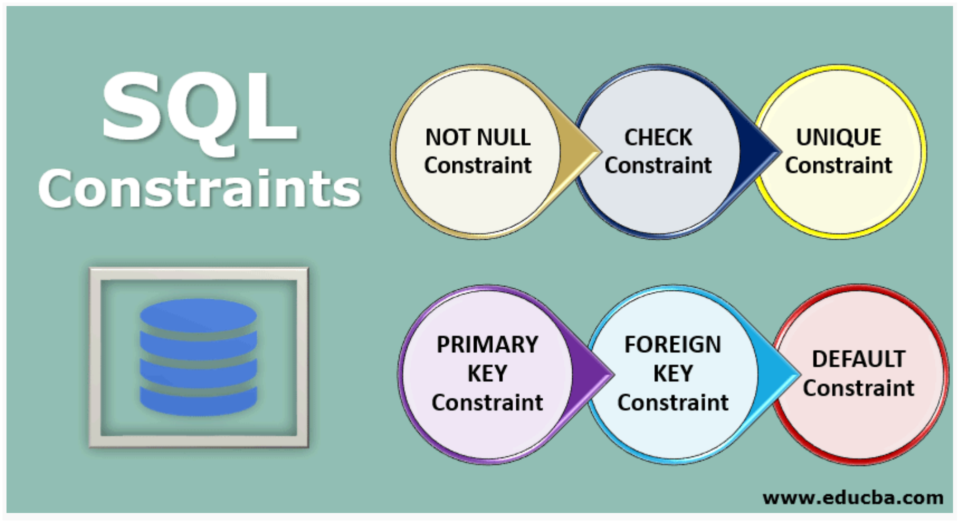
SQL basic

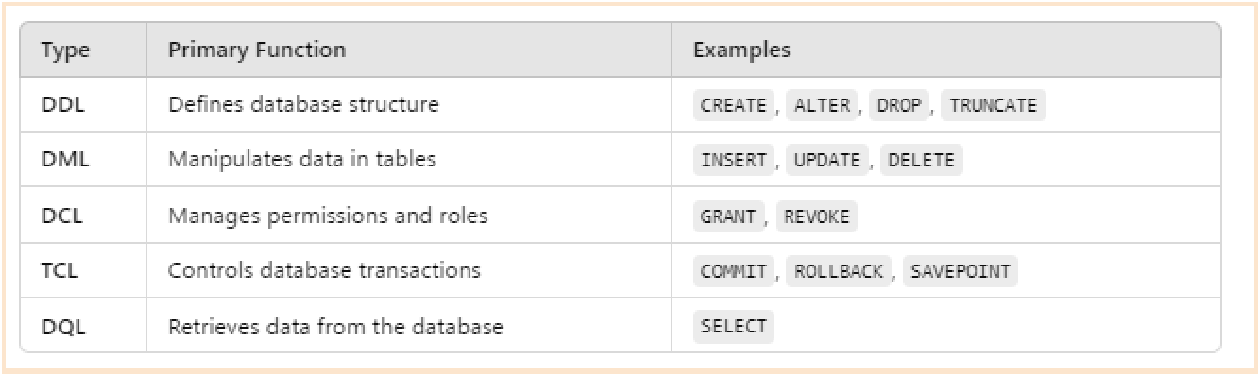
SQL basic Outlines

- **Create Table (DDL)**
- **Drop (Delete) Table (DDL)**
- **ALTER Table (DDL)**
- **DELETE Command (DDL)**
- **TRUNCATE Command (DDL)**
- **UPDATE Command (DML)**
- **Comparison Conditions**
    - =, > , >=, < , <=
    - BETWEEN ... AND …
    - IN (Set)
    - LIKE
- **Logical Conditions**
    - AND
    - OR
    - NOT
- **Arithmetic Expressions on columns numeric data**
- **ORDER BY Clause (ASC, DESC)**
- **GROUP BY with HAVING Clause**

DB_part1.sql Link

https://drive.google.com/file/d/13p9uKbRe7OLAC2Zrm7TN_uRIWqV5-_mh/view?usp=sharing





Type	Primary Function	Examples
DDL	Defines database structure	`CREATE`, `ALTER`, `DROP`, `TRUNCATE`
DML	Manipulates data in tables	`INSERT`, `UPDATE`, `DELETE`
DCL	Manages permissions and roles	`GRANT`, `REVOKE`
TCL	Controls database transactions	`COMMIT`, `ROLLBACK`, `SAVEPOINT`
DQL	Retrieves data from the database	`SELECT`

Drop (Delete) Table (DDL)

To drop the `users` table, you can use the following SQL command:

```
DROP TABLE [Table Name];
```

Eg:

```
DROP TABLE users;
```

ALTER Command (DDL)

The `ALTER` DDL command is used to modify the structure of an existing table.

