### **SQL** keywords

### **1. SELECT**

Use: Retrieve data from one or more tables.

#### **Example:**

SELECT name, age FROM employees;

Real-Time Use Case: Used in dashboards to fetch data like employee records, customer orders, etc.

### 🔍 2. FROM

Use: Specifies the table from which to retrieve data.

#### **Example:**

SELECT \* FROM orders;

**Real-Time Use Case:** Tells the system where to look for the data—like "orders" table in an ecommerce app.

### 3. WHERE

**Use:** Filters records based on specific conditions.

#### **Example:**

SELECT \* FROM customers WHERE city = 'New York';

**Real-Time Use Case:** Filtering users based on location in a CRM or marketing tool.

### 4. JOIN (INNER, LEFT, RIGHT, FULL)

**Use:** Combine rows from two or more tables based on a related column.

### **Example:**

SELECT orders.id, customers.name

**FROM orders** 

JOIN customers ON orders.customer\_id = customers.id;

Real-Time Use Case: Combine customer and order data to generate invoices or analytics reports.

### S. GROUP BY

**Use:** Groups rows that have the same values into summary rows.

1 | Page

SQL

M. ROHIT MCA, M. TECH

### **Example:**

SELECT department, COUNT(\*)

FROM employees

GROUP BY department;

Real-Time Use Case: Summarize employee count by department, used in HR dashboards.

### 🔦 6. ORDER BY

Use: Sort the result set by one or more columns.

### **Example:**

SELECT \* FROM products ORDER BY price DESC;

**Real-Time Use Case:** Display the most expensive products first in an online store.

### **3.** 7. INSERT INTO

Use: Adds new records into a table.

#### **Example:**

INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');

**Real-Time Use Case:** Registering a new user on a website or app.

### **%** 8. UPDATE

Use: Modifies existing records.

#### **Example:**

UPDATE orders SET status = 'shipped' WHERE id = 101;

**Real-Time Use Case:** Change the status of an order in a logistics app.

### 9. DELETE

Use: Removes one or more records.

#### **Example:**

DELETE FROM users WHERE last\_login < '2022-01-01';

**Real-Time Use Case:** Clean up old or inactive accounts from a database.

### 10. CREATE TABLE

Use: Defines a new table and its columns.

### **Example:**

```
id INT PRIMARY KEY,
name VARCHAR(100),
price DECIMAL(10, 2)
);
```

Real-Time Use Case: Used during database setup for applications like inventory systems.

### 11. ALTER TABLE

Use: Modifies an existing table (add/modify/delete columns).

### **Example:**

ALTER TABLE users ADD phone\_number VARCHAR(15);

Real-Time Use Case: Add a new column for additional user details in production.

### 12. DROP TABLE

Use: Deletes a table and all its data.

### **Example:**

DROP TABLE test users;

Real-Time Use Case: Removing temporary tables created during testing.

### **13. HAVING**

Use: Filters groups (used with GROUP BY).

### **Example:**

SELECT department, COUNT(\*)

FROM employees

**GROUP BY department** 

HAVING COUNT(\*) > 5;

**Real-Time Use Case:** Find departments with more than 5 employees in reporting tools.

DAY-II

SQL

M. ROHIT MCA, M. TECH

### 14. DISTINCT

**Use:** Removes duplicate values in the result set.

**Example:** 

SELECT DISTINCT city FROM customers;

**Real-Time Use Case:** Show unique locations where customers are based.

### 15. LIMIT / OFFSET

Use: Restrict number of records returned (for pagination).

**Example:** 

SELECT \* FROM products LIMIT 10 OFFSET 20;

Real-Time Use Case: Load next set of 10 products in a paginated product list.

