Data & Applications
Tutorial-4

BASICS OF SQL

SQL (Structured Query Language)

- SQL is a language used to create, manage, and retrieve data from relational databases.
- Relational databases organize data in tables made up of rows and columns. **(Examples: MySQL, PostgreSQL, etc.)**
- SQL is the most common language used to store and manage data in relational databases.
- It is not case-sensitive — for example, **SELECT * FROM STUDENTS;** and **select * from students;** mean the same.(However, table and column names might be case-sensitive in some systems.)

DATABASE LEVEL COMMANDS

Creating a Database:

CREATE DATABASE database_name;

Viewing All Databases:

SHOW DATABASES;

Using a Database:

USE database_name;

Deleting a Database:

DROP DATABASE database_name;

TABLE LEVEL COMMANDS

- A table represents a collection of related data, also called a **relation**.
- Each row is a **tuple**, and each column is an **attribute**.
- The number of columns = **Degree** of the table.
- The number of unique rows = **Cardinality** of the table.
- CREATE TABLE table_name (
  column1 datatype [constraints],
  column2 datatype [constraints],
  ...
  PRIMARY KEY (one_or_more_columns)
  );
- Common data types include CHAR, VARCHAR, INT, DECIMAL etc
- Common constraints include:
  NOT NULL → prevents empty values
  UNIQUE → no duplicate values allowed

TABLE LEVEL COMMANDS

Describing a table: Viewing the table's metadata

DESC <TABLE NAME>;

Eg: DESC CUSTOMERS;

desc or describe command shows the structure of the table which include the name of the column, the data type of the column and the nullability which means, that column can contain null values or not.

```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(50) | YES  |     | NULL    |       |
| email | varchar(100)| YES  |     | NULL    |       |
| age   | int(11)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Deleting a table

DROP TABLE <TABLE_NAME>

INSERTING DATA INTO A TABLE

Use the **INSERT INTO** command to add new rows to a table.

Explicit Insertion(safest and most flexible)-You explicitly name the columns you are inserting into, so the order doesn't matter

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

Implicit Insertion-(Order-Based)-You provide a value for every column in the table, in the exact order they were defined when the table was created.

INSERT INTO table_name VALUES (value1, value2, value3, ...);

NOTE: To avoid manually figuring out the next ID for each insertion, you can set the primary key to generate its own value. **Use AUTO_INCREMENT (in MySQL)** or a similar feature. eg. **CREATE TABLE CUSTOMERS (ID INT AUTO INCREMENT PRIMARY KEY , ...)** Now, when you insert a new customer, you can just omit the ID column, and the database will handle it.

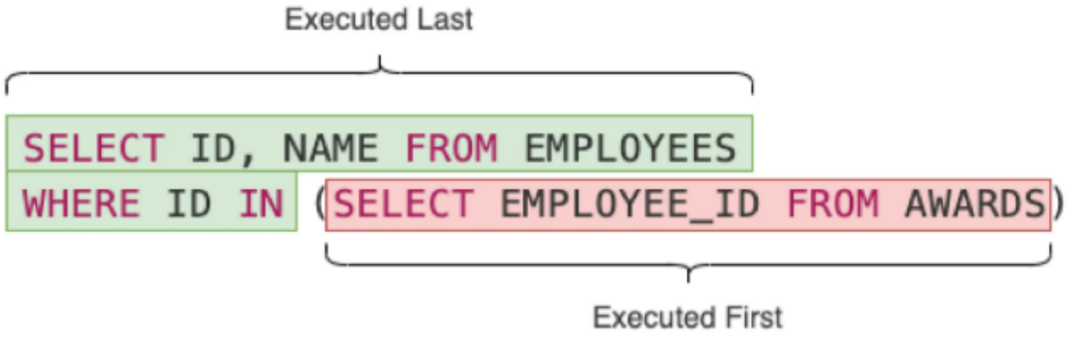
MODIFYING AN EXISTING TABLE STRUCTURE

Use the **ALTER TABLE** command to change the definition (or schema") of a table that already exists.

- **Add a Column**
  - **Syntax**-ALTER TABLE table_name ADD column_name datatype;
  - **Example**-ALTER TABLE CUSTOMERS ADD Phone VARCHAR(20);
- **Drop (Delete) a Column**
  - **Syntax**-ALTER TABLE table_name DROP COLUMN column_name;
  - **Example**-ALTER TABLE CUSTOMERS DROP COLUMN Email;
- **Modify a Column-**changes the properties of an existing column, such as its data type or size.
  - **Syntax**-ALTER TABLE table_name MODIFY COLUMN column_name new_datatype;
  - **Example**-ALTER TABLE CUSTOMERS MODIFY COLUMN Email VARCHAR(511);
    (initially Email column was varchar(255))

NESTED QUERIES

A nested query in SQL contains a query inside another query. The outer query will use the result of the inner query. For instance, a nested query can have two SELECT statements, one on the inner query and the other on the outer query.



Types of Nested Queries

1. Independent (Non-Correlated) Subquery

- Executed once for the entire main query.
- Result of subquery is used by the outer query.
- The subquery does not depend on values from the outer query.

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

NESTED QUERIES

2. Correlated Subquery

- Executed for each row of the outer query.
- The subquery depends on values from the outer query.
- Usually slower than independent subqueries.