Here are some examples of common `ALTER` operations for the `users` table:

- ➢ 1. Add a New Column (ADD word)

  Syntax:

  ```
  ALTER TABLE table_name

  ADD COLUMN column_name data_type constraints;
  ```

  Eg:

  ```
  ALTER TABLE users ADD COLUMN phone VARCHAR(15);

  ALTER TABLE users ADD COLUMN phone1 varchar(20) NOT null;
  ```

- ➢ 2. Modify an Existing Column

  Syntax:

  ```
  ALTER TABLE table_name MODIFY COLUMN column_name new_data_type
  constraints;
  ```

  Eg:

  ```
  ALTER TABLE users MODIFY COLUMN phone VARCHAR(15) DEFAULT 'Not Provided';
  ```

  Eg:

  ```
  ALTER TABLE users

  MODIFY COLUMN name VARCHAR(100) NOT NULL;
  ```

  Eg:

  ```
  ALTER TABLE users

  MODIFY COLUMN id INT AUTO_INCREMENT PRIMARY KEY;
  ```

➢ 3. Rename a Column

Syntax:

```
ALTER TABLE table_name

CHANGE COLUMN old_column_name new_column_name data_type constraints;
```

Eg:

```
ALTER TABLE users CHANGE COLUMN email user_email VARCHAR(255) NOT NULL;
```

➢ 4. Drop a Column

Syntax:

```
ALTER TABLE table_name

DROP COLUMN column_name;
```

Eg:

```
ALTER TABLE users DROP COLUMN phone;
```

➢ **5. Add a Constraint**

Syntax:

```
ALTER TABLE table_name

ADD CONSTRAINT constraint_name constraint_type(column_name);
```

Eg:

```
ALTER TABLE users

ADD CONSTRAINT unique_email UNIQUE(email);
```

➢ **6. Rename the Table**

Syntax:

```
ALTER TABLE table_name

RENAME TO new_table_name
```

Eg:

```
ALTER TABLE users RENAME TO customers;
```

UPDATE Command (DML)

The **`UPDATE`** command is used to modify existing records in a table.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Eg: Update a Single Column

```
UPDATE users
SET email = 'updated.email@example.com'
WHERE id = 3;
```

Eg: Update Multiple Column

```
UPDATE users
SET name = 'Alice Johnson', email = 'alice.johnson@example.com'
WHERE id = 4;
```

DELETE Command Examples

The **`DELETE`** command is used to remove rows from a table. Without resetting for any `AUTO_INCREMENT` counter

Syntax:

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

Eg1: Delete a Specific Row

```
DELETE FROM orders
WHERE order_id = 5;
```

Eg2: Delete All Rows (without Dropping Table)

```
DELETE FROM orders;
```

TRUNCATE Command

The `TRUNCATE` command is used to quickly remove all rows from a table while resetting any `AUTO_INCREMENT` counters.

```
TRUNCATE TABLE table_name;
```

- For example:
  - Deletes all rows from the `users` table and resets the `id` column's `AUTO_INCREMENT` counter.
  - `TRUNCATE TABLE orders;`

Key Differences: DELETE vs. TRUNCATE

Feature	DELETE	TRUNCATE
Deletes Specific Rows?	Yes (with `WHERE` clause).	No, deletes all rows.
Resets `AUTO_INCREMENT`?	No.	Yes.
Speed	Slower, logs individual row deletions.	Faster, no row-by-row logging.

Task Part 1: SQL Practice with ALTER and UPDATE Command

END AT 8:40 PM

ALTER TABLE

1. Add a new column `birth_date` of type `DATE` to the `employees` table.

2. Change the data type of the `order_total` column in the `orders` table to `DECIMAL(12,2)`.

3. Rename the column `job_title` in the `employees` table to `position_title` (`VARCHAR(255)`).

🔹 UPDATE

4. Increase the `order_total` in the `orders` table by 5% for all orders placed after `'2024-01-17'`.

```
UPDATE orders
SET order_total = order_total * 1.05
WHERE order_date > STR_TO_DATE('17-01-2024', '%d-%m-%Y');
```

🔹 DELETE

5. Delete all employees in the `employees` table who work in the `Support` department.

6. Delete all orders from the `orders` table where `order_total` is less than `200`.

7. “After completing the tasks, drop all tables and then import the SQL file again.”

Comparison Conditions Explanation

Here’s a detailed explanation of each comparison condition with a description, syntax, and examples:

1) =, > , >=, < , <=

Syntax:

```
column_name = value
```

Eg1: Retrieves all rows where the name is Ahmed Ali.

```
SELECT * FROM users WHERE name = 'Ahmed Ali';
```

Eg2: Retrieves employees earning more than 5000.

```
SELECT * FROM employees WHERE salary >= 5000;
```

2) BETWEEN ... AND ...