# **SQL** data types

### 1. INT / INTEGER

• Use: Stores whole numbers (no decimals)

• Example: INT, INTEGER, INT(11)

• Real-Time Use Case:

- o Employee ID (employee\_id INT)
- o Product quantity (stock INT)
- User age (age INT)

```
create table employees (
id INT PRIMARY KEY,
age INT
);
```

## (§) 2. DECIMAL(p, s) or NUMERIC(p, s)

- Use: Stores exact numeric values with decimals
  - o p = precision (total digits), s = scale (digits after decimal)

- Example: DECIMAL(10, 2) means max 10 digits, 2 after decimal
- Real-Time Use Case:
  - o Product prices (price DECIMAL(8, 2))
  - o Financial transactions (amount DECIMAL(12, 2))

```
CREATE TABLE products (

name VARCHAR(100),

price DECIMAL(10, 2)
);
```

### 3. VARCHAR(n)

- Use: Variable-length string, max length n
- Example: VARCHAR(50)
- Real-Time Use Case:
  - o Usernames, emails, product names, city names
  - o Flexible for fields with varying length

```
CREATE TABLE users (
username VARCHAR(50),
email VARCHAR(100)
);
```

### 4. CHAR(n)

- Use: Fixed-length string, always n characters
- Example: CHAR(2) for state codes
- Real-Time Use Case:
  - o Country codes, gender codes ('M', 'F'), abbreviations

```
CREATE TABLE country_codes (

code CHAR(2),

name VARCHAR(100)
);
```

### 5. DATE

```
• Use: Stores only date (YYYY-MM-DD)
```

• Example: '2025-08-29'

### • Real-Time Use Case:

- o Employee birth dates
- o Order date, join date, invoice date

```
CREATE TABLE orders (
order_id INT,
order_date DATE
);
```

### **⚠** 6. DATETIME

• Use: Stores date and time (YYYY-MM-DD HH:MM:SS)

Example: '2025-08-29 14:30:00'

Real-Time Use Case:

o Timestamp of order placements, logins, messages

```
CREATE TABLE logs (
user_id INT,
login_time DATETIME
);
```

### **7. TIMESTAMP**

- Use: Similar to DATETIME but with auto-update features in some DBs (e.g., MySQL)
- Real-Time Use Case:
  - o Auto track record creation or modification time

```
CREATE TABLE activities (
id INT,
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### **✓** 8. BOOLEAN

• Use: Stores true/false or 1/0

### **Real-Time Use Case:**

- User is\_active flag
- Product is\_available flag
- o Email verified or not

```
CREATE TABLE users (
id INT,
is_active BOOLEAN
);
```

### **₽** 9. TEXT

- Use: Stores long text (no size limit like VARCHAR)
- Real-Time Use Case:
  - o Product descriptions, reviews, blog content

```
CREATE TABLE reviews (
review_id INT,
content TEXT
);
```

### ☐ 10. BLOB (Binary Large Object)

- Use: Stores binary data like images, files, videos
- Real-Time Use Case:
  - User profile photos
  - o Uploaded documents (e.g., resumes)

