

Бази даних. SQL

План лекції

Що таке DB та їх види



SQL типи даних



SQL створення та редагування таблиць



SQL constraints



SQL маніпуляція даними



Що таке ДВ та їх ВИДИ

Бази даних

БД - сукупність даних, які зберігаються за певними правилами та використовуються для задоволення інформаційних потреб користувачів.



Види ДБ

```
graph TD; A[Види ДБ] --> B[Реляційні системи баз даних]; A --> C[Нереляційні системи баз даних]; C --> D[Ієрархічні]; C --> E[Мережеві]; C --> F[Key value]; C --> G[Document];
```

**Реляційні
системи баз
даних**

**Нереляційні
системи баз
даних**

Ієрархічні

Мережеві

Key value

Document

Nosql DB

No SQL - це підхід до проектування бази даних, який дозволяє зберігати та запитувати дані поза традиційними структурами реляційних баз даних.

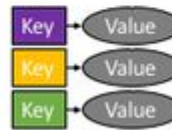
NoSQL DATABASES



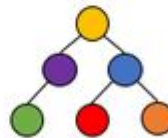
Column



Graph



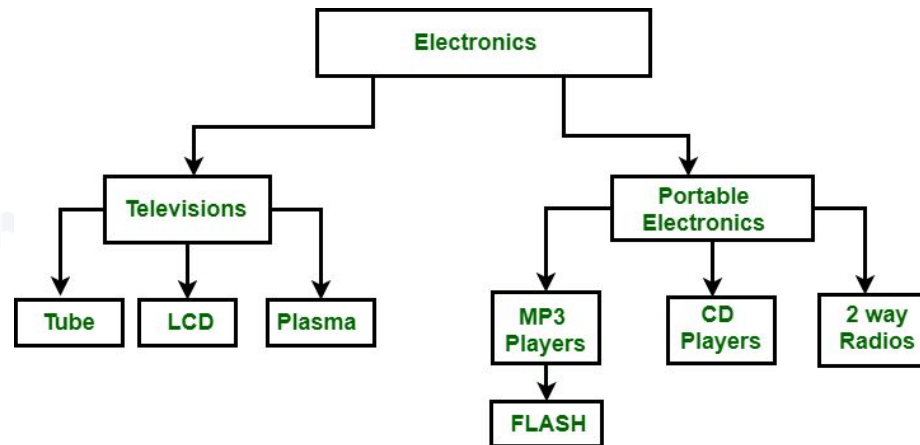
Key-Value



Document

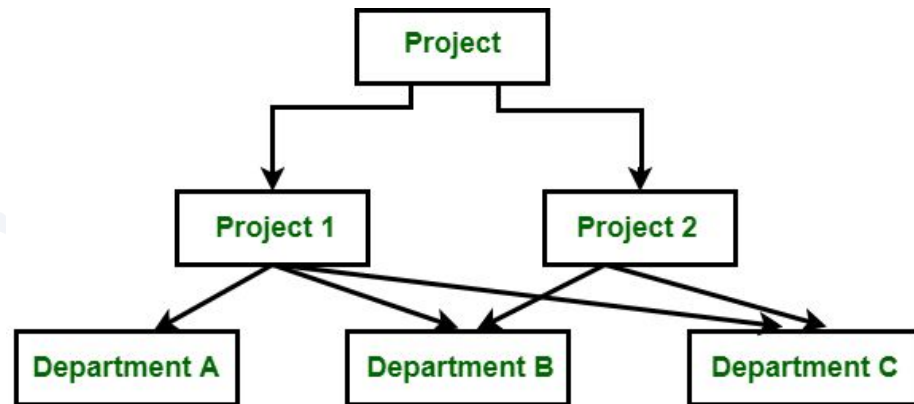
Ієрархічні

Ієрархічні - надані у вигляді перевернутого дерева, де є один кореневий запис (унікальний) та некореневі записи (унікальний ключ тільки в рамках групового відношення)



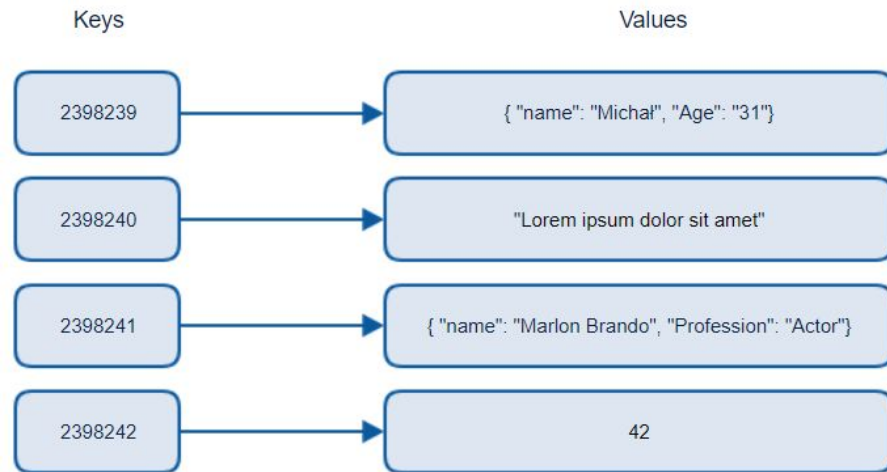
Мережеві

Мережеві - складається з наборів записів, які пов'язані між собою так, що записи можуть містити явні посилання на інші набори записів.



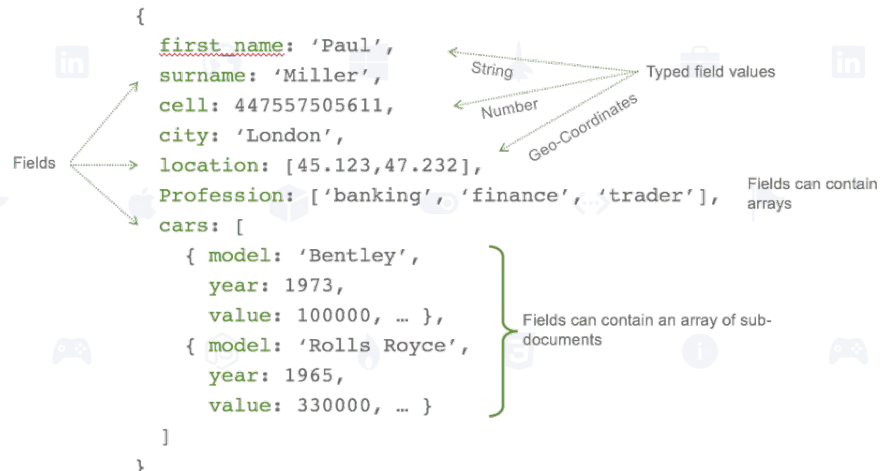
Key value

Key value - це тип нереляційної бази даних, яка використовує простий метод «ключ-значення» для зберігання даних. База даних «ключ-значення» зберігає дані як набір пар «ключ-значення», у яких ключ служить унікальним ідентифікатором. І ключі, і значення можуть бути чим завгодно, від простих об'єктів до складних складених об'єктів.



