SQL Documentation

In SQL, there are four main types of language commands, each serving a specific purpose. These are DCL (Data Control Language), DDL (Data Definition Language), DML (Data Manipulation Language), and TCL (Transaction Control Language). Let's break down each one with definitions, examples, and sample SQL queries.

A. Data Control Language (DCL)

DCL is used to control access to data in a database. It deals with permissions and access control.

1. **GRANT:** Gives a user access privileges to the database.

   >> GRANT SELECT ON employees TO user_name;

2. **REVOKE:** Removes user access privileges to the database.

   >> REVOKE SELECT ON employees FROM user_name;

B. Data Definition Language (DDL)

DDL is used to define and modify the structure of database objects like tables, indexes, and schemas. It directly impacts the structure of the database.

1. **CREATE:** Creates new database objects (tables, views, indexes).

   >> CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50), department VARCHAR(50), salary INT);

2. **ALTER:** Modifies the structure of existing database objects.

   >> ALTER TABLE employees ADD date_of_joining DATE;

3. **TRUNCATE:** Removes all records from a table but keeps its structure.

   >> TRUNCATE TABLE employees;

4. **DROP:** Deletes the complete data and structure.

   >> DROP TABLE employees;

C. Data Manipulation Language (DML)

DML is used for managing data within schema objects. It focuses on manipulating the data present in the database.

1. **INSERT:** Adds new data into a table.

   >> INSERT INTO employees (id, name, department, salary) VALUES (1, ‘Adam’, ‘HR’, 50000);

2. **UPDATE:** Modifies existing data within a table.

   >> UPDATE employees SET salary=55000 WHERE id=1;

3. **DELETE:** Removes existing data from a table.

   >> DELETE FROM employees WHERE id=1;

4. **SELECT:** Select existing data within a table.

   >> SELECT * FROM employees;

D. Transaction Control Language (TCL)

TCL is used to manage the changes made by DML statements. It ensures the integrity of the database by controlling transactions.

1. **COMMIT:** Save all changes made during the current transaction.

   >> COMMIT;

2. **ROLLBACK:** Undoes all changes made during the current transaction.

   >> ROLLBACK TO sp1;

3. **SAVEPOINT:** Sets a point within a transaction to which you can later roll back.

   >> SAVEPOINT sp1;

4. **SET TRANSACTION:** Set properties for the current transaction (like isolation level).

   >> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

DATABASE

1. ***Create Database***
   - Used to create a new SQL database.

>> CREATE DATABASE testdb;

```
mysql> create database testdb;
Query OK, 1 row affected (0.05 sec)
```

2. ***Drop Database***
   - Used to drop an existing SQL database.

>> DROP DATABASE testdb;

```
mysql> drop database testdb;
Query OK, 0 rows affected (0.07 sec)
```

3. ***Select a specific database***
   - Used to select a database to manage tables.

>> USE fsds;

```
mysql> create database fsds;
Query OK, 1 row affected (0.01 sec)

mysql> use fsds;
Database changed
mysql>
```

4. ***Create Table***
   - used to create a new table in a database.

>> CREATE TABLE persons(PersonID INT, LastName VARCHAR(20), FirstName VARCHAR(20), Address VARCHAR(20), City VARCHAR(20));

```
mysql> create table persons(PersonID int, LastName varchar(20),FirstName varchar(20),Address varchar(20),City varchar(20
));
Query OK, 0 rows affected (0.04 sec)
```

5. *Create Table Using Another Table*

- A copy of an existing table can also be created using CREATE TABLE. The new table gets the same column definitions. All columns or specific columns can be selected. If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

>> CREATE TABLE copy_persons AS SELECT PersonID, Address, City FROM persons;

```
mysql> create table copy_persons as select PersonID, Address,City from persons;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

6. *Drop Table*

- The following SQL statement drops the existing table.

>> DROP TABLE copy_persons;

```
mysql> drop table copy_persons;
Query OK, 0 rows affected (0.02 sec)
```

7. *Truncate Table*

- used to delete the data inside a table, but not the table itself.

>> TRUNCATE TABLE persons;

```
mysql> TRUNCATE TABLE persons;
Query OK, 0 rows affected (0.04 sec)
```

8. *Alter Table- ADD Column*

- To add a column in a persons table

>> ALTER TABLE persons ADD Salary int;

```
mysql> ALTER TABLE persons ADD Salary int;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

9. *Alter Table- Drop Column*

- To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column)

>> ALTER TABLE persons DROP COLUMN Salary;

```
mysql> ALTER TABLE persons DROP COLUMN Salary;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

10. *Alter Table- Rename & Modify Column*

- To rename & change the data type of a column in a table.

>> ALTER TABLE persons RENAME COLUMN old_name TO new_name;

>> ALTER TABLE persons MODIFY COLUMN LastName varchar(40);