```
CREATE TABLE documents (
id INT,
file BLOB
);
```

### **✓** Summary Table

# D&Y-II

SQL

M. ROHIT MCA, M. TECH

Data Type Description Real-Time Use

INT Whole numbers IDs, quantities, counters

DECIMAL Exact decimals Prices, financial data

VARCHAR(n) Variable text Names, emails, city

CHAR(n) Fixed text Gender, country codes

DATE Only date Birthdate, order date

DATETIME Date and time Logins, transactions

TIMESTAMP Auto-tracking time Created/updated times

BOOLEAN True/false Status flags (active/inactive)

TEXT Long text Reviews, descriptions

BLOB Binary files Images, files

### ☐ teaching Tip:

Encourage students to **choose the right data type** during table design — this leads to better performance and cleaner data. For example:

- Don't use TEXT for short labels (use VARCHAR)
- Use DECIMAL instead of FLOAT for money
- Prefer BOOLEAN for flags instead of INT(1)

some real-time SQL data type interview questions that are commonly asked in MNC interviews (TCS, Infosys, Accenture, Wipro, Cappemini, Cognizant, etc.), especially when evaluating candidates for database-related roles, backend development, or data analysis.

These questions often test not just theory, but **why and when** to use a particular data type in real-world scenarios.

### Frequently Asked Interview Questions (SQL Data Types)

#### 1. What is the difference between CHAR and VARCHAR?

Interview Focus: Data storage, performance, use cases

**Key Answer Points:** 

# DAY-II

# SQL

# QL M. ROHIT MCA, M. TECH

- CHAR(n) is fixed-length, always uses n bytes.
- VARCHAR(n) is variable-length, uses only the space needed (plus 1-2 bytes for length).
- Use CHAR when all entries are the same length (e.g., country codes: 'IN', 'US').
- Use VARCHAR for fields with varying length (e.g., names, email addresses).

### Follow-up:

"In which scenario would CHAR be better than VARCHAR?"

### 2. Why should we use DECIMAL instead of FLOAT for storing currency values?

Interview Focus: Precision and data accuracy

### **Key Answer Points:**

- FLOAT is **approximate** and can introduce rounding errors.
- DECIMAL(p, s) is **exact**, preserves precision.
- For money-related fields (e.g., price, salary), use DECIMAL(10,2).

#### Follow-up:

"What will happen if we store ₹100.75 in a FLOAT column?"

### 3. What is the difference between DATE, DATETIME, and TIMESTAMP?

Interview Focus: Time-related data storage and default behavior

### **Key Answer Points:**

- DATE: Only date (e.g., '2025-08-29')
- DATETIME: Date + time (e.g., '2025-08-29 15:30:00')
- TIMESTAMP: Also includes date + time, auto-updates in some DBMS (e.g., MySQL)

### **Real-Time Use Cases:**

• DATE: Birthdate

• DATETIME: Appointment time

• TIMESTAMP: Record creation/modification tracking

### 4. What is the maximum size of VARCHAR?

Interview Focus: Limits and best practices

### **Key Answer Points:**

- Depends on DBMS:
  - MySQL: up to 65,535 bytes (shared with row size)



# SQL

# M. ROHIT MCA, M. TECH

- o PostgreSQL: practically unlimited
- Use appropriate size, e.g., VARCHAR(100) for names.
- Don't over-allocate; affects indexing and performance.

#### 5. What is the difference between TEXT and VARCHAR? When to use TEXT?

Interview Focus: Long string data handling

#### **Key Answer Points:**

- TEXT is used for very **long strings** (e.g., articles, reviews).
- VARCHAR is better for **short to medium-length** strings (e.g., name, email).
- TEXT columns are usually stored **outside the table row**, can't be indexed efficiently.

**Use TEXT:** product descriptions, blog content.

Avoid TEXT: searchable fields.

### 6. Can you store an image or a PDF file in a SQL table? If yes, which data type would you use?

Interview Focus: BLOB usage and file handling

#### **Key Answer Points:**

- Yes, use BLOB (Binary Large Object).
- Stores binary data like images, PDFs, etc.
- Not always recommended for large files better to store file paths and keep files in storage.

### 7. When would you use BOOLEAN, and how is it stored internally in SQL databases?

Interview Focus: Data representation

#### **Key Answer Points:**

- BOOLEAN is for true/false values (e.g., is\_active, is\_deleted).
- Internally stored as 1 (true) or 0 (false) in most DBMS.
- Useful for flags in tables (e.g., is\_email\_verified BOOLEAN).

### 8. What's the default value behavior of TIMESTAMP in MySQL?

Interview Focus: Auto-timestamping

#### **Key Answer Points:**

Can be set to auto-update with DEFAULT CURRENT\_TIMESTAMP

Useful for created\_at, updated\_at tracking

### **Example:**

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

#### 9. What is the difference between NUMERIC and DECIMAL?

Interview Focus: Data type aliasing

#### **Key Answer Points:**

- Functionally identical in most DBMS (like PostgreSQL, SQL Server).
- Both are exact numeric types for fixed precision.
- Use them for financial data.

### 10. Why is it important to choose the right data type in real-time projects?

Interview Focus: Design and performance thinking

### **Key Answer Points:**

- Saves storage space
- Improves query performance
- Prevents data integrity issues
- Ensures accurate indexing and constraints

### Bonus Tip for Students:

"Don't just memorize definitions — think about how that data type affects performance, storage, and accuracy in real apps."

This means:

Don't just know what a data type is, but understand why and when to use it — based on how it behaves in real systems.

**11** | Page R@HIT

### Let's take 3 key factors:

- 1. Performance
- 2. Storage
- 3. Accuracy

We'll explain each with real-world examples so it's clear.

### 1. Performance

How fast queries run, especially with large data

### **Example:**