```
SELECT e1.name, e1.salary
  FROM employees e1
 WHERE e1.salary > (
   SELECT AVG(e2.salary)
     FROM employees e2
  WHERE e2.department_id =
  e1.department_id
          );
```

SET OPERATIONS IN SQL

- A relation is defined as a set of tuples. Hence, we can apply all standard set theory operations on relations.
- Several set theoretic operations are used to merge the elements of two sets in various ways, including UNION, INTERSECTION, and SET DIFFERENCE (also called MINUS or EXCEPT).
- These are binary operations; that is, each is applied to two sets.
- The two relations on which any of these three operations are applied must have the same type of tuples; this condition has been called union compatibility

SET OPERATIONS IN SQL

1. **UNION**

- Combines results of both queries
- Removes duplicates by default

```
SELECT name FROM students_2024
UNION
SELECT name FROM students_2025;
```

Returns all unique student names from both tables.

2. **UNION ALL**

- Similar to UNION
- Keeps duplicates

```
SELECT name FROM students_2024
UNION ALL
SELECT name FROM students_2025;
```

Returns all names, including duplicates.

3. **INTERSECT**

- Returns common rows from both queries

```
SELECT name FROM math_class
INTERSECT
SELECT name FROM science_class;
```

Students enrolled in both classes.

4. **MINUS / EXCEPT**

- Returns rows from the first query not in the second

```
SELECT name FROM math_class
EXCEPT
SELECT name FROM science_class;
```

Students in Math but not in Science.

SUMMARIZING DATA (AGGREGATIONS)

- Aggregate functions perform a calculation on a set of rows to return a single, summary value.They are often used to get a "big picture" view of your data.
- Common Aggregate Functions
  - COUNT( ): Counts the number of rows.
  - SUM( ): Adds up all the values in a column.
  - AVG( ): Calculates the average of a column's values.
  - MAX( ): Finds the highest value in a column.
  - MIN( ): Finds the lowest value in a column.
- Example-Find the total number of employees and the average salary for the entire company
  - SELECT

    COUNT(*) AS TotalEmployees,

    AVG(Salary) AS AverageSalary

    FROM

    EMPLOYEES;

GROUP BY & HAVING CLAUSE

- We use GROUP BY to run aggregate functions on sub-groups of data, not just the whole table.
- **GROUP BY Clause**
  - Groups rows that have the same values in a specified column.
  - Used with aggregate functions (like COUNT, AVG, SUM) to calculate a summary for each group.
  - example-Count employees in each department.
- **HAVING Clause**
  - Filters the groups after they have been aggregated.
  - WHERE filters rows before grouping; HAVING filters groups after grouping.
  - example-Show only departments with more than one employee.