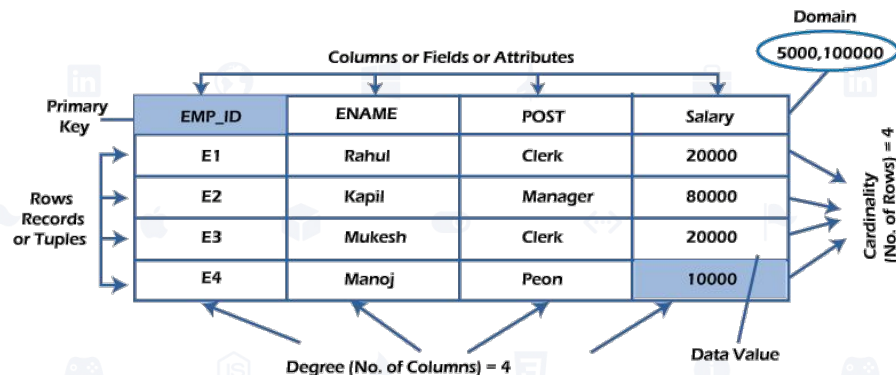
Document

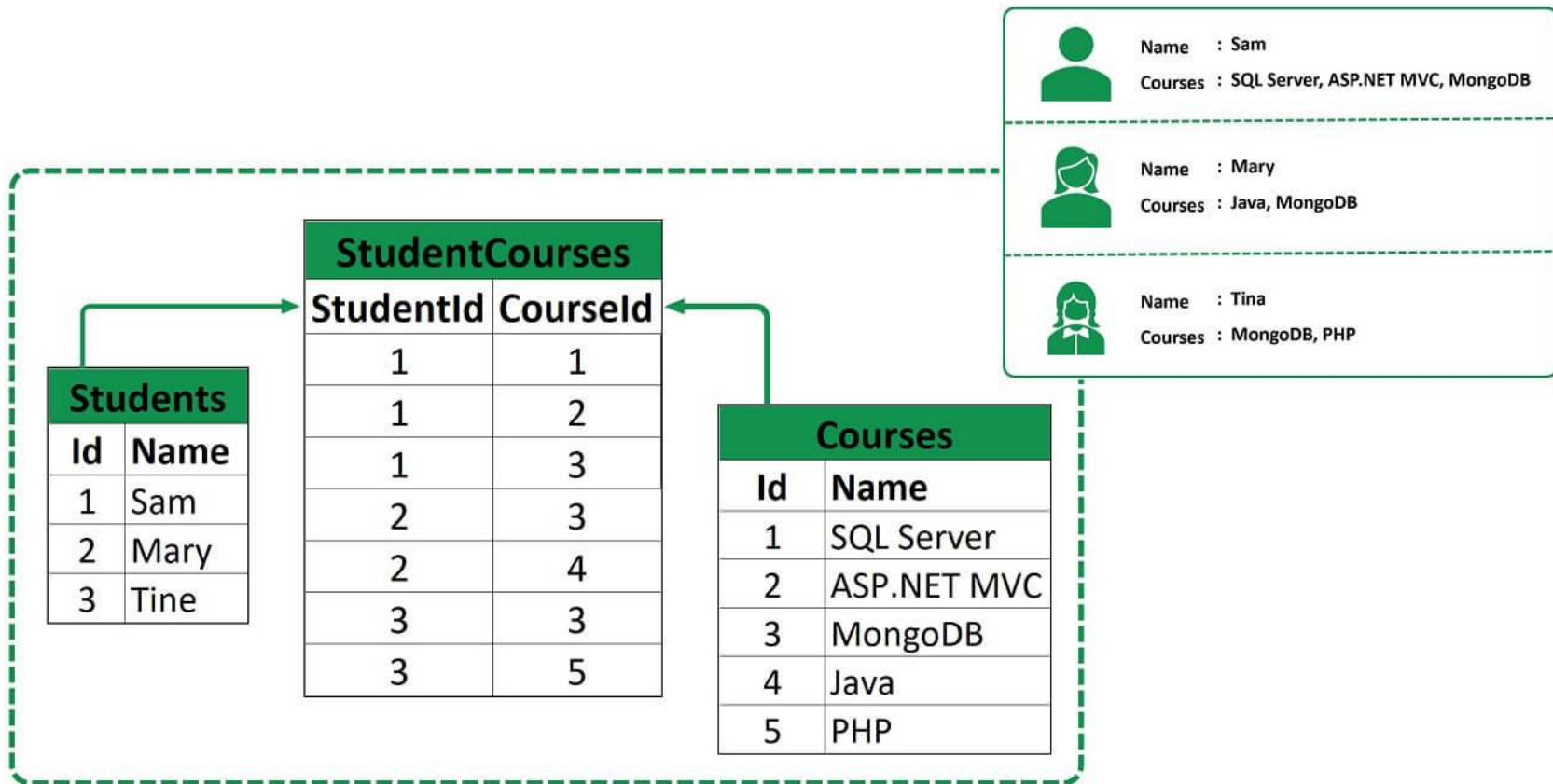
Document - це тип нереляційної бази даних, яка призначена для зберігання та запиту даних як документів, подібних до JSON



Реляційні

Реляційна - тип бази даних, яка зберігає та надає доступ до точок даних, пов'язаних одна з одною. Реляційні бази даних базуються на реляційній моделі, інтуїтивно зрозумілому, простому способі представлення даних у таблицях.







Трохи термінів

- **Сутність (entity)** - те, що представляє інтерес для користувачів бази даних, наприклад клієнти, запчастини, географічне становище і т. Д.
- **Стовпець (column)** - окремий елемент даних, що зберігається у таблиці.
- **Рядок (row)** - набір стовпців, які разом описують сутність або деякі дії, що виробляється над сутністю. Також називається записом (record).
- **Таблиця (table)** - набір рядків, що зберігається в пам'яті
- **Результуючий набір (result set)** - інша назва непостійної таблиці, зазвичай що дається результатом SQL запиту

А тепер що таке SQL



SQL

SQL - це стандартизована мова програмування, яка використовується для керування реляційними базами даних і виконання різних операцій над даними в них.