```
CREATE TABLE users (

phone_number VARCHAR(100)
);
```

This is **functional** but **inefficient**. Why?

- A phone number is always fixed length (e.g., 10 digits).
- If you use VARCHAR(100), the DB has to check how long the string is for each record.
- Instead, use: CHAR(10) → better performance on large tables.

### **✓** Better Design:

phone\_number CHAR(10)

### 於 Takeaway:

Use fixed-length types (CHAR) when the data is predictable. It helps indexing and searching run faster.

### 2. Storage

How much space your database uses (and wastes)

### Reample:

```
CREATE TABLE cities (
name VARCHAR(255)
);
```

Seems okay, right? But if 99% of your city names are under 50 characters, you're **reserving too much space**.

Also, in MySQL:

VARCHAR(255) needs up to 256 bytes

SQL

M. ROHIT MCA, M. TECH

VARCHAR(50) needs only 51 bytes

So, on millions of rows, this becomes expensive in memory.

### **✓** Better Design:

name VARCHAR(50)



Don't blindly use large sizes like VARCHAR(255) unless really needed. Choose the **shortest possible** size that fits your data.

### 3. Accuracy

How precisely your data is stored (especially important in numbers)

### **Example:**

CREATE TABLE transactions (

amount FLOAT

);

But FLOAT is an approximate type. It might store:

• ₹99.99 as 99.989999999 or 99.9900001

That's unacceptable for money, right?

### **✓** Better Design:

amount DECIMAL(10, 2)

This ensures:

- Always exact 2 digits after decimal
- ₹99.99 will stay exactly 99.99

### 

For financial data, **never use FLOAT or REAL** — use **DECIMAL/NUMERIC** for accuracy.

#### **%** Real-Time Project Summary Table:

| Scenario       | Bad Data Type | Good Data Type | Why? (Issue Solved)  |
|----------------|---------------|----------------|----------------------|
| Phone number   | VARCHAR(100)  | CHAR(10)       | Better performance   |
| City name      | VARCHAR(255)  | VARCHAR(50)    | Saves storage        |
| Salary / price | FLOAT         | DECIMAL(10,2)  | Keeps precise values |

DAY-II

SQL M. ROHIT MCA, M. TECH

Scenario Bad Data Type Good Data Type Why? (Issue Solved)

True/false field INT(1) BOOLEAN Clear intent, better readability

Product description VARCHAR(1000) TEXT No size limit, suited for long text

Image or PDF file VARCHAR (path only) BLOB Binary data support

### ☐ Final Thought:

A good developer doesn't just *make it work*, they make it *work well*. Choosing the **right data type** is your first step to building clean, fast, and scalable apps.

### **SQL** operators

### (a) Common SQL Operators and Their Uses in Real-Time Projects

### 1. = (Equal To)

- Use: Compares if two values are equal
- Real-time example: Find users with a specific email

SELECT \* FROM users WHERE email = 'user@example.com';

#### 2. <> or != (Not Equal To)

- Use: Filters out records that do not match a value
- Real-time example: Find products that are not discontinued

SELECT \* FROM products WHERE status <> 'discontinued';

### 3. > (Greater Than)

- Use: Finds values greater than a specified value
- Real-time example: Find orders with amount greater than 1000

SELECT \* FROM orders WHERE amount > 1000;

### 4. < (Less Than)

• Use: Finds values less than a specified value

# DAY-II

# SQL M. ROHIT MCA, M. TECH

Real-time example: Find employees with salary less than 50000

SELECT \* FROM employees WHERE salary < 50000;

### 5. >= (Greater Than or Equal To)

- Use: Finds values greater than or equal to a specified value
- Real-time example: Find products with stock greater than or equal to 10

SELECT \* FROM products WHERE stock >= 10;

### 6. <= (Less Than or Equal To)

- Use: Finds values less than or equal to a specified value
- Real-time example: Find orders placed before or on a specific date

SELECT \* FROM orders WHERE order\_date <= '2025-08-29';

#### **7. BETWEEN ... AND ...**

- Use: Filters values within a range (inclusive)
- Real-time example: Find employees with salary between 30000 and 60000