```sql
SELECT
    Department,
    COUNT(*) AS NumberOfEmployees
FROM
    EMPLOYEES
GROUP BY
    Department;
```

```sql
SELECT
    Department,
    COUNT(*) AS NumberOfEmployees
FROM
    EMPLOYEES
GROUP BY
    Department
HAVING
    COUNT(*) > 1;
```

IF-THEN-ELSE LOGIC (THE CASE STATEMENT)

- The **CASE** statement is SQL's way of handling **if-then-else** logic. It allows you to create new, conditional values in your query results.
- Think of it as a way to create a **new temporary column** where the value for each row is determined by a set of rules.
- The CASE statement goes through a list of conditions for each row. The moment it finds a condition that is true, it returns the specified result and stops checking
  - **WHEN ... THEN ...:** This is your "if-then" condition.
  - **ELSE ...**: This is the optional "fallback" or default value if no WHEN conditions are met.
  - **END**: This is mandatory and signals the end of the CASE block.
  - **AS:** It's best practice to give this new conditional column a name using an alias.

Syntax

```
SELECT
    column1,
    column2,
    CASE
        WHEN condition1 THEN result1
        WHEN condition2 THEN result2
        ...
        ELSE default_result
    END AS new_column_name
FROM
    table_name;
```

EXAMPLE-CATEGORIZING EMPLOYEE SALARIES

You have an EMPLOYEES table that lists employees and their annual salaries.
Write a query to categorize each employee's salary into one of three bands: 'High', 'Medium', or 'Low'.

- High: Salary is greater than $80,000
- Medium: Salary is between $50,001 and $80,000
- Low: Salary is $50,000 or less

EmployeeID	Name	Salary
101	Sarah	95000
102	Mark	62000
103	Jane	50000
104	Tom	80000

SOLUTION