Description: Checks if a column value is within a specified range, inclusive of the boundary values.

Syntax:

```
column_name BETWEEN value1 AND value2
```

Eg1: Retrieves employees whose salaries are between 4000 and 6000, inclusive.

```
SELECT * FROM employees

WHERE salary BETWEEN 4000 AND 6000;
```

Eg2: Retrieves employees hired in the year 2024.

```
SELECT * FROM employees

WHERE hire_date

BETWEEN '2024-01-01' AND '2024-12-31';
```

3) IN (Set)

Description: Checks if a column value matches any value in a specified list.

Syntax:

```
column_name IN (value1, value2, ...)
```

Eg1: Retrieves employees who are either Software Engineer or Data Analyst.

```
SELECT * FROM employees

WHERE job_title IN ('Software Engineer', 'Data Analyst');
```

Eg2: Retrieves users with names Ahmed Ali, Layla Sami, or Omar Adel.

```
SELECT * FROM users

WHERE name IN ('Ahmed Ali', 'Layla Sami', 'Omar Adel');
```

4) LIKE

Description: Matches a value to a specified character **pattern using wildcards.**

Syntax:

```
column_name LIKE 'pattern'
```

Wildcards:

`%`: Matches any sequence of characters.

`_`: Matches exactly one character.

Eg1: Retrieves users whose names start with A.

```
SELECT * FROM users

WHERE name LIKE 'A%';
```

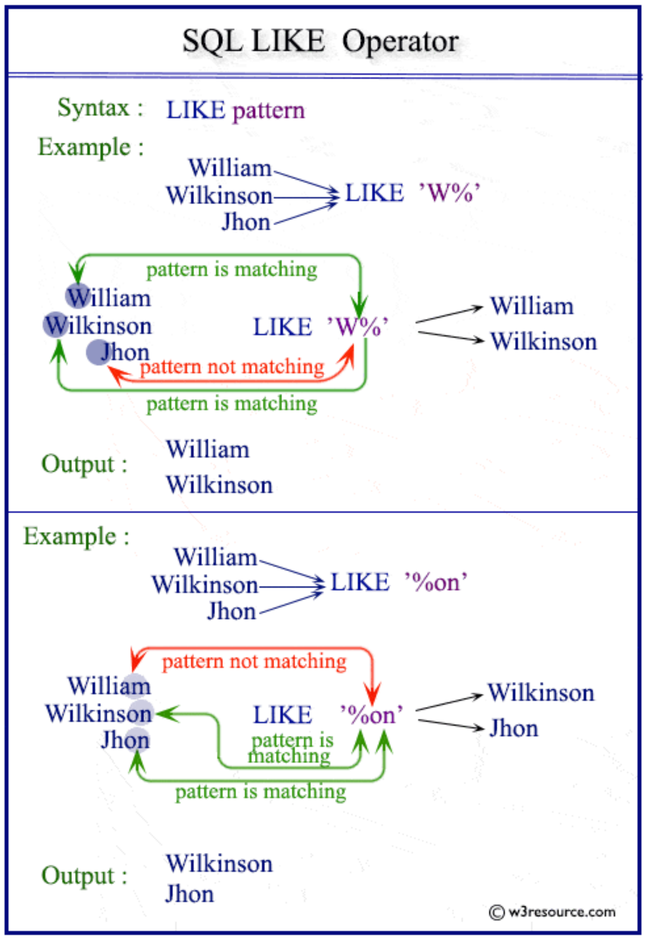
Eg2: Retrieves employees whose job title ends with Engineer.

```
SELECT * FROM employees

WHERE job_title LIKE '%Engineer';
```

Eg3: Find Users Whose Names Start with Any Single Character Followed by `hmed`

```
SELECT * FROM users
WHERE name LIKE '_hmed%';
```



Logical Conditions Explanation

Logical conditions in SQL are used to combine multiple conditions in a query. The most common logical operators are AND, OR, and NOT.

1. AND

- Description: Combines two or more conditions and returns rows only when all conditions are true.

  Syntax:

  ```
  condition1 AND condition2
  ```

  Eg1: Retrieves employees in the IT department with a salary greater than 5300.

  ```
  SELECT * FROM employees

  WHERE department = 'IT' AND salary > 5300;
  ```

  Eg2: Retrieves users whose names start with A **and** emails belong to @gmail.com.

  ```
  SELECT * FROM users

  WHERE name LIKE 'A%' AND email LIKE '%@example.com';
  ```

2. OR

- Description: Combines two or more conditions and returns rows when at least one condition is true.