DML

- Select
- Insert
- Update
- Delete

DDL

- Create
- Alter
- Drop
- Truncate

DCL

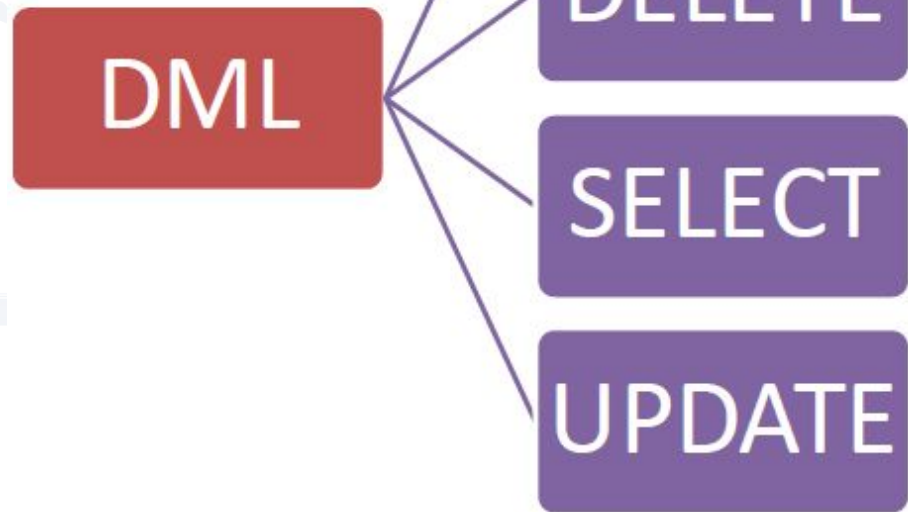
- Grant
- Revoke

TCL

- Save Point
- Roll Back
- Commit

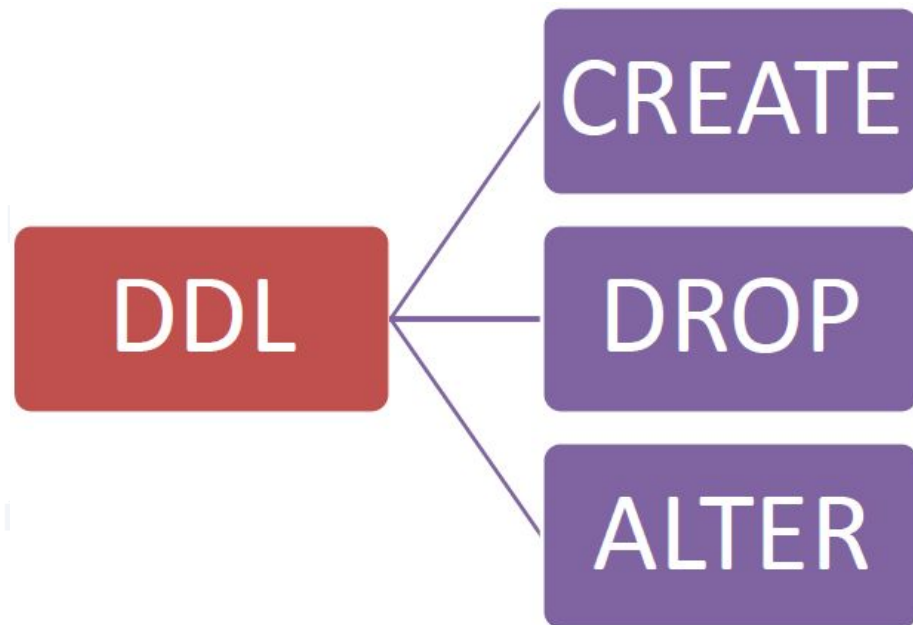
DML

Data Manipulation Language (DML) – це група операторів для маніпуляції даними. З допомогою цих операторів ми можемо додавати, змінювати, видаляти та вивантажувати дані з бази, тобто. маніпулювати ними.



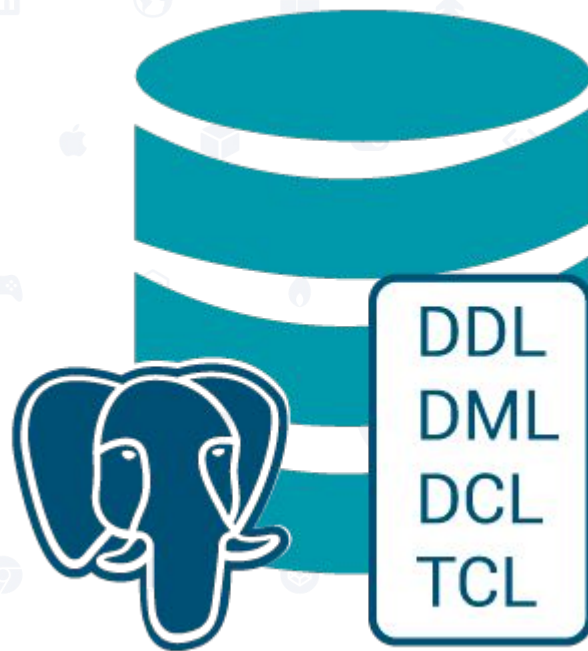
DDL

Data Definition Language (DDL) – це група операторів визначення даних. Інакше кажучи, з допомогою операторів, які входять до цієї групи, ми визначаємо структуру бази даних, і працюємо з об'єктами цієї бази, тобто. створюємо, змінюємо та видаляємо їх



DCL

Data Control Language (DCL) – група операторів визначення доступу до даних. Іншими словами, це оператори для керування дозволами, за допомогою яких ми можемо дозволяти або забороняти виконання певних операцій над об'єктами бази даних.



TCL

Transaction Control Language (TCL) – група операторів для керування транзакціями.

Транзакція – це команда чи блок команд (інструкцій), які успішно завершуються як єдине ціле, причому у базі даних усі внесені зміни фіксуються постійно чи скасовуються, тобто. усі зміни, внесені будь-якою командою, що входить до транзакції, буде скасовано.





Також існує таке поняття як СУБД



СУБД

СУБД - сукупність програмних та лінгвістичних засобів загального або спеціального призначення, що забезпечують управління створенням та використанням баз даних.



PostgreSQL



SYBASE

SQL типи даних

Числові типи

TINYINT	представляє цілі числа від -127 до 128, займає 1 байт
BOOL	фактично не представляє окремий тип, а є лише псевдонімом для типу TINYINT(1) і може зберігати два значення 0 і 1. Однак даний тип може також приймати вбудовані константи TRUE (представляє число 1) і FALSE (надає число 0).
SMALLINT	представляє цілі числа від -32768 до 32767, займає 2 байти
MEDIUMINT	представляє цілі числа від -8388608 до 8388607, займає 3 байти
INT	представляє цілі числа від -2147483648 до 2147483647, займає 4 байти
Unsigned - дає можливість усунути негативне значення	

Числові типи

BIGINT	Представляє цілі числа від -9223372036854775808 до 9223372036854775807, займає 8 байт
DECIMAL	Зберігає цифри з фіксованою точністю. Цей тип може приймати два параметри precision та scale: <code>DECIMAL(precision, scale)</code>. Параметр precision представляє максимальну кількість цифр, які можуть зберігати число. Це значення має бути в діапазоні від 1 до 65. Параметр scale представляє максимальну кількість цифр, які можуть містити число після коми. Це значення має знаходитися в діапазоні від 0 до значення параметра precision. За умовчанням воно дорівнює 0. <code>salary DECIMAL(5,2)</code> Число 5 - precision, а число 2 - scale, тому даний стовпець може зберігати значення діапазону від -999.99 до 999.99.
FLOAT	Зберігає дробові числа з плаваючою точкою одинарної точності від -3.4028×10^{38} до 3.4028×10^{38} , займає 4 байти. Може набувати форми <code>FLOAT(M,D)</code> , де M - загальна кількість цифр, а D - кількість цифр після коми
DOUBLE	Зберігає дробові числа з плаваючою точкою подвійної точності від -1.7976×10^{308} до 1.7976×10^{308} , займає 8 байт. Також може набувати форми <code>DOUBLE(M,D)</code> , де M - загальна кількість цифр, а D - кількість цифр після коми.

Символьні типи

CHAR	Представляє рядок фіксованої довжини. Довжина рядка, що зберігається, вказується в дужках, наприклад, CHAR(10) - рядок з десяти символів. І якщо в таблицю в даний стовпець зберігається рядок із 6 символів (тобто менше встановленої довжини в 10 символів), то рядок доповнюється 4 пробілами і в результаті все одно займатиме 10 символів.
VARCHAR	Представляє рядок змінної довжини. Довжина рядка, що зберігається, також вказується в дужках, наприклад, VARCHAR(10). Однак на відміну від CHAR рядок, що зберігається, буде займати саме стільки місця, скільки необхідно. Наприклад, якщо певна довжина в 10 символів, але в стовпець зберігається рядок у 6 символів, то рядок, що зберігається, так і займатиме 6 символів плюс додатковий байт, який зберігає довжину рядка.
Починаючи з MySQL 5.6 типи CHAR і VARCHAR за замовчуванням використовують кодування UTF-8, яке дозволяє використовувати до 3 байт для зберігання символу в залежності від мови (для багатьох європейських мов по 1 байту на символ, для ряду східноєвропейських і близькосхідних - 2 байти , а для китайської, японської, корейської - по 3 байти на символ).	