SELECT \* FROM employees WHERE salary BETWEEN 30000 AND 60000;

#### 8. LIKE

- Use: Pattern matching with wildcards (% and \_)
- Real-time example: Find customers whose name starts with 'J'

SELECT \* FROM customers WHERE name LIKE 'J%';

#### 9. IN

- Use: Checks if a value matches any value in a list
- Real-time example: Find orders from specific customers

SELECT \* FROM orders WHERE customer id IN (101, 102, 103);

### 10. IS NULL and IS NOT NULL

- Use: Checks if a value is or isn't null (missing)
- Real-time example: Find users who haven't set their phone number

SELECT \* FROM users WHERE phone\_number IS NULL;

### 11. AND, OR, NOT (Logical Operators)

- **Use:** Combine multiple conditions
- Real-time example: Find active customers from a city

SELECT \* FROM customers WHERE city = 'New York' AND is\_active = TRUE;

### 12. +, -, \*, / (Arithmetic Operators)

- Use: Perform calculations on numeric fields
- Real-time example: Calculate total price after tax

SELECT price, price \* 1.10 AS price\_with\_tax FROM products;

#### 13. EXISTS

- Use: Checks if a subquery returns any rows
- Real-time example: Find customers who have placed orders

SELECT \* FROM customers c WHERE EXISTS (SELECT 1 FROM orders o WHERE o.customer\_id = c.id);

### **Summary Table:**

| Operator             | Use Case              | Real-Time Example                      |
|----------------------|-----------------------|--|
| =                    | Equality              | Find user by email                     |
| <> or !=             | Not equal             | Find products not discontinued         |
| >                    | Greater than          | Find orders with amount > 1000         |
| <                    | Less than             | Find employees with salary < 50000     |
| >=                   | Greater or equal      | Find products with stock >= 10         |
| <=                   | Less or equal         | Find orders placed before a date       |
| BETWEEN AND          | Range check           | Salary between 30k and 60k             |
| LIKE                 | Pattern matching      | Customers with names starting with 'J' |
| IN                   | Multiple values check | Orders from specific customers         |
| IS NULL / IS NOT NUL | L Null check          | Users without phone number             |
|                      |                       |  |

**16** | Page

D&Y-II

SQL M. ROHIT MCA, M. TECH

| Operator     | Use Case                 | Real-Time Example              |
|--------------|--------------------------|--------------------------------|
| AND, OR, NOT | Logical operators        | Active customers from New York |
| +, -, *, /   | Arithmetic calculations  | Price calculation with tax     |
| EXISTS       | Subquery existence check | c Customers who placed orders  |

# Real-Time SQL Operator Interview Questions (with explanation of what interviewers look for)

### 1. What is the difference between WHERE and HAVING clauses? When do you use each?

- Focus: Filtering rows vs filtering groups
- Real Scenario: Filtering data after aggregation
- Answer Tip:
  - o WHERE filters rows before aggregation (e.g., before GROUP BY)
  - o HAVING filters groups after aggregation (e.g., after GROUP BY)

#### 2. How does the IN operator differ from multiple OR conditions? Which one is more efficient?

- Focus: Query readability and optimization
- Real Scenario: Filtering by multiple values
- Answer Tip:
  - o IN is shorthand for multiple ORs, easier to read
  - o Performance depends on the database, but generally both are similar
  - o For very large lists, JOIN with temp table or indexed table might be better

#### 3. Explain how the LIKE operator works. What are the wildcard characters?

- Focus: Pattern matching
- Real Scenario: Searching for users with partial input
- Answer Tip:
  - % matches zero or more characters
  - \_ matches exactly one character

- LIKE 'a%' matches strings starting with 'a'