```
mysql> ALTER TABLE persons MODIFY COLUMN LastName varchar(40);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

11. *Describe Table*

>> DESC persons;

```
mysql> desc persons;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| PersonID  | int         | YES  |     | NULL    |       |
| LastName  | varchar(40) | YES  |     | NULL    |       |
| FirstName | varchar(20) | YES  |     | NULL    |       |
| Address   | varchar(20) | YES  |     | NULL    |       |
| City      | varchar(20) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.02 sec)
```

12. *MySQL Constraints*

- *NOT NULL:* Ensures that a column cannot have a NULL value
- *UNIQUE:* Ensures that all values in a column are different
- *PRIMARY KEY:* A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- *FOREIGN KEY:* Prevents actions that would destroy links between tables

- *CHECK:* Ensures that the values in a column satisfy a specific condition
- *DEFAULT:* Ensures that the values in a column satisfy a specific condition
- *CREATE INDEX:* Used to create and retrieve data from the database very quickly

- Create the department table for foreign key reference

>> CREATE TABLE department(dept_id INT PRIMARY KEY, dept_name VARCHAR(50) NOT NULL);

- Create the student table

>> CREATE TABLE student(student_id INT PRIMARY KEY, first_name VARCHAR(50) NOT NULL, last_name VARCHAR(50) NOT NULL, email VARCHAR(100) UNIQUE NOT NULL, phone_number VARCHAR(15) UNIQUE, dept_id INT, enrollment_date DATE, FOREIGN KEY(dept_id) REFERENCES department(dept_id));

```
mysql> CREATE TABLE department(dept_id INT PRIMARY KEY, dept_name VARCHAR(50) NOT NULL);
Query OK, 0 rows affected (0.04 sec)

mysql> CREATE TABLE student( student_id INT PRIMARY KEY, first_name VARCHAR(50) NOT NULL, last_name VARCHAR(50) NOT NULL
, email VARCHAR(100) UNIQUE NOT NULL, phone_number VARCHAR(15) UNIQUE, dept_id INT, enrollment_date DATE, FOREIGN KEY(de
pt_id) REFERENCES department(dept_id));
Query OK, 0 rows affected (0.06 sec)

mysql> desc department;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| dept_id   | int         | NO   | PRI | NULL    |       |
| dept_name | varchar(50) | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> desc student;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| student_id      | int          | NO   | PRI | NULL    |       |
| first_name      | varchar(50)  | NO   |     | NULL    |       |
| last_name       | varchar(50)  | NO   |     | NULL    |       |
| email           | varchar(100) | NO   | UNI | NULL    |       |
| phone_number    | varchar(15)  | YES  | UNI | NULL    |       |
| dept_id         | int          | YES  | MUL | NULL    |       |
| enrollment_date | date         | YES  |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Explanation:

- student_id: The primary key for the student table.
- first_name and last_name: Both fields are required and cannot be NULL.
- email: Must be unique and cannot be NULL.
- phone_number: Must be unique but can be NULL if a student doesn't provide it.
- dept_id: A foreign key referencing the dept_id in the department table, allowing us to associate a student with a specific department.

13. Auto Increment field

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

>> CREATE TABLE test(id INT NOT NULL AUTO_INCREMENT, Name varchar(20), PRIMARY KEY(id));

14. Working with Dates

>> SELECT CURRENT_DATE();

>> SELECT CURRENT_DATE()+1;

```
mysql>
mysql>
mysql> SELECT CURRENT_DATE();
+----------------+
| CURRENT_DATE() |
+----------------+
| 2024-11-07     |
+----------------+
1 row in set (0.00 sec)

mysql> SELECT CURRENT_DATE()+1;
+------------------+
| CURRENT_DATE()+1 |
+------------------+
|         20241108 |
+------------------+
1 row in set (0.00 sec)
```

>> SELECT * FROM table WHERE date='2023-10-08';

15. Working with time

>> SELECT CURRENT_TIME();

>> SELECT CURRENT_TIME()+2;

```
mysql>
mysql>
mysql> SELECT CURRENT_TIME();
+----------------+
| CURRENT_TIME() |
+----------------+
| 12:43:00       |
+----------------+
1 row in set (0.00 sec)

mysql> SELECT CURRENT_TIME()+2;
+------------------+
| CURRENT_TIME()+2 |
+------------------+
|           124307 |
+------------------+
1 row in set (0.00 sec)
```

16. Create View

- In SQL, a view is a virtual table based on the result set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

>> CREATE VIEW view_table AS SELECT student_id, first_name, last_name, email, phone_number FROM student;

```
mysql>
mysql>
mysql> CREATE VIEW view_table AS SELECT student_id, first_name, last_name, e
mail, phone_number FROM student;
Query OK, 0 rows affected (0.01 sec)

mysql> desc view_table;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| student_id   | int          | NO   |     | NULL    |       |
| first_name   | varchar(50)  | NO   |     | NULL    |       |
| last_name    | varchar(50)  | NO   |     | NULL    |       |
| email        | varchar(100) | NO   |     | NULL    |       |
| phone_number | varchar(15)  | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

SQL

1. *Insert*

Used to select data from a database

- Secure Insert

>> INSERT INTO test(id, Name, City) VALUES (1, ‘Adam’, ‘Bangalore’);

```
mysql> INSERT INTO test(id,Name,City) VALUES (1,'Adam','Bangalore');
Query OK, 1 row affected (0.02 sec)

mysql> select * from test;
+------+------+-----------+
| id   | Name | City      |
+------+------+-----------+
|    1 | Adam | Bangalore |
+------+------+-----------+
1 row in set (0.00 sec)
```

- Insecure Insert

>> INSERT INTO test VALUES(2,’Bob’,’Delhi);

```
mysql> insert into test values(2,'Bob','Delhi');
Query OK, 1 row affected (0.01 sec)

mysql> select * from test;
+------+------+------------+
| id   | Name | City       |
+------+------+------------+
|    1 | Adam | Bangalore  |
|    2 | Bob  | Delhi      |
+------+------+------------+
2 rows in set (0.00 sec)
```

>> INSERT INTO test
VALUES(3,’Cathy’,’Kerala’),(4,’David’,’Pune’),(5,’Evan’,’Tamilnadu’);

```
mysql> insert into test values(3,'Cathy','Kerala'),(4,'David','Pune'),(5,'Ev
an','Tamilnadu');
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from test;
+------+-------+-----------+
| id   | Name  | City      |
+------+-------+-----------+
|    1 | Adam  | Bangalore |
|    2 | Bob   | Delhi     |
|    3 | Cathy | Kerala    |
|    4 | David | Pune      |
|    5 | Evan  | Tamilnadu |
+------+-------+-----------+
5 rows in set (0.00 sec)
```

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

---

2. *Select*

Used to select data from a database.

>> SELECT * FROM Customers;

```
mysql> SELECT * FROM Customers;
+---------+------------+-----------+------+---------+
| Cust_id | First_name | Last_name | Age  | Country |
+---------+------------+-----------+------+---------+
|       1 | John       | Doe       |   31 | USA     |
|       2 | Robert     | Luna      |   22 | USA     |
|       3 | David      | Robinson  |   22 | UK      |
|       4 | John       | Reinhardt |   25 | UK      |
|       5 | Betty      | Doe       |   28 | UAE     |
+---------+------------+-----------+------+---------+
5 rows in set (0.00 sec)
```

>> SELECT Cust_id, Age, Country FROM Customers;

```
mysql> SELECT Cust_id, Age, Country FROM Customers;
+---------+------+---------+
| Cust_id | Age  | Country |
+---------+------+---------+
|       1 |   31 | USA     |
|       2 |   22 | USA     |
|       3 |   22 | UK      |
|       4 |   25 | UK      |
|       5 |   28 | UAE     |
+---------+------+---------+
5 rows in set (0.00 sec)
```

>> SELECT DISTINCT Cust_id FROM Orders;

Gives the unique values of column id.

```
mysql> SELECT DISTINCT Cust_id FROM Orders;
+---------+
| Cust_id |
+---------+
|       4 |
|       3 |
|       1 |
|       2 |
+---------+
4 rows in set (0.01 sec)
```

>> SELECT COUNT(DISTINCT Cust_id) FROM Orders;

Gives the count of unique values of column id.

```
mysql> SELECT Count(DISTINCT Cust_id) FROM Orders;
+------------------------+
| Count(DISTINCT Cust_id) |
+------------------------+
|                      4 |
+------------------------+
1 row in set (0.00 sec)
```

3. *Where Clause*

Used to filter the records. The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

>>SELECT * FROM Orders WHERE Cust_id=4;

```
mysql> SELECT * FROM Orders WHERE Cust_id=4;
+----------+----------+--------+---------+
| Order_id | Item     | Amount | Cust_id |
+----------+----------+--------+---------+
|        1 | Keyboard | 400    |       4 |
|        2 | Mouse    | 300    |       4 |
+----------+----------+--------+---------+
2 rows in set (0.00 sec)
```

- Operators in Where clause

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

4. *MySQL AND, OR and NOT operators*

