DATABASE - це набір даних, що зберігаються у легко доступному форматі, для керування ними.

Relational Databases - реляційні бази даних. Де інформація розміщується у вигляді таблиць. А також таблиці пов'язані між собою.

SQL - Structured Query Languages - це мова запитів даних. Набір даних, що зберігаються у легко доступному форматі, для керування ними. Ми створюємо таблиці, колонки рядки в них, зв’язки між таблицями. Контроль даступу до даних (визначати користувачів, які можуть редагувати дані або лише читати дані). Для вставки, оновлення, видалення даних з бази даних.

Queries - запити які ми пишемо та які повідомляють базу даних, яку інформацію ми хочемо з неї отримати. З якої таблиці, що саме, відфільтрувати інформацію і т.д.

```
SELECT employee.name, employee.age
FROM employee
WHERE employee.salary > 30000
```

Column - колонка, представляє один атрибут усіх записів в таблиці

Rows - рядок в таблиці, представляє повністю весь запис, інформацію наприклад про студента, з різними атрибутами (**student_id**, **fname**, **age**, **major**)

Кожна таблиця в реляційній базі даних має колонку (атрибут) **PRIMARY KEY**, це унікальний атрибут кожного запису, за допомогою якого, ми ідентифікуємо (знаходимо) його. Наприклад **student_id**. Може бути 2 ключі в таблиці

FOREIGN KEY - зовнішній ключ. Це атрибут, який ми зберігаємо в таблиці бази даних і це зв'яже нас з іншою таблицею. В **FOREIGN KEY** ми зберігаємо **PRIMARY KEY** іншої таблиці з якою створюємо зв'язок. Допомагає та створює зв'язок між таблицями. В таблиці може бути декілька **FOREIGN KEY**. Ключ також може посилатись на саму себе таблицю

Student

PRIMARY KEY FOREIGN KEY

student_id	name	major	group_id
1	Kate	Sociology	1
2	Jack	Biology	2
3	Claire	English	2
4	Jack	Biology	1

Group

PRIMARY KEY

group_id	name	date_created
1	DMS 125	2019-05-05
2	DMS 200	2020-05-07

Приклад використання два Foreign Key (Композитних ключі)для визначення скільки продукту продав працівник клієнту. Ствобці **emloyee_id**/**client_id**, посилаються на PRIMARY KEY таблиць **employee**/**client**

Два зовнішніх ключі цієї таблиці створюють ніби один первинний ключ.

Work_Whith

PRIMARY KEY

emp_id	client_id	total_sales
1	300	50,000
2	301	40,000
1	302	45,000

MAINS DATA TYPES IN PostgreSQL

BIGSERIAL/SERIAL - - For id autoincrement
INT/FLOAT - - Whole Numbers
DECIMAL(10, 2) - - Decimal Numbers (55.85)
VARCHAR(9)/TEXT/CHAR - - String of text of length 9
BINARY/BLOB - - Binary Large Object(img…)
DATE/TIME - - 'YYYY-MM-DD'
TIMESTAMP - - 'YYYY-MM-DD HH:MM:SS'

```
#CREATE DATABASE

-- Створюємо базу даних

CREATE DATABASE nesik;


-- Створюємо/моделюємо таблиці в нашій базі даних з необхідними атрибутами(колонками) з різними типами даних

CREATE TABLE student (
# визначаємо назви стовпців з типом даних
     student_id INT,
     name VARCHAR(20),
     major VARCHAR(20),
     PRIMARY KEY(student_id)
);



#DESCRIBE table

-- Отримати дані про стовпці та їх типи даних таблиці

DESCRIBE student;



#DROP table

-- Щоб видалити таблицю або базу даних

DROP student/nesik;



#ADD COLUMN INTO TABLE

-- Щоб добавити нову/додаткову колонку в таблицю

ALTER TABLE student ADD avg_mark DECIMAL(3, 2);
```

```
#CHANGE COLUMN INTO TABLE

-- Щоб змінити наприклад назву колонки в таблиці
ALTER TABLE student
CHANGE COLUMN avg_mark mark DECIMAL(3, 2);

#DROP COLUMN FROM TABLE

-- Щоб видалити колонку з таблиці

ALTER TABLE student DROP COLUMN avg_mark;

#INSERTING DATA INTO table

-- Вставляємо дані в нашу створену таблицю student в тому порядку та з тим типом даних, які в ній стовпці

INSERT INTO student VALUES (1, 'Yevhen', 'Biology');

# Якщо ми не поставили AI (Auto_increment) для id, тоді його потрібно вказувати руками при додаванні VALUE. Два однакових PRIMARY KEY в таблицю вставити не можемо, так як його значення мають бути унікальними.

-- Якщо ми хочемо вставити дані лише в одну колонку таблиці, то нам потрібно вказати назву колонок, а тоді value

INSERT INTO student(student_id, name) VALUES (4, 'Mira');

DROP TABLE student;
```