- o Explain performance issues with leading % (e.g., LIKE '%abc' can be slow)

### 4. How would you write a query to find records where a column is NULL or empty?

- Focus: NULL handling in SQL
- Real Scenario: Data cleansing or filtering
- Answer Tip:
- WHERE column IS NULL OR column = "
  - o Explain difference between NULL and empty string "

#### 5. What is the difference between = and LIKE when used in WHERE clause?

- Focus: Exact vs pattern matching
- Real Scenario: Search vs exact match
- Answer Tip:
  - = is for exact matches
  - LIKE is for partial or pattern matches

### 6. When would you use the BETWEEN operator? Are the boundaries inclusive or exclusive?

- **Focus:** Range filtering
- Real Scenario: Filtering dates, numbers, or ranges
- Answer Tip:
  - Use BETWEEN for inclusive ranges
  - o Boundaries are **inclusive** (e.g., BETWEEN 10 AND 20 includes 10 and 20)

### 7. Explain the difference between AND and OR operators. What happens if you combine them without parentheses?

- Focus: Logical operator precedence
- Real Scenario: Complex filtering conditions
- Answer Tip:
  - o AND has higher precedence than OR
  - Use parentheses to explicitly define order

**18** | Page R@HIT Example: A OR B AND C is A OR (B AND C)

### 8. What is the purpose of the EXISTS operator? How does it differ from IN?

- Focus: Subquery usage and performance
- Real Scenario: Checking existence of related records
- Answer Tip:
  - EXISTS checks if a subquery returns any row (returns TRUE/FALSE)
  - o IN compares against a list of values
  - o EXISTS can be more efficient with correlated subqueries

### 9. Can you use arithmetic operators in WHERE clause? Give an example.

- Focus: Expression evaluation
- Real Scenario: Filtering based on calculated columns
- Answer Tip:
- WHERE (price \* quantity) > 1000
  - Shows understanding of SQL expressions

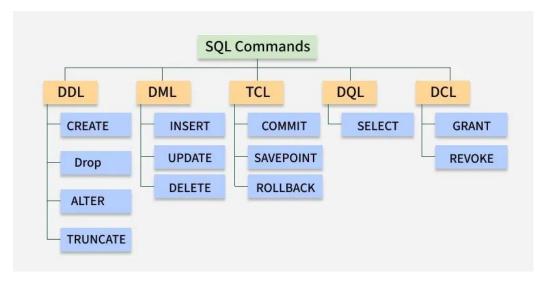
### 10. What happens if you use NULL in arithmetic operations or comparisons? How do you handle it?

- Focus: Null behavior in SQL
- Real Scenario: Data calculation and filtering
- Answer Tip:
  - o Any arithmetic operation with NULL results in NULL
  - Use IS NULL to check for nulls
  - o Use COALESCE(column, default\_value) to replace nulls in calculations

# **SQL Commands | DDL, DQL, DML, DCL and TCL Commands**

SQL commands are fundamental building blocks for communicating with a database management system (DBMS) used to interact with database with some operations. It is also used to perform specific tasks, functions and queries of data. SQL can perform various tasks like creating a table, adding data to tables, dropping the table, modifying the table, set permission for users.

SQL Commands are mainly categorized into five categories:



#### **SQL** Commands

### 1. DDL - Data Definition Language

DDL (Data Definition Language) actually consists of SQL commands that can be used for defining, altering and deleting database structures such as tables, indexes and schemas. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database

### **Common DDL Commands**

| Command  | Description  | Syntax   |
|----------|--|--|
| CREATE   | Create database or its objects (table, index, function, views, store procedure and triggers) | CREATE TABLE table_name (column1 data_type, column2 data_type,); |
| DROP     | Delete objects from the database   | DROP TABLE table_name;   |
| ALTER    | Alter the structure of the database  | ALTER TABLE table_name ADD COLUMN column_name data_type;         |
| TRUNCATE | Remove all records from a table, including all spaces allocated for the records are removed  | TRUNCATE TABLE table_name;                                       |

| Command | Description                               | Syntax   |
|---------|---|--|
| COMMENT | Add comments to the data dictionary       | COMMENT ON TABLE table_name IS 'comment_text'; |
| RENAME  | Rename an object existing in the database | RENAME TABLE old_table_name TO new_table_name; |

### **Example:**