Символьні типи

TINYTEXT	Представляє текст довжиною до 255 байт.
TEXT	Представляє текст довжиною до 65 КБ.
MEDIUMTEXT	Представляє текст довжиною до 16 МБ
LARGETEXT	Представляє текст довжиною до 4 ГБ

Типи для роботи з датою та часом

DATE	Зберігає дати з 1 січня 1000 року до 31 грудня 9999 року (з "1000-01-01" до "9999-12-31"). За промовчанням для зберігання використовується формат уууу-mm-dd. Займає 3 байти.
TIME	Зберігає час від -838:59:59 до 838:59:59. За умовчанням для зберігання часу використовується формат "hh:mm:ss". Займає 3 байти.
DATETIME	Об'єднує час і дату, діапазон дат і часу - з 1 січня 1000 року до 31 грудня 9999 року (з "1000-01-01 00:00:00" до "9999-12-31 23:59:59"). Для зберігання за промовчанням використовується формат "уууу-mm-dd hh:mm:ss". Займає 8 байт
TIMESTAMP	Також зберігає дату та час, але в іншому діапазоні: від "1970-01-01 00:00:01" UTC до "2038-01-19 03:14:07" UTC. Займає 4 байти
YEAR	Зберігає рік у вигляді 4 цифр. Діапазон доступних значень від 1901 до 2155 року. Займає 1 байт.

Складові типи

ENUM	Зберігає одне значення зі списку допустимих значень. Займає 1-2 байти
SET	Може зберігати кілька значень (до 64 значень) із певного списку допустимих значень. Займає 1-8 байт.

SQL створення та редагування таблиць

Розберемо приклад створення таблиці та інші команди DDL



DDL commands

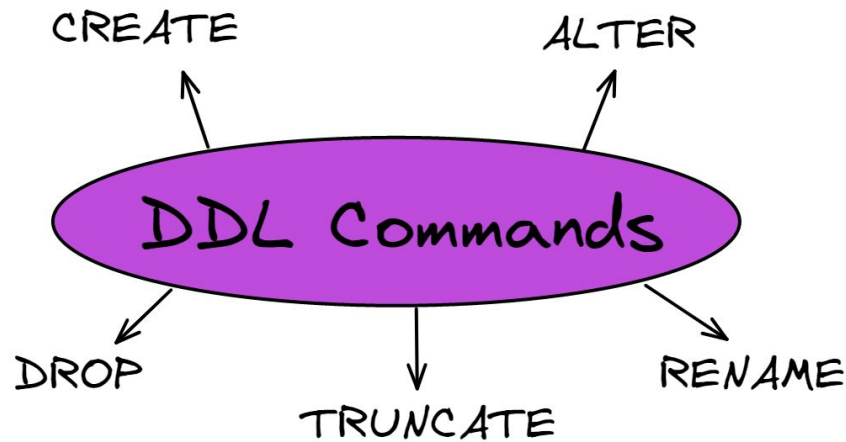
CREATE – використовується для створення об'єктів бази даних;

ALTER – використовується для зміни об'єктів бази даних;

DROP – використовується для видалення об'єктів бази даних.

TRUNCATE – служить для очищення всіх даних

RENAME – використовуйте команду **ALTER TABLE RENAME**, щоб перейменувати назви стовпців.





```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ...  
);
```

Давайте створимо таблицю з полями

- Ім'я
- Прізвище
- Рік народження
- Стать

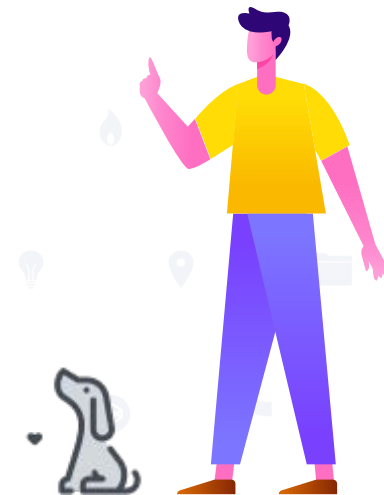




```
CREATE TABLE users(  
    id int unsigned,  
    userName varchar(255),  
    surname varchar(255),  
    birthday DATE,  
    sex enum ('male', 'female')  
)
```

Також можна створювати таблицю на основі іншої таблиці ⚙️

```
CREATE TABLE users2 as (select id,  
userName,surname, sex from users)
```





Додамо колонку до таблиці

```
ALTER TABLE table_name  
ADD column_name datatype;
```

Приклад:

```
ALTER TABLE users2 add (salary  
double(10,2) )
```

Зверніть увагу на double(10,2), 10 це загальна кількість цифр,
2 цифр після коми.



Змінити колонку до таблиці

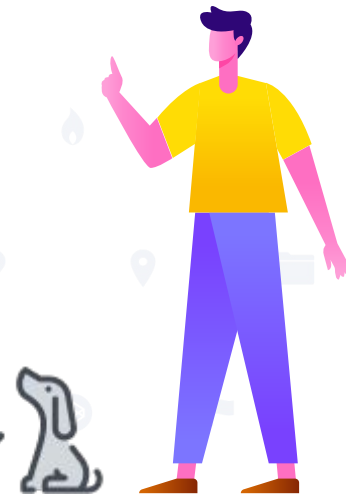
```
ALTER TABLE table_name
```

```
MODIFY COLUMN column_name datatype;
```

Приклад

```
ALTER TABLE users2
```

```
MODIFY COLUMN sex enum  
( 'male', 'female', 'not decided' );
```



Як видалити колонку

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

Приклад

```
ALTER TABLE users2  
DROP COLUMN salary;
```



Змінити назву таблиці

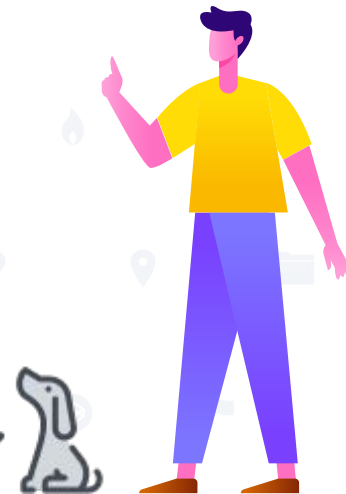
```
ALTER TABLE table_name
```

```
RENAME COLUMN column_name to  
column_name;
```

Приклад

```
ALTER TABLE user2
```

```
RENAME COLUMN sex to gender
```





Видалення таблиці

```
DROP TABLE  
schema.table_name;
```

Приклад

```
DROP TABLE example.user2
```

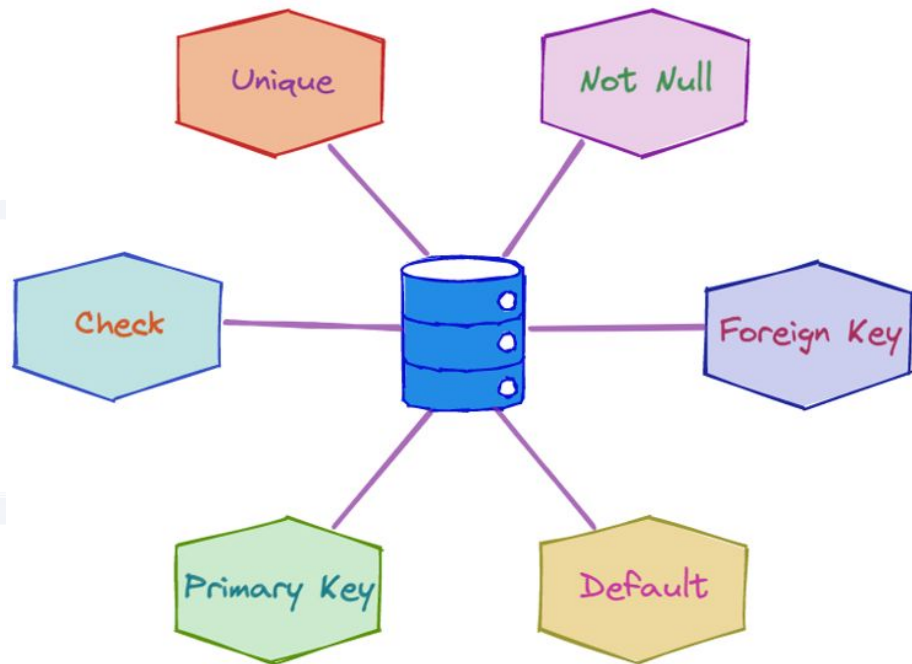


SQL constraints

Constraint

CONSTRAINT використовується в інструкціях `ALTER TABLE` та `CREATE TABLE` для створення та видалення обмежень.