Тепер створимо таку саму, але більш автоматизовану та інформативну таблицю

```
CREATE TABLE student (
    student_id INT AUTO_INCREMENT, # автоматично student_id
     створюватиметься та добавляє +1 при добавленні запису в таблицю
    name VARCHAR(20) NOT NULL, # не може бути пустим
    major VARCHAR(20) UNIQUE,    # значення, що не повторюються
    # Або ми можемо встановити Default Value для цього стовпця, якщо не було добавлено до нього значення
    major VARCHAR(20) DEFAULT 'undecided'
    PRIMARY KEY(student_id)

);

INSERT INTO student VALUE('Mira', 'Psychology');
```

 # MAINS DATA TYPES IN SQL

```
INT                          - - Whole Numbers
DECIMAL(10, 2)               - - Decimal Numbers (55.85)
VARCHAR(9)                   - - String of text of length 9
BLOB                         - - Binary Large Object(img …)
DATE                         - - 'YYYY-MM-DD'
TIMESTAMP                    - - 'YYYY-MM-DD HH:MM:SS'
```

UPDATE & DELETE - оновлення та видалення рядків в таблиці. Використовуючи оператор **WHERE** ми можемо вказати який конкретний рядок або інформацію в стовпці ми хочемо оновити або видалити.

```
-- Зміна даних рядку в таблиці. Змінити в усіх студентів в стовпці major в яких спеціальність 'Biology' на 'Bio'

UPDATE student SET major = 'Bio'
WHERE major = 'Biology';

-- Зміна даних рядку, в певній колонці таблиці та лише певного студента.

UPDATE student SET name = 'Bill, major = 'Bio''
WHERE student_id = 1;


-- Видалити всі дані/рядки з таблиці

DELETE FROM student;


-- Видалити лише дані(рядок) з таблиці певного студента

DELETE FROM student
WHERE student_id = 1;

# Якщо стовпець name може бути NULL, то
DELETE name FROM student
WHERE student_id = 1;
```

FOREIGN KEY

– - FOREIGN KEY ми можемо створити одразу при створенні нової таблиці за умови, якщо в нас є вже створена таблиця на яку ми можемо посилатись щоб їх з’єднати. Якщо таблиці такої не має, то FOREIGN KEY ми добавляємо пізніше, змінюємо вже існуючий стовпець в таблиці або добавляємо.

– - 1 варіант змінюємо стовпець в вже існуючій таблиці

```
ALTER TABLE employee ADD FOREIGN KEY(branch_id) REFERENCES
branch(branch_id) ON DELETE SET NULL;
```

– - 2 варіант створюємо його при створенні таблиці

```
CREATE TABLE employee (
      employee_id INT PRIMARY KEY AUTO INCREMENT,
     name VARCHAR(50),
     salary
     branch_id INT,
     FOREIGN KEY(branch_id)REFERENCES branch(branch_id)
     ON DELETE SET NULL

);
```

SELECT

```
# Вказуємо з якою базою даних працюємо, або якщо працюємо через MySQL Workbench, то просто заходимо в цю базу даних
USE sql_store


# Щоб завантажити дані з усієї таблиці
SELECT * FROM customers


# Отримати дані певного стовпця з таблиці
SELECT first_name, second_name FROM customers


# Можемо одразу виконувати арифметичні операції з стовпцями, якщо їх тип даних числовий
SELECT
     first_name,
     second_name,
    (salary + 10) * 20 AS `discount price`
FROM customers


# Щоб отримати наприклад дані певного стовпця, але без дублікатів значень, то використовуємо DISTINCT
SELECT DISTINCT state FROM customer
```

WHERE - використовуємо для фільтрації даних. Оператори, які ми можемо використовувати.(**>,<,>=,<=,=,**(**!=** or **<>)**)

```
# Наприклад нам потрібні усі клієнти з балансом > 1000
SELECT * FROM customer WHERE balance > 1000

# Наприклад нам потрібні усі клієнти, які живуть New York
SELECT * FROM customer WHERE city = 'New York'

# Наприклад нам потрібні усі клієнти, які не живуть Lviv
SELECT * FROM customer WHERE city != (<>) 'Lviv'
```