```
CREATE TABLE employees (
employee_id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
hire_date DATE
);
```

In this example, a new table called employees is created with columns for employee ID, first name, last name and hire date.

### 2. DQL - Data Query Language

DQL is used to fetch data from the database. The main command is SELECT, which retrieves records based on the query. The output is returned as a result set (a temporary table) that can be viewed or used in applications.

### **DQL Command**

| Command     | Description   | Syntax   |
|-------------|---|--|
| SELECT      | It is used to retrieve data from the database               | SELECT column1, column2,FROM table_name WHERE condition; |
| FROM        | Indicates the <b>table(s)</b> from which to retrieve data.  | SELECT column1 FROM table_name;                          |
| WHERE       | Filters rows <b>before</b> any grouping or aggregation      | SELECT column1 FROM table_name WHERE condition;          |
| GROUP<br>BY | Groups rows that have the same values in specified columns. | SELECT column1, AVG_FUNCTION(column2)                    |

| Command     | Description   | Syntax   |
|-------------|---|--|
|             |   | FROM table_name GROUP BY column1;  |
| HAVING      | Filters the results of GROUP BY   | SELECT column1, AVG_FUNCTION(column2) FROM table_name GROUP BY column1 HAVING condition; |
| DISTINCT    | Removes <b>duplicate rows</b> from the result set                       | SELECT DISTINCT column1, column2, FROM table_name;                                       |
| ORDER<br>BY | Sorts the result set by one or more columns                             | SELECT column1 FROM table_name ORDER BY column1 [ASC   DESC];                            |
| LIMIT       | By default, it sorts in <b>ascending order</b> unless specified as DESC | SELECT * FROM table_name LIMIT number;   |

**Note:** DQL has only one command, **SELECT**. Other terms like FROM, WHERE, GROUP BY, HAVING, ORDER BY, DISTINCT and LIMIT are **clauses** of SELECT, not separate commands.

#### **Example:**

SELECT first\_name, last\_name, hire\_date FROM employees WHERE department = 'Sales' ORDER BY hire\_date DESC;

This query retrieves employees first and last names, along with their hire dates, from the employees table, specifically for those in the 'Sales' department, sorted by hire date.

### 3. DML - Data Manipulation Language

DML commands are used to manipulate the data stored in database tables. With DML, you can insert new records, update existing ones, delete unwanted data or retrieve information.

#### **Common DML Commands**

| Command       | Description                          | Syntax  |
|---------------|--------------------------------------|---|
| INSERT        | Insert data into a table             | INSERT INTO table_name (column1, column2,) VALUES (value1, value2,);      |
| <u>UPDATE</u> | Update existing data within a table  | UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition; |
| DELETE        | Delete records from a database table | DELETE FROM table_name WHERE condition;                                   |

### **Example:**

INSERT INTO employees (first\_name, last\_name, department)
VALUES ('Jane', 'Smith', 'HR');

This query inserts a new record into employees table with first name 'Jane', last name 'Smith' and department 'HR'.

### 4. DCL - Data Control Language

DCL (Data Control Language) includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions and other controls of the database system. These commands are used to control access to data in the database by granting or revoking permissions.

### **Common DCL Commands**

| Command | Description   | Syntax   |
|---------|---|--|
| GRANT   | Assigns new privileges to a user account, allowing access to specific database objects, actions or functions.               | GRANT privilege_type [(column_list)] ON [object_type] object_name TO user [WITH GRANT OPTION];             |
| REVOKE  | Removes previously granted privileges from a user account, taking away their access to certain database objects or actions. | REVOKE [GRANT OPTION FOR] privilege_type [(column_list)] ON [object_type] object_name FROM user [CASCADE]; |

### **Example:**

GRANT SELECT, UPDATE ON employees TO user\_name;

This command grants the user user\_name the permissions to select and update records in the employees table.

### 5. TCL - Transaction Control Language

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group are successfully completed. If any of the tasks fail, transaction fails. Therefore, a transaction has only two results: success or failure.

#### **Common TCL Commands**

| Command              | Description  | Syntax                                |
|----------------------|--|---------------------------------------|
| BEGIN<br>TRANSACTION | Starts a new transaction                           | BEGIN TRANSACTION [transaction_name]; |
| СОММІТ               | Saves all changes made during the transaction      | COMMIT;                               |
| ROLLBACK             | Undoes all changes made during the transaction     | ROLLBACK;                             |
| SAVEPOINT            | Creates a savepoint within the current transaction | SAVEPOINT savepoint_name;             |

### **Example:**

**BEGIN TRANSACTION**;

UPDATE employees SET department = 'Marketing' WHERE department = 'Sales';

SAVEPOINT before\_update;

UPDATE employees SET department = 'IT' WHERE department = 'HR';

ROLLBACK TO SAVEPOINT before\_update;

COMMIT;

**24** | Page