- Not null
- Unique
- Check
- Primary key
- Foreign key
- Default



Not null

Null - у деяких випадках неможливо надати значення певного стовпця таблиці, означає, що поле пусте, без значення.

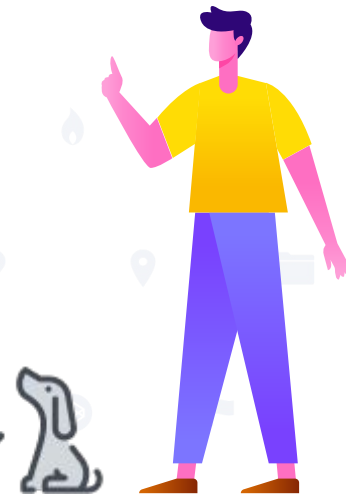
Not null - не дозволяє стовпцям утримувати значення null.

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
  
    LastName varchar(255) NOT NULL,  
  
    FirstName varchar(255) NOT NULL,  
  
    Age int  
);
```



Змінити Constraint

```
ALTER TABLE Persons  
MODIFY Age NOT NULL;
```



UNIQUE

Обмеження **UNIQUE** гарантує, що значення у стовпці різняться.

```
CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255) NOT NULL,  
    Age int  
);
```

Check

Обмеження **CHECK** використовується для обмеження діапазону значень, які можуть бути розміщені в стовпчику.

```
CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
  
    LastName varchar(255) NOT NULL,  
  
    FirstName varchar(255) NOT NULL,  
  
    Age int,  
    CHECK (Age >= 18)  
  
);
```

Primary key

Первинний ключ - набір певних ознак, унікальних кожного запису.

Primary key не дозволяє створювати однакових записів (рядків) у таблиці;

Первинні ключі повинні містити унікальні значення та не можуть містити значення **NULL**.

Primary key може бути лише один у таблиці.

Також можна побачити **AUTO_INCREMENT**, використовується для автоматичного збільшення

```
CREATE TABLE Persons (  
  
    ID int PRIMARY KEY AUTO_INCREMENT,  
  
    LastName varchar(255) NOT NULL,  
  
    FirstName varchar(255) NOT NULL,  
  
    Age int,  
  
    CHECK (Age>=18)  
);
```

Foreign key

Стовпець (або набір стовпців) значення якого
посилаються на первинний ключ іншої таблиці

```
CREATE TABLE Faculty(  
    ID int PRIMARY KEY,  
    FacultyName varchar(255) NOT NULL,  
    PersonId int,  
    FOREIGN KEY (PersonId) REFERENCES Person (ID)  
);
```

<u>studentId</u>	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

Foreign Keys

Relationship

Primary Keys

<u>courseId</u>	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

Default

Default – обмеження використовується для встановлення значення за замовчуванням для стовпця.

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255) DEFAULT 'Sandnes'  
);
```

Типи зв'язків у базах даних



Один до одного



Один до багатьох



багато до багатьох

Один до одного

User Info

ID	passpor_number	licenseId
1	KM1222	1
2	KM1224	2
3	KM1225	3

Driver license

ID	license_number
1	123456
2	123457
3	123458

Один до багатьох

Students

ID	name	teacherId
1	Mikola	2
2	Andrii	2
3	Oksana	3

Teachers

ID	name
1	Oleksandr
2	Yevhenii
3	Anton



Багато до багатьох



actors

ID	Full name
1	Leo Dicaprio
2	Kate Winslet
3	Billy Zane

Films

ID	name
1	Titanic
2	Revolutionary Road
3	Mad Dogs

actors_films

actor_id	film_id
1	1
1	2
2	1
2	2

SQL маніпуляція даними

DML commands

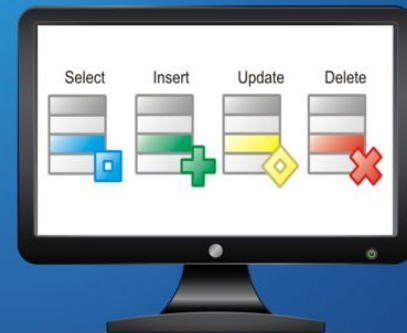
Insert - використовується для вставки нових записів у таблицю.

Update - використовується для зміни існуючих записів у таблиці.

Delete - використовується для видалення існуючих записів у таблиці.

Select - використовується для вибору даних із бази даних.

Data Manipulation Language



Вставка даних до таблиці

```
INSERT INTO table_name (column1,  
column2, column3, ...)  
  
VALUES (value1, value2, value3, ...);
```

Приклад

```
INSERT INTO users  
  
(id, userName, surname, birthday, sex)  
  
VALUES  
  
(1, 'Test', 'Test', '2023-04-26', 'male')
```



Зміна існуючих записів

```
UPDATE table_name
```

```
SET column1 = value1, column2 = value2,  
...
```

```
WHERE condition;
```

Приклад

```
UPDATE users
```

```
SET userName = 'Hello'
```

```
Where id = 1
```





Видалення існуючих записів

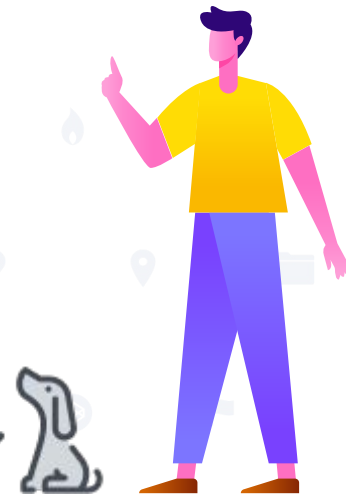
```
DELETE FROM table_name WHERE condition;
```

Приклад

```
DELETE FROM users Where id = 3
```

Також можна видалити відразу всі дані

```
DELETE FROM users
```





**Так, але видалення даних
в таблиці це TRUNCATE.
В чому різниця ?**



DELETE	TRUNCATE
DML command	DDL command
Може використовувати where	Не можна використовувати where
Працює повільніше	Миттєве видалення
Після видалення можна відновити дані	Дані не можна відновити

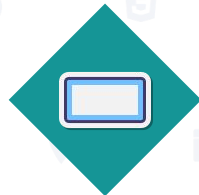


SELECT

3 фундаментальні концепції



Projection - вибір стовпів з таблиці



Selection - вибір рядків з таблиці



Joining – об'єднання таблиць



Далі всі приклади будуть
показані у БД qauto

Select projection

Select - це statement (не змінює дані).