AND/OR/NOT - оператори, які використовуємо для фільтрації даних за декількома умовами.

```
# Наприклад ми хочемо отримати усіх клієнтів, які народилися після 1990.01.01 а також мають баланс > 1000
SELECT * FROM customer
WHERE date_birth > '1990.01.01' AND balance > 1000
```

IN operator - застосовуємо до рядків з рядковим типом

```
SELECT * FROM customer
WHERE place_birth IN ('UK', 'PL', 'UA')

SELECT * FROM customer
WHERE place_birth NOT IN ('UK', 'PL', 'UA')
```

BETWEEN operator

```
SELECT * FROM customer
WHERE salary >= 100 AND salary <= 3000

# Краще використати оператор BETWEEN, коли ми порівнюємо та шукаємо певний діапазон значень від -> до (включно).

SELECT * FROM customer
WHERE salary BETWEEN 1000 AND 3000

SELECT * FROM customer
WHERE birth_day BETWEEN '1990-01-01' AND '2000-01-01'
```

LIKE operator

```
# Наприклад нам потрібно отримати усіх клієнтів в яких ім'я починається на букву М (Також можна використовувати декілька букв одразу для пошуку, наприклад Мар%% … ) не враховуючи довжини ім'я і т.д.

SELECT * FROM customer
WHERE name LIKE 'M%'

# Клієнтів в яких ім'я закінчуєтьсяться на букву М

SELECT * FROM customer
WHERE name LIKE '%M'

# Клієнтів в яких в ім'ї є буква а

SELECT * FROM customer
WHERE name LIKE '%a%'

# Отримуємо клієнта який складається рівно з 2-ох символів, але в якого останній символ мусить бути - у
SELECT * FROM customer
WHERE name LIKE '_y'
```

```
# Зробивши 5 _ і в кінці y, ми говоримо, що нам потрібні лише клієнти з ім'ям в 6 букв, остання буква закін. на Y
SELECT * FROM customer
WHERE name LIKE '_____y'


SELECT * FROM customer
WHERE name LIKE 'B____y'


# Щоб отримати всі номера телефонів, окрім тих, які закінчуються 11
SELECT * FROM customer
WHERE phone_number NOT LIKE '%9'
```

The REGEXP operator - regular expressions, потужний оператор для роботи з даними типу рядки (CHARFIELD, TEXTFIELD і т.д.) для вибору рядкових полів.

-- ^ Beginning

-- $ End

-- | Logical OR

-- '[rgt]e' -> 're', 'ge', 'te'

-- 'e[a-f]' -> 'ea', 'eb', 'ec' …

Наприклад нам потрібні рядки, де в прізвищі клієнта міститься слово 'field'.

```
SELECT * FROM customer
WHERE last_name REGEXP 'field'
```

В **REGEXP** операторі, нам доступно багато знаків(символів) дії роботи з пошуком рядків, аніж в операторі **LIKE**

^ - використовуємо цей знак для позначення початку рядка. '^field' - вибрати слова, які починається field.

```
SELECT * FROM customer
WHERE last_name REGEXP '^field'
```

$ - використовуємо цей знак для позначення кінця рядка. 'field$' - вибрати слова, які закінчуються на field.

```
SELECT * FROM customer
WHERE last_name REGEXP 'field$'
```

| - використовуємо цей знак для OR expression, для представлення декількох шаблонів пошуку. Наприклад нам потрібно вибрати усі прізвища, які починаються на 'field' та закінчуються на 'mac'

```
SELECT * FROM customer
WHERE last_name REGEXP '^field|mac$'
```

[] - наприклад, коли нам потрібно шукати якесь слово, яке закінчується на 'e', а перед цією буквою мусить бути будь яка буква з цього набору ['g', 'i', 'm']. Тобто слова, які закінчуються на 'ge', 'ie', 'me'.

```
SELECT * FROM customer
WHERE last_name REGEXP '[gim]e$'
```

Можна шукати і по параметрам, коли всі букви з набору, стоять після букви 'e'

```
SELECT * FROM customer
WHERE last_name REGEXP 'e[gim]'
```

Також ми можемо надати діапазон символів, які мають бути після цієї букви. Наприклад від букви a до f. [a-f]

```
SELECT * FROM customer
WHERE last_name REGEXP 'e[a-f]'
```

