Л. Організація баз даних і знань. Організація реляційних баз даних: Конспект лекцій.−К.:КНУБА, 2000. − 76 с.
6. М. В. Добролюбова. Програмування баз даних. Конспект лекцій. – Київ. КПІ їм. Ігоря Сікорського, 2021. – 275с.
7. Берко А.Ю., Верес О.М., Пасічник В.В. Системи баз даних та знань. Книга 1. Організація баз даних та знань. Навчальний посібник (рек.МОН України). Львів, ЛПІ, 2021. – 440с.
8. Берко А.Ю., Верес О.М. , Пасічник В.В.Системи баз даних та знань, Книга 2: Системи управління базами даних та знань. Навчальний посібник (рек.МОН України). Львів, ЛПІ, 2021. – 584с.
9. Соколова Н.О. Бази даних в інформаційних системах. Конспект лекцій. Для студентів галузі знань 12 "Інформаційні технології" спеціальності 126 "Інформаційні системи та технології". – Д.: НТУ «ДП» (електронне видання), 2024. – 280 с.
10. Адміністрування баз даними // Режим доступу: http://firebirdsql.org/manual/ru/migration- mssql-db-admin-ru.html.
11. Управління і адміністрування баз даними // Режим доступу: http://www.interface.ru/home.asp?artId=50&cId=3.