Повертаються дані які зберігаються в таблиці результатів, яка називається результуючим набором.

```
SELECT * FROM table;
```

```
SELECT columns FROM table;
```

Приклад

```
SELECT * FROM users
```

```
SELECT email,password FROM users
```

DISTINCT

Students

ID	name	age
1	Petro	21
2	Mikola	21
3	Nadia	33

```
SELECT DISTINCT age  
FROM students;
```

Команда DISTINCT дозволяє вибирати лише унікальні значення з бази даних (тобто відсіває дублі:). Запит виведе всі унікальні age.

EXPRESSIONS

SQL expression – це комбінація одного чи кількох значень, операторів і функцій SQL, яка призводить до значення. Ці ВИРАЗИ SQL схожі на формулу та написані мовою запитів. Ви також можете використовувати їх для запиту до бази даних для певного набору даних.

```
SELECT columns, expressions FROM  
table;
```

Приклад

```
SELECT *, totalCost * 3 FROM  
expenses
```

SELECTION. Оператор WHERE

Оператор **WHERE** може використовуватися для вибірки рядків за допомогою **SELECT**, видалення рядків за допомогою **DELETE**, для редагування рядків за допомогою **UPDATE**.

```
SELECT columns, expressions  
FROM table  
WHERE conditions;
```

Приклад

```
SELECT *  
FROM users  
WHERE id = 1;  
Або  
SELECT *  
FROM users  
WHERE createdAt = updatedAt;
```

Математичні оператори порівняння

=	Дорівнює	<code>SELECT email FROM users WHERE id = 2;</code>
!=	Не дорівнює	<code>SELECT email FROM users WHERE id != 2;</code>
>	Більше	<code>SELECT email FROM users WHERE id > 2;</code>
<	Менше	<code>SELECT email FROM users WHERE id < 2;</code>
>=	Більше або дорівнює	<code>SELECT email FROM users WHERE id >= 2;</code>
<=	Менше або дорівнює	<code>SELECT email FROM users WHERE id <= 2;</code>



Оператори порівняння

Between	Дорівнює	<code>SELECT * FROM users WHERE id BETWEEN 2 AND 10;</code>
IN	Включає	<code>SELECT * FROM users WHERE id IN(1,2,3,10);</code>
IS NULL	Не null	<code>SELECT * FROM user_profiles WHERE dateBirth IS NULL;</code>
NOT	Додається до операторів порівняння	<code>SELECT * FROM users WHERE id NOT BETWEEN 2 AND 10;</code> <code>SELECT * FROM users WHERE id NOT IN(1,2,3,10);</code> <code>SELECT * FROM user_profiles WHERE dateBirth IS NOT NULL;</code>

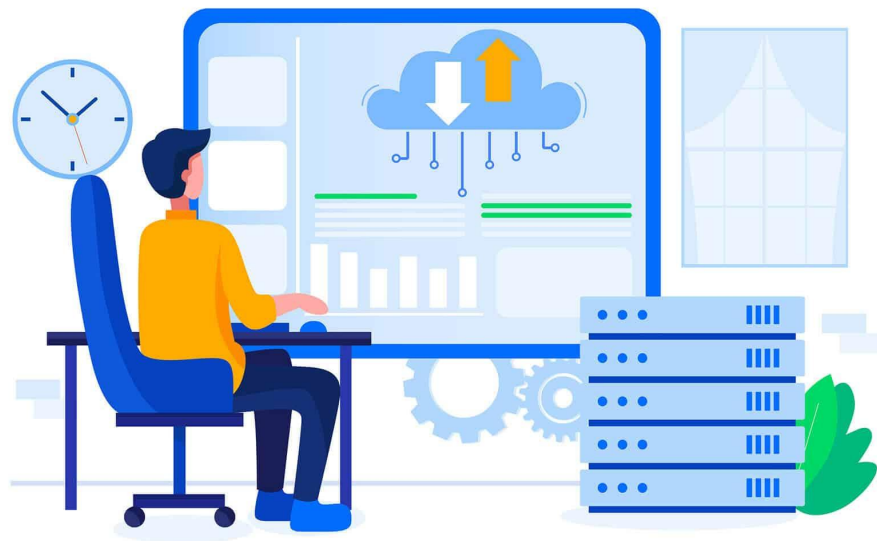


LIKE

Оператор **LIKE** використовується в реченні **WHERE** для пошуку заданого шаблону в стовпці.

У поєднанні з оператором **LIKE** часто використовуються два символи підстановки:

- Знак відсотка (%) означає нуль, один або декілька символів
- Знак підкреслення (_) означає один символ





Оператор	Опис
'a%'	Знайде будь-які значення, які починаються з a
'%a'	Знайде будь-які значення, які закінчуються буквою a
'%or%'	Знайде будь-яке значення, яке має or в будь-якій позиції
'_r%'	Знайде будь-які значення в яких літера r друга
'a_%'	Знайде будь-які значення, що починаються з a, у яких довжина мінімум два значення
'a%o'	Знайде значення с початком a і закінчується o



Використання LIKE

```
SELECT column1, column2, ...  
FROM table_name  
WHERE column LIKE pattern;
```

Приклад

```
SELECT * FROM users  
WHERE email LIKE 'Test';
```

```
SELECT * FROM users  
WHERE email LIKE '_est';
```



Логічні оператори



AND

весь вираз набуває значення true тоді і лише тоді, коли всі умови, об'єднані and, окремо повертають true



OR

весь вираз набуває значення TRUE тоді і лише тоді, коли хоча б одна з умов, об'єднаних OR, окремо теж повертає TRUE



NOT

змінює значення умови на протилежне.



```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND
condition3 ...;
```

Приклад

```
SELECT * FROM users WHERE email like  
'_mail%' AND createdAt = '2021-09-02  
17:52:49';
```

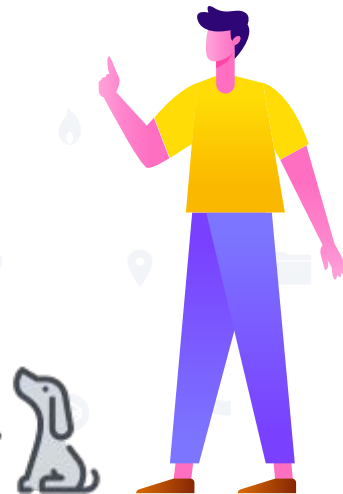


OR

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 OR condition2 OR  
condition3 ...;
```

Приклад

```
SELECT *  
FROM users  
WHERE createdAt= '2021-09-02 17:52:49'  
OR createdAt = '2021-09-02 17:54:33';
```



NOT

```
SELECT column1, column2, ...  
FROM table_name  
WHERE NOT condition;
```

Приклад

```
SELECT *  
FROM users  
WHERE email NOT LIKE '_email%'
```



ORDER BY

Команда **ORDER BY** дозволяє сортувати записи за певним полем при виборі з бази даних.

За замовчуванням записи сортуються за зростанням **ASC**, щоб відсортувати за спаданням - поставте **DESC**:

```
SELECT column1, column2, ...  
FROM table_name  
ORDER BY column1, column2, ...  
ASC | DESC;
```

Приклад