The IS NULL operator - **IS NULL** використовуємо для пошуку полів без значень(порожніх полів/атрибутів).

```
-- Вибрати всіх клієнтів в яких немає номеру телефону

SELECT *
FROM customer
WHERE phone_number IS NULL

-- Вибрати всіх клієнтів в яких є дані номеру телефону

SELECT *
FROM customer
WHERE phone_number IS NOT NULL
```

The ORDER BY Clause - **ORDER BY** використовуємо для сортування запитуваних даних

За дефолтом, сортування отримання даних відбувається по **id** (**SELECT * FROM** customer) в порядку зростання. Так як id це за замовчуванням Primary Key кожної створеної нами таблиці. (Унікальний(без дублікатів)ідентифікатор кожного поля. Клієнт(id 1), Клієнт(id 2) і т.д.)

```
SELECT *, quantity * unit_price AS total_price
FROM order_items
WHERE id = 2
ORDER BY quantity * unit_price DESC(по спаданню)
```

```
- - При сортуванні даних важливий порядок сортування

SELECT first_name, birth_day
FROM customer
ORDER BY birth_day DESC(1), first_name (2)
```

The LIMIT Clause - обмеження записів (полів), які повертаються з бази даних при нашому запиті.

```
-- Коли ми хочемо повернути лише 3 - ох клієнтів

SELECT *
FROM customer
LIMIT 3
```

```
-- Коли на сторінках розміщується різна кількість продуктів, а ми хочемо лише отримати з другої сторінки 2 перших продукти
  - page 1: 1 - 10
  - page 2: 11 - 21

SELECT *
FROM product
LIMIT 10, 2 (пропустити 10 продуктів і взяти наступних 2)
```

```
-- Отримати ТОП 3 найбільш лояльних клієнтів (в яких оцінка стосовно обслуговування найвища)

SELECT *
FROM customer
ORDER BY rating DESC
LIMIT 3
```

```
! Важливий порядок застосування операторів !
LIMIT завжди в кінці.

SELECT -> FROM -> WHERE -> ORDER BY -> LIMIT
```

ФУНКЦІЇ SQL

```
- - Порахувати всіх працівників в нашій таблиці

SELECT COUNT(employee_id)
FROM employee;


- - Порахувати всіх працівників жінок, які народились
після 2000 р

SELECT COUNT(employee_id)
FROM employee
WHERE male = 'Ж' AND date_of_birth > '2000-01-01';


- - Порахувати середню зарплату всіх працівників

SELECT AVG(salary)
FROM employee;


- - Порахувати загальну суму зарплати всіх працівників

SELECT SUM(salary)
FROM employee;


- - Порахувати скільки жінок та чоловіків є в компанії

SELECT SUM(male), male
FROM employee
GROUP BY male;
```

Inner Joins - для вибору та отримання даних одразу з декількох зв'язаних таблиць (по певних полях).

— - Inner Join, об’єднання двох таблиць order з customer, на основі id_customer (кожне замовлення має customer_id - інформацію про клієнта, а кожен клієнт має свій унік. id)

```
SELECT first_name,
FROM order
JOIN customer
# Умова по якій робимо об’єднання двох таблиць
ON order.costomer_id = customer.customer_id

# Зліва виводяться дані таблиці order, а праворуч customer (таблиці яку приєднуємо)

# Якщо ми робимо об’єднання, то при SELECT нам необхідно вказувати спершу назву таблиці, а тоді назву стовпців. Так як назви стовпців двох таблиць можуть бути однакові.
```

– - Можемо писати назви таблиць скорочено

```
SELECT o.order_id c.first_name, c.phone_number
FROM order o
JOIN customer c
# Умова по якій робимо об’єднання двох таблиць
ON o.costomer_id = c.customer_id
```

Об'єднання таблиць з різних баз даних

– - До прикладу таблиця order_item знаходиться в одній базі даних, а таблиця custumer в іншій, щоб їх об’єднати.

```
SELECT o.order_id c.first_name, c.phone_number
FROM order_item oi (потрібно знаходитись в цій таблиці)
JOIN sql_name.customer c
ON oi.product_id = c.product_id
```

SELF JOIN - об’єднання таблиці за допомогою себе.

