DBMS

1.Forgein key:

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table

CREATE TABLE Orders (

OrderID int NOT NULL PRIMARY KEY,

OrderNumber int NOT NULL,

PersonID int FOREIGN KEY REFERENCES Persons(PersonID)

);

2.Primary key:

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key.

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

PRIMARY KEY (ID)

);

3.Unique key:

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

4.Distinct keyword:

The SELECT DISTINCT statement is used to return only distinct (different) values.

SELECT DISTINCT Country FROM Customers;

5.Limit and Offset keyword:

LIMIT and OFFSET are SQL clauses used for efficient data retrieval. LIMIT restricts the number of rows returned, useful for displaying specific results. OFFSET skips a certain number of rows, enabling pagination and efficient navigation through large datasets.

Select name from student LIMIT 10 OFFSET 20;

6.Joins and its types:

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

FROM Orders

INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

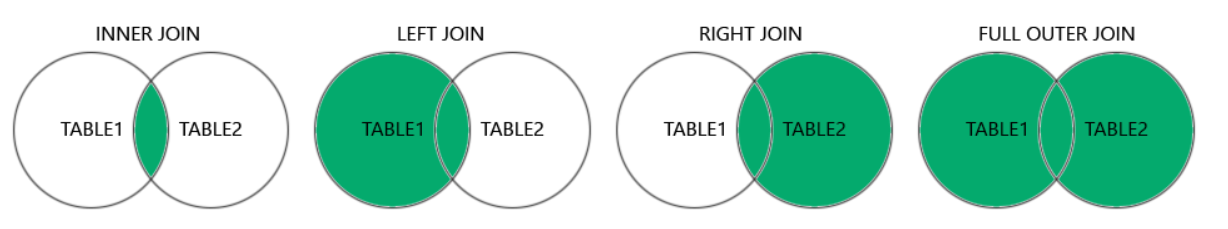
Types:

(INNER) JOIN: Returns records that have matching values in both tables

LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table

RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table

FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table



7.LIKE clause:

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

The percent sign % represents zero, one, or multiple characters

The underscore sign \_ represents one, single character

SELECT \* FROM Customers  
WHERE CustomerName LIKE 'a%';

8.Wildcard:

The \_ wildcard represents a single character.

It can be any character or number, but each \_ represents one, and only one, character.

SELECT \* FROM Customers

WHERE city LIKE 'L\_nd\_\_';

9.Aggregate functions:

An aggregate function is a function that performs a calculation on a set of values, and returns a single value.

MIN() - returns the smallest value within the selected column

MAX() - returns the largest value within the selected column

COUNT() - returns the number of rows in a set

SUM() - returns the total sum of a numerical column

AVG() - returns the average value of a numerical column

10.ORDER BY keyword:

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

SELECT column1, column2, ...

FROM table\_name

ORDER BY column1, column2, ... ASC|DESC;

11.Aliases:

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

SELECT CustomerID AS ID

FROM Customers;

12.Between operator:

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

SELECT \* FROM Products  
WHERE Price BETWEEN 10 AND 20;

13.IN operator:

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

SELECT \* FROM Customers  
WHERE Country IN ('Germany', 'France', 'UK');

14.HAVING clause:

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

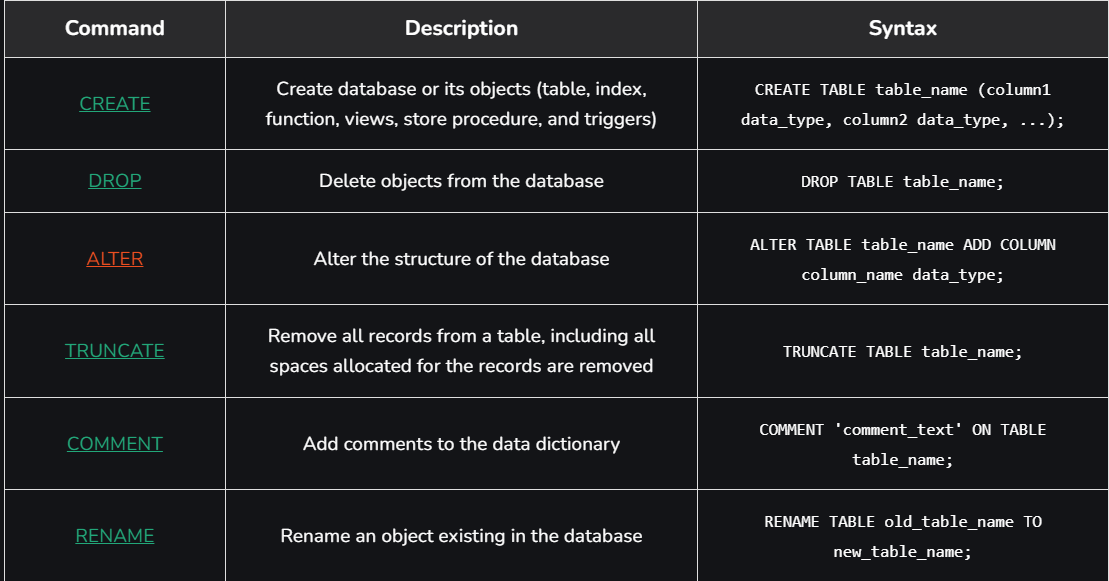
SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;

15.Triggers:

A trigger is a stored procedure in a database that automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when specific table columns are updated.

16. DDL:

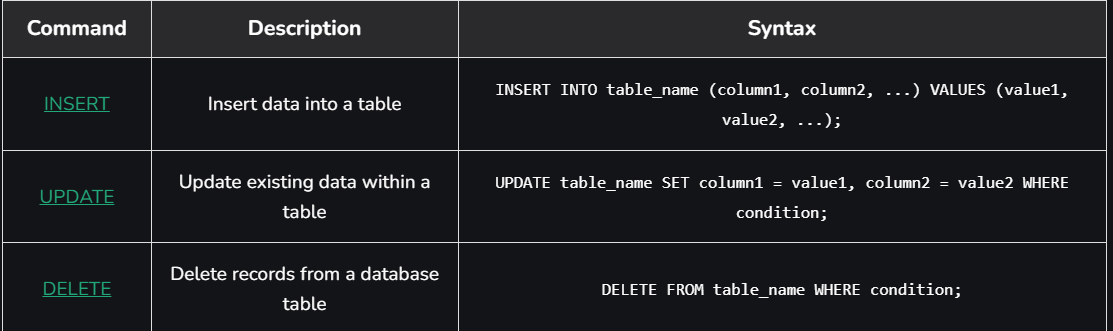
DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.



DML(Data Manipulation Language):

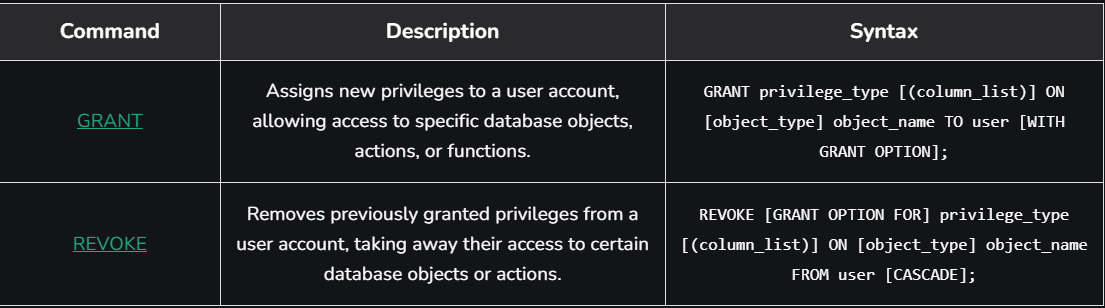
The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.



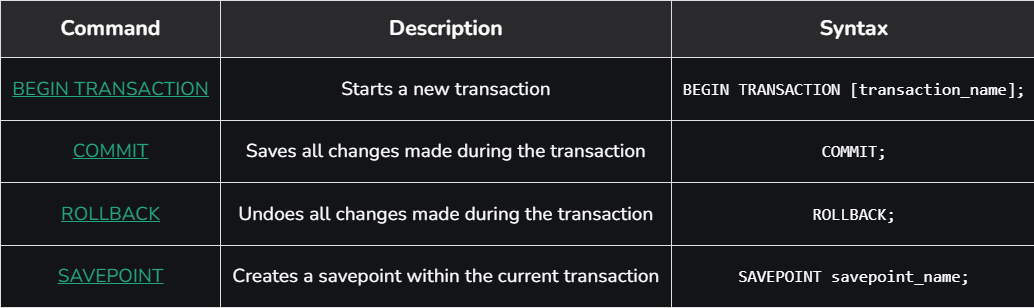
DCL:

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.



TCL:

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, the transaction fails.



17.SQL vs noSQL

The choice between SQL (Structured Query Language) and NoSQL (Not Only SQL) databases depends on the specific requirements and constraints of your project. Here are the main differences and considerations for each:

**SQL Databases**

**Characteristics:**

* **Structured Data:** Uses a fixed schema with predefined tables and columns.
* **ACID Transactions:** Ensures atomicity, consistency, isolation, and durability.
* **Relational Model:** Data is stored in tables with relationships (foreign keys).
* **Examples:** MySQL, PostgreSQL, SQLite, SQL Server, Oracle.

**Use Cases:**

* Applications requiring complex queries and transactions.
* Financial systems, ERP systems, and other applications where data integrity and consistency are crucial.
* Situations where relationships between data entities are important.

**NoSQL Databases**

**Characteristics:**

* **Flexible Schema:** Can store unstructured, semi-structured, or structured data.
* **Horizontal Scalability:** Designed to scale out by adding more servers.
* **Diverse Data Models:** Includes document (e.g., MongoDB), key-value (e.g., Redis), column-family (e.g., Cassandra), and graph (e.g., Neo4j) databases.
* **Eventual Consistency:** Prioritizes availability and partition tolerance over immediate consistency (CAP theorem).

**Use Cases:**

* Applications with large volumes of diverse data.
* Real-time web applications, big data analytics, and content management systems.
* Situations where quick scalability and high availability are needed.

**Pros and Cons**

**SQL Databases:**

* **Pros:** Strong consistency, complex query support, mature and well-supported.
* **Cons:** Less flexible schema, harder to scale horizontally, performance bottlenecks with very large datasets.

**NoSQL Databases:**

* **Pros:** Flexible schema, easy to scale horizontally, handles large volumes of data well.
* **Cons:** Weaker consistency (eventual consistency), less support for complex queries and transactions, relatively newer and less mature.