```
SELECT *  
FROM users  
WHERE id BETWEEN 1 and 10 ORDER BY  
id DESC
```



Функції в SQL



SQL функції

Single row

Numeric

POWER

ROUND

String

LOWER

UPPER

CONCAT

LENGTH

Date

NOW

QUARTER

WEEK

Multi row/ Aggregate

COUNT

SUM

AVG

MAX

MIN

**Варто зауважити що це
тільки маленька кількість
функцій і їх набагато більше.**



STRING FUNCTIONS

LOWER	Перетворює рядок у нижній регістр	<code>SELECT LOWER(name) FROM user_profiles</code>
UPPER	Перетворює рядок у верхній регістр	<code>SELECT UPPER(name) FROM user_profiles</code>
CONCAT	Додає два або більше виразів разом	<code>SELECT CONCAT(name, ' and my surname', lastName) FROM user_profiles</code>
LENGTH	Повертає довжину рядка	<code>SELECT LENGTH(name) FROM user_profiles</code>

NUMERIC FUNCTIONS

POWER	Повертає значення числа, зведеного до степеня іншого числа	<code>SELECT POWER(id,2) FROM user_profiles</code>
ROUND	Округлює число до вказаної кількості знаків після коми	<code>SELECT *, ROUND(totalCost,0) FROM expenses WHERE carId = 43241</code>

DATE FUNCTIONS

NOW	Повертає поточну дату й час	<code>SELECT NOW () ;</code>
QUARTER	Повертає квартал року для заданого значення дати	<code>SELECT QUARTER ("2017-06-15") ;</code>
WEEK	Повертає номер тижня для вказаної дати	<code>SELECT WEEK ("2017-06-15") ;</code>



**Давайте тепер розглянемо
функції агрегації.**



AGGREGATE FUNCTIONS

COUNT	Повертає кількість записів, повернутих запитом на вибірку	<code>SELECT COUNT (totalCost) FROM `expenses` WHERE totalCost > 100</code>
SUM	Обчислює суму набору значень	<code>SELECT SUM (totalCost) FROM `expenses` WHERE totalCost > 100</code>
MIN	Повертає мінімальне значення в наборі значень	<code>SELECT MIN (totalCost) FROM `expenses`</code>
MAX	Повертає максимальне значення в наборі значень	<code>SELECT MAX (totalCost) FROM `expenses`</code>
AVG	Повертає середнє значення виразу	<code>SELECT MIN (totalCost) FROM `expenses`</code>

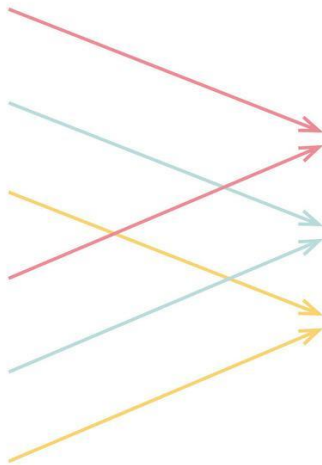
Дуже часто коли ми використовуємо функції агрегації нам необхідно якось згрупувати ці результати. Для цього нам знадобиться оператор **GROUP BY**.

Інструкція **GROUP BY** групує рядки з однаковими значеннями в підсумковій рядки, як-от «знайти кількість клієнтів у кожній країні»





title	genre	qty
book 1	adventure	4
book 2	fantasy	5
book 3	romance	2
book 4	adventure	3
book 5	fantasy	3
book 6	romance	1



genre	total
adventure	7
fantasy	8
romance	3



GROUP BY

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Приклад.

Необхідно знайти кількість авто у користувачів.

```
SELECT COUNT(carBrandId),userId
FROM cars
GROUP BY userId
```

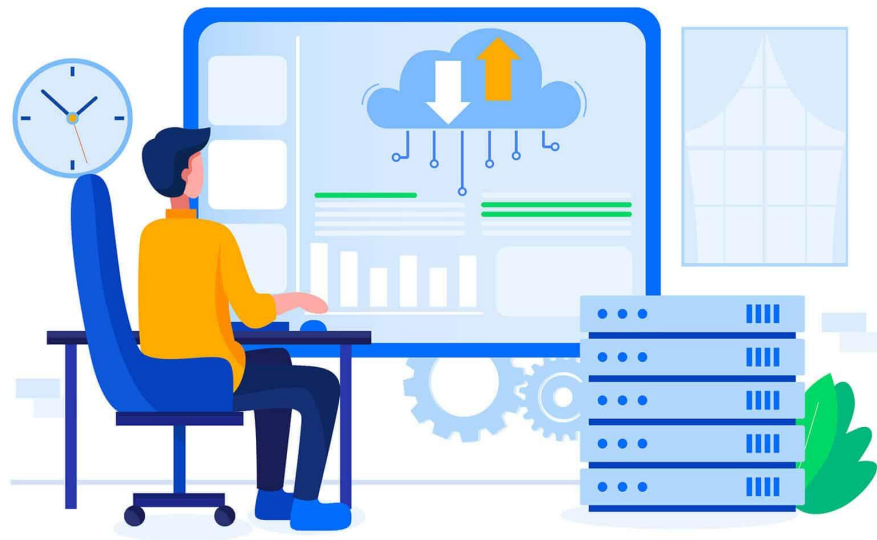




**Якщо мені необхідно якось
відфільтрувати знайдені
дані ?**

HAVING

Речення **HAVING** було додано до **SQL**, оскільки ключове слово **WHERE** не можна використовувати з агрегатними функціями.



```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Приклад

```
SELECT COUNT(carBrandId),userId
FROM cars
GROUP BY userId
HAVING userId BETWEEN 200 and 300
```



Розглянемо дуже важливу тему в SQL це

JOINS

JOIN використовується для об'єднання рядків з двох або більше таблиць на основі пов'язаного стовпця між ними.

Розглянемо основні JOINS

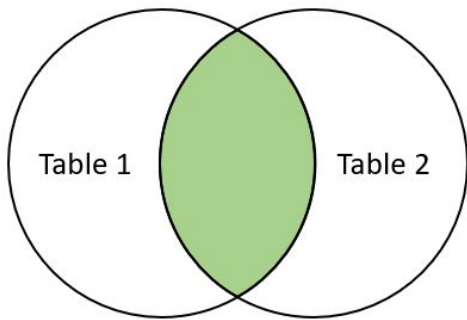
INNER JOIN - вибираються тільки збігаються дані з таблиць, що об'єднуються

LEFT JOIN - вибираються всі дані таблиці зліва та збігаються дані таблиці праворуч

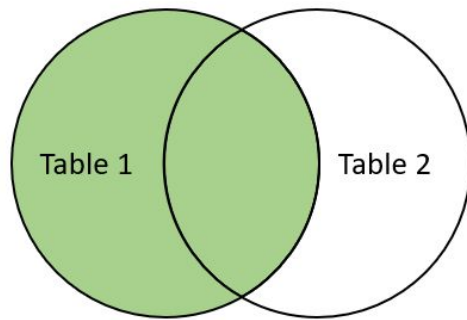
RIGHT JOIN – повертає всі рядки з правої таблиці та відповідні записи з лівої таблиці.

FULL JOIN - повертає всі записи, якщо є відповідність у лівих (таблиця1) або правих (таблиця2) записах таблиці. (MySQL не підтримує)

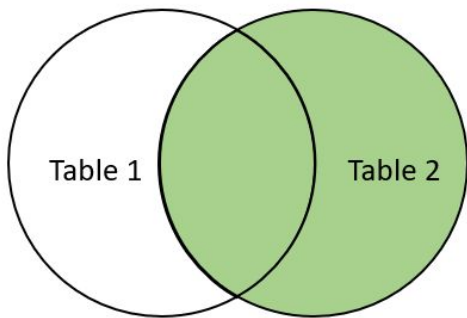




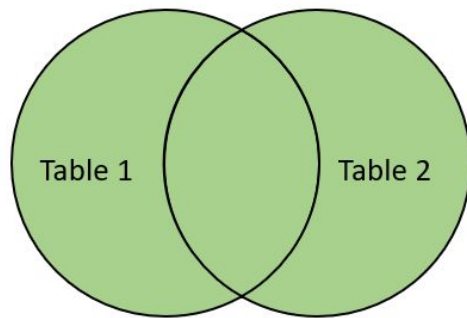
INNER JOIN



LEFT JOIN

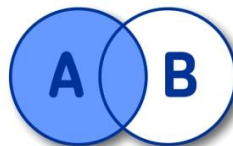


RIGHT JOIN

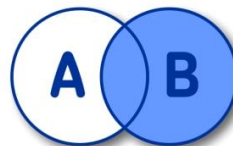


FULL JOIN

SQL JOINS



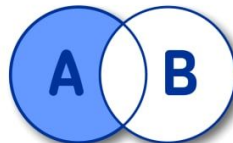
SELECT * FROM
A **LEFT** JOIN B
ON A.KEY = B.KEY



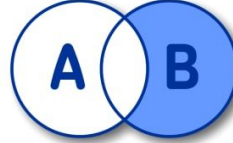
SELECT * FROM
A **RIGHT** JOIN B
ON A.KEY = B.KEY



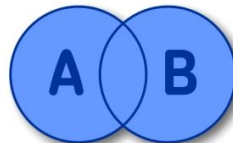
SELECT * FROM
A **INNER** JOIN B
ON A.KEY = B.KEY



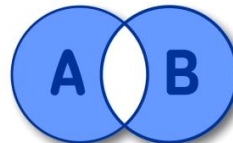
SELECT * FROM A
LEFT JOIN B
ON A.KEY = B.KEY
WHERE B.KEY IS NULL



SELECT * FROM A
RIGHT JOIN B
ON A.KEY = B.KEY
WHERE A.KEY IS NULL



SELECT * FROM A
FULL OUTER JOIN B
ON A.KEY = B.KEY



SELECT * FROM A **FULL**
OUTER JOIN B ON A.KEY =
B.KEY WHERE A.KEY IS
NULL OR B.KEY IS NULL

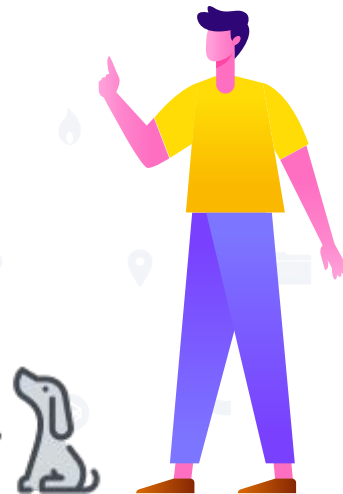


INNER JOIN

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

Приклад

```
SELECT * FROM `cars`
INNER JOIN users
ON cars.userId = users.id
```





LEFT JOIN

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

Приклад

```
SELECT * FROM `cars`
LEFT JOIN users
ON cars.userId = users.id
```

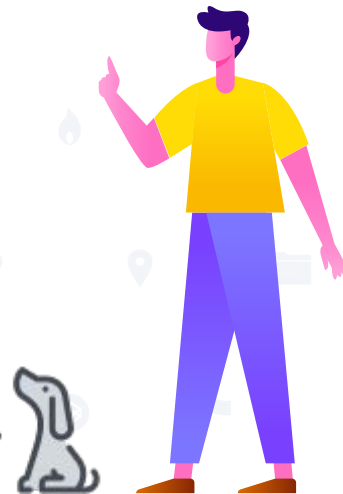


RIGHT JOIN

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

Приклад

```
SELECT * FROM `cars`
RIGHT JOIN users
ON cars.userId = users.id
```





**Якщо в MySQL не
підтримується full join як
його отримати ?**



FULL JOIN

Приклад

```
SELECT *  
FROM cars  
LEFT JOIN users  
ON cars.userId = users.id  
UNION ALL  
SELECT *  
FROM cars  
RIGHT JOIN users  
ON cars.userId = users.id
```





**Але можна об'єднувати не
тільки дві таблиці але
більше**

Вивести name, lastName користувачів у яких є авто. Вивести carBrand цього авто та його title

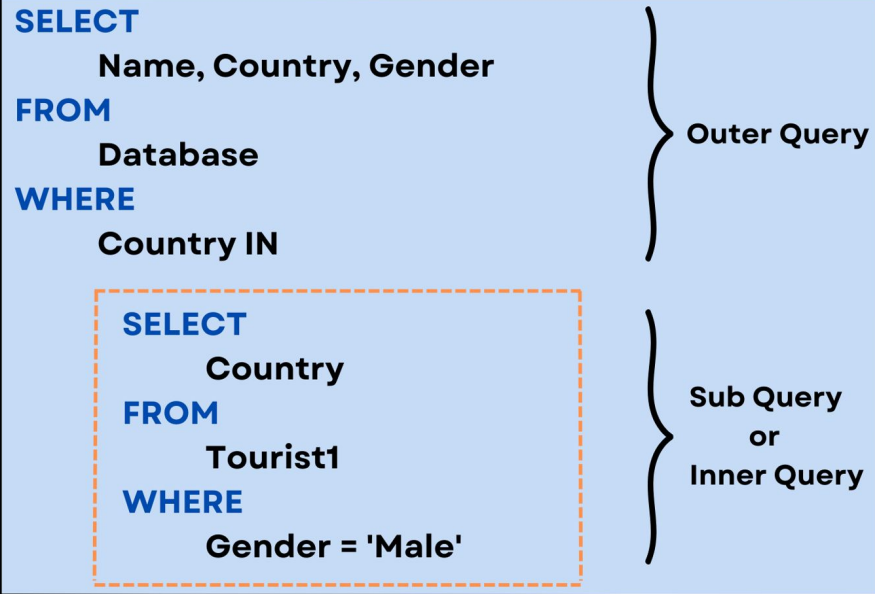
Приклад

```
SELECT name, lastName, cars.carBrandId,  
car_brands.title  
FROM `user_profiles` JOIN cars ON  
user_profiles.userId = cars.userId JOIN  
car_brands ON cars.carBrandId =  
car_brands.id
```



SUBQUERY

Вкладений запит (SUBQUERY) — це запит, який використовується всередині інструкцій SELECT, INSERT, UPDATE або DELETE або всередині іншого вкладеного запиту.



```
SELECT
    Name, Country, Gender
FROM
    Database
WHERE
    Country IN
    (
        SELECT
            Country
        FROM
            Tourist1
        WHERE
            Gender = 'Male'
    )
```

The diagram illustrates a SQL query structure. The outer query is defined by the first four lines: **SELECT**, **Name, Country, Gender**, **FROM**, **Database**, and **WHERE**. The inner query, which is a subquery or inner query, is enclosed in a dashed orange box and consists of the following lines: **SELECT**, **Country**, **FROM**, **Tourist1**, **WHERE**, and **Gender = 'Male'**. Brackets on the right side of the diagram group the lines into two sections: the top section is labeled "Outer Query" and the bottom section is labeled "Sub Query or Inner Query".



SUBQUERY можна використовувати при взаємодії з функціями агрегації. Наприклад, нам необхідно знайти всі машини у яких initialMileage більше середнього.

```
SELECT *  
FROM cars  
WHERE initialMileage >  
(  
  SELECT AVG(initialMileage)  
  FROM cars  
)
```



SUBQUERY можна використовувати також для роботи з операціями projection. Наприклад, нам необхідно знайти всі максимальні totalCost та їх carId.

```
SELECT  
(  
  SELECT MAX(totalCost)  
    FROM expenses  
) , carId  
FROM `expenses`
```





ANY QUESTIONS ?