– - Уявімо, що в нас є таблиця працівників, і в ній також є 2 менеджери. Кожен працівник має свого менеджера (колонка з id менеджера). Нам потрібно додати колонку та знайти менеджера який відповідальний кожного працівника.

```
USE sql_hr;

SELECT
     e.employee_id,
     e.first_name,
     m.first_name AS manager
FROM employees e
# З'єднуємо таблицю з собою. Але даємо інший псевдонім.
JOIN employees m
ON e.reports_to = m.employee_id
```

Joining Multiple Tables - об'єднання даних декількох таблиць при написанні запиту до бази даних.

```
- - В таблиці order є посилання (зв'язок) на таблицю (customer -> id customer, який зробив замовлення та на status замовлення (1 - finish, 2 - in process)

USE sql_store;

SELECT
     o.order_id,
     o.order_date,
    c.first_name,
    c.last_name,
    os.name AS status

FROM orders o
JOIN customer c
     ON o.customer_id = c.customer_id
JOIN order_statuses os
     ON o.status = os.order_status_id
```

Складна умова для об'єднання 2-ох таблиць, коли є декілька умов для їх з'єднання.

```
SELECT *
FROM order_items oi
JOIN order_item_notes oin
     ON oi.order_id = oin.order_id
    AND oi.customer_id = oin.customer_id
```

Неявний синтаксис об'єднання в MySql -рекоменд. не юзати

```
- - Стандартний синтаксис

SELECT *
FROM orders o
JOIN customers c
     ON o.customer_id = c.customer_id


- - Неявний/додатковий синтаксис


SELECT *
FROM orders o, customers c
WHERE o.customer_id = c.customer_id
```

Outer Joins - в sql ми маємо 2 типи з'єднань. **INNER JOINS** and **OUTER JOINS** / внутрішні та зовнішні з'єднання

-- При використанні INNER JOIN (JOIN) ми отримаємо лише результат які відповідають умові. Тобто ми отримаємо лише тих користувачів, які зробили замовлення

```
SELECT
     c.customer_id,
     c.first_name,
     o.order_id
FROM customers c
JOIN orders o
     ON o.customer_id = c.customer_id
ORDER BY c.customer_id
```

-- Використовуючи OUTER JOIN (LEFT OR RIGHT). Ми отримаємо не лише клієнтів, які відповідають умові та зробили замовлення, але і також усіх клієнтів та інформацію (усі записи в лівій таблиці), які навіть не відповідають нашій умові, але є в лівій таблиці customer.

Навпроти клієнтів, які зробили замовлення order_id буде номер замовлення, навпроти клієнтів без замовлення order_id буде NULL.

```
SELECT
     c.customer_id,
     c.first_name,
     o.order_id
FROM customers c
LEFT JOIN orders o
     ON o.customer_id = c.customer_id
ORDER BY c.customer_id
```

Outer Joins Between Multiple Tables

```
SELECT
     c.customer_id,
     c.first_name,
     o.order_id
     sh.name AS shipper
FROM customers c
LEFT JOIN orders o
     ON o.customer_id = c.customer_id
LEFT JOIN shippers sh
     ON o.shipper_id = sh.shipper_id
ORDER BY c.customer_id
```

Self Outer Joins

```
USE sql_hr;

SELECT
     e.employee_id,
     e.first_name,
     m.first_name AS manager
FROM employees e
LEFT JOIN employees m
ON e.reports_to = m.employee_id
```

The USING Clause - функція для спрощення запитів.

```
SELECT
     c.customer_id,
     c.first_name,
     o.order_id
FROM customers c
JOIN orders o
     # Ми можемо замінити це за умови, що однакова
     назва стовпців в 2-ох таблицях
     - - ON o.customer_id = c.customer_id
     USING (customer_id)
LEFT JOIN shippers s
     USING (shippers_id)
```

The NATURAL JOIN - автоматичне з'єднання двох таблиць. Не рекомендується робити, бо інколи дає бед результати.

```
SELECT
     c.customer_id,
     c.first_name,
     o.order_id
FROM customers c
NATURAL JOIN orders oS
```

The CROSS JOIN - перехресне з'єднання, 1 рядка з лівої таблиці, з кожним записом з правої таблиці, і так всі наявні рядки лівої таблиці.

```
SELECT
     c.customer_name,
     p.name AS product_name
FROM customers c
CROSS JOIN product p
```

UNIONS - з’єднання/комбінування рядків з різних таблиць / запитів(на відміну від join, який з'єднує стовпцями). Об’єднання має здійснюватись між запитами, які повертають однакову кількість стовпців.

```
SELECT
     order_id,
     order_date,
     ‘Active’ AS status
FROM orders o
WHERE order_date >= ‘2019-01-01’;

UNION

SELECT
     order_id,
     order_date,
     ‘Archived’ AS status
FROM orders o
WHERE order_date <= ‘2019-01-01’
```

Connecting MySQL in Python

```
$ pip install mysql-connector
```

```
import mysql.connector
```