  Eg1: Retrieves employees who are in either the IT or HR department.

  ```
  SELECT * FROM employees

  WHERE department = 'IT' OR department = 'HR';
  ```

  Eg2: Retrieves users whose names start with F or whose emails end with .com.

  ```
  SELECT * FROM users

  WHERE name LIKE 'F%' OR email LIKE '%.com';
  ```

  Eg3: Retrieves employees who are in the IT or HR department and have a salary greater than 5300.

  ```
  SELECT * FROM employees

  WHERE (department = 'IT' OR department = 'HR') AND salary > 5300;
  ```

  Eg3: Find Employees in Non-IT Departments

  ```
  SELECT * FROM employees WHERE NOT department = 'IT';
  ```

Arithmetic Expressions in SQL

Description

Arithmetic expressions are used to perform mathematical operations on numeric data in SQL.

Supported operators include:

- `+` (Addition)
- `-` (Subtraction)
- `*` (Multiplication)
- `/` (Division)
- `%` (Modulo, remainder after division)

```
SELECT column_name, column_name [arithmetic_expression] AS alias_name
FROM table_name;
```

Eg1: Calculate Total Salary with Bonus

Adds a bonus of `500` to each employee's salary and returns the total.

```
SELECT employee_id, salary,
(salary + 500) AS total_salary
FROM employees;
```

Eg2: Calculate Annual Salary

Multiplies the monthly salary by 12 to compute the annual salary.

```
SELECT employee_id, salary, (salary * 12) AS annual_salary FROM employees;
```

`ORDER BY` Clause (ASC, DESC)

Description

The `ORDER BY` clause is used to sort the query result based on one or more columns. The sorting can be in:

- `ASC` (Ascending order, default): Smallest to largest.
- `DESC` (Descending order): Largest to smallest.

```
SELECT column1, column2

FROM table_name

ORDER BY column_name [ASC|DESC];
```

Eg1: Retrieves employees sorted by salary from the lowest to the highest.

```
SELECT employee_id, job_title, salary

FROM employees

ORDER BY salary ASC;
```

Eg2: Sort Users Alphabetically by Name

```
SELECT id, name, email

FROM users

ORDER BY name ASC;
```

Eg3: Sort by Multiple Columns

```
SELECT employee_id, department, salary FROM employees

ORDER BY department ASC, salary DESC;
```

GROUP BY Clause in SQL

Description

The `GROUP BY` clause is used to aggregate rows with the same values in specified columns into summary rows, like calculating totals, averages, counts, etc. It is often used with **aggregate functions** such as:

- `COUNT()`
- `SUM()`
- `AVG()`
- `MAX()`
- `MIN()`

Syntax

Select "picks the columns"

from "picks the table(s)"

group by "selected column(s)"

having "impose a condition";

Example 1: Counts the number of employees in each department.

```
SELECT department, COUNT(*) AS employee_count

FROM employees

GROUP BY department;
```

Example 2: Calculate Average Salary Per Department

```
SELECT department, AVG(salary) AS average_salary
FROM employees
GROUP BY department;
```

Example 3: Find the Total Salary Paid in Each Department

```
SELECT department, SUM(salary) AS total_salary

FROM employees

GROUP BY department;
```

GROUP BY with HAVING Clause

The `HAVING` clause is used to filter groups based on aggregate conditions (similar to `WHERE` but for aggregated data).

Example: Departments with More Than 1 Employee

```
SELECT department, COUNT(*) AS employee_count

FROM employees

GROUP BY department

HAVING COUNT(*) > 1;
```

- Explanation: Only returns departments that have more than one employee.

Task #2 on Comparison, Filtering, and Aggregation

🔹 Level 1 → Basic Queries

1. Select all employees from the `employees` table whose **salary is greater than 5000**.
2. Select all orders from the `orders` table where the **order_total is between 200 and 400**.
3. Get employees with a **salary greater than or equal to 4500 but less than 6000**.
4. Select all users from the `users` table whose **email contains "gmail"** (use **LIKE**).

🔹 Level 2 – Conditional & Filtering Queries

4. Select the employee id and salary where the employee works in the **IT department AND salary > 5200**.
5. Select all orders where the **order_total < 300 OR order_date > '2024-01-18'**.
6. Select all employees whose department is **IN ('IT', 'Marketing', 'HR')**.
7. Select all employees ordered by salary in **descending (DESC)** order.

🔹 Level 3 – Aggregation & Joins

8. Show the **number of orders per employee** (use GROUP BY employee_id).
9. Show the **department name and average salary**, but only for departments where the **average salary > 5000** (GROUP BY with HAVING).
10. List employees whose job titles have **exactly one character before "ngineer"** (e.g., "Engineer", "Tngineer").