```sql
SELECT
    Name,
    Salary,
    CASE
        WHEN Salary > 80000 THEN 'High'
        WHEN Salary > 50000 THEN 'Medium'
        ELSE 'Low'
    END AS SalaryBand
FROM
    EMPLOYEES;
```

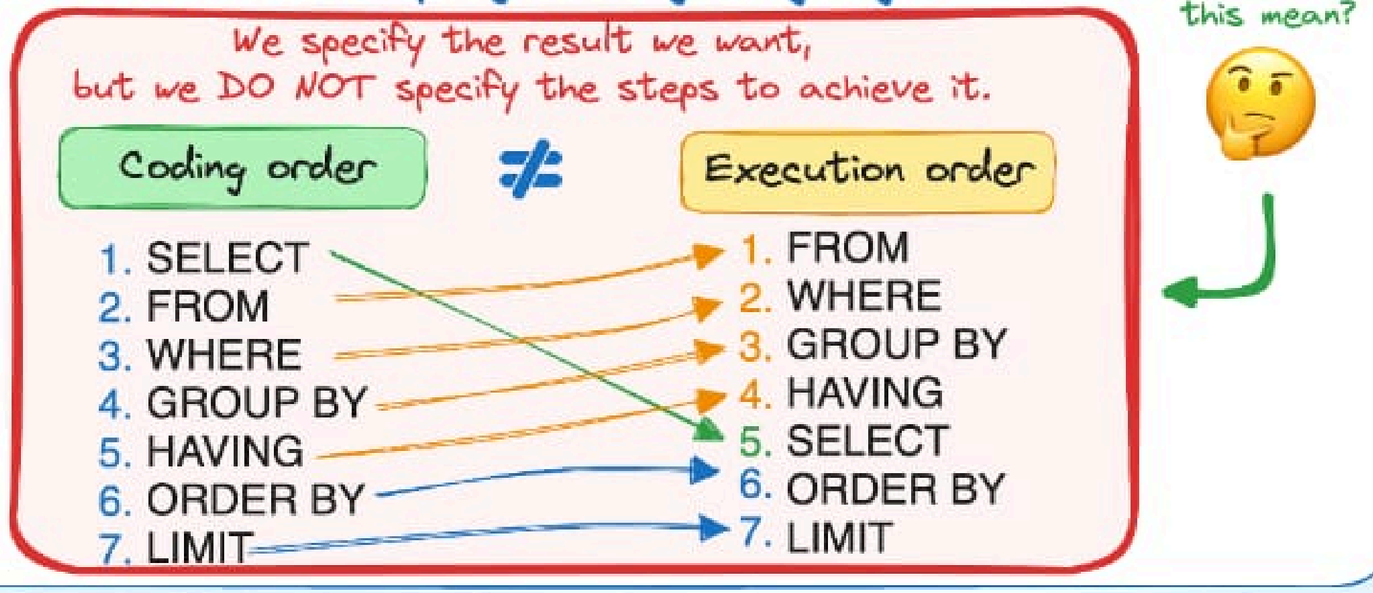
Name	Salary	SalaryBand
Sarah	95000	High
Mark	62000	Medium
Jane	50000	Low
Tom	80000	Medium

databites.tech @rfeers Josep Ferrer

SQL BITE - SQL's EXECUTION ORDER WITH JOIN

#1 SQL AS A DECLARATIVE LANGUAGE

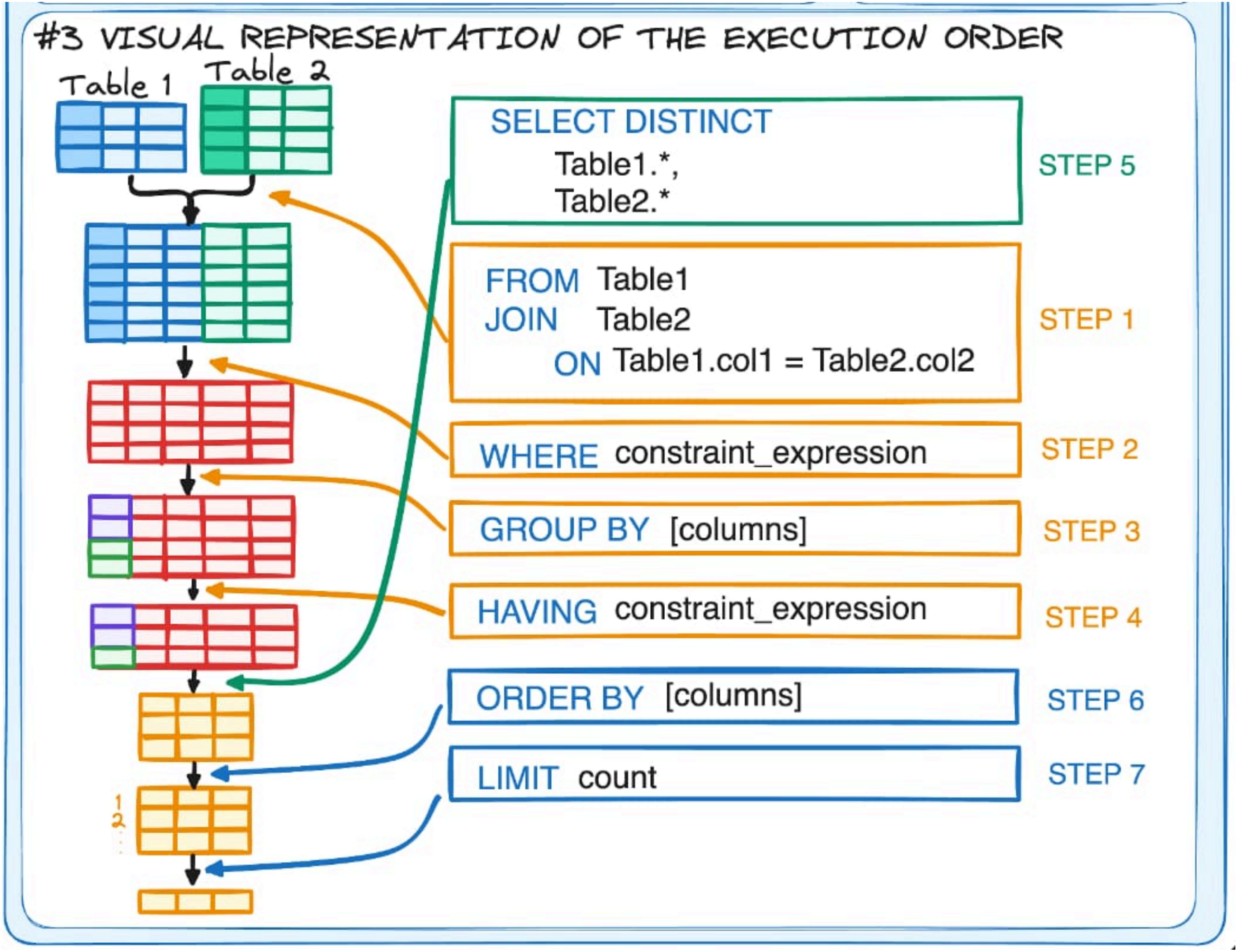
SQL is a declarative programming language. → What does this mean?



#2 SQL QUERY

The most common query has a structure like follows:

```
SELECT DISTINCT
    Table1.*,
    Table2.*
FROM Table1
JOIN Table2
    ON matching_condition
WHERE constraint_expression
GROUP BY [columns]
HAVING constraint_expression
ORDER BY [columns]
LIMIT count
```


#3 VISUAL REPRESENTATION OF THE EXECUTION ORDER
Table 1
Table 2
SELECT DISTINCT
Table1.*,
Table2.*
STEP 5
FROM Table1
JOIN Table2
ON Table1.col1 = Table2.col2
STEP 1
WHERE constraint_expression
STEP 2
GROUP BY [columns]
STEP 3
HAVING constraint_expression
STEP 4
ORDER BY [columns]
STEP 6
LIMIT count
STEP 7

SOME MORE EXAMPLES

- You have the Customers and Orders tables. Write a query to find the names of any customers who have placed more than one order, and also show how many orders they placed.

CustomerID	CustomerName	Country
1	Alice Smith	USA
2	Ben Lee	Canada
3	Clara Jones	USA
4	David Kim	UK

OrderID	CustomerID	OrderTotal
1001	2	250
1002	1	1200
1003	3	400
1004	2	300

SOLUTION

```sql
SELECT
    c.CustomerName,
    COUNT(o.OrderID) AS NumberOfOrders
FROM
    Customers c
JOIN
    Orders o ON c.CustomerID = o.CustomerID
GROUP BY
    c.CustomerName
HAVING
    COUNT(o.OrderID) > 1;
```

```
CustomerName    NumberOfOrders
-------------------------------
Ben Lee         2
```

SOME MORE EXAMPLES

- You have the Employees and Departments tables. Using a nested query, write a query to find the names of all employees who work in the 'Sales' department.

DepartmentID	DepartmentName
10	Sales
20	Engineering
30	Marketing
40	HR

EmployeeID	Name	DepartmentID
1	Anna	10
2	Bill	20
3	Chris	30
4	Dana	10
5	Evan	40

SOLUTION

```sql
SELECT
    Name
FROM
    Employees
WHERE
    DepartmentID IN (SELECT DepartmentID
                     FROM Departments
                     WHERE DepartmentName = 'Sales');
```

```
Name
-------
Anna
Dana
```

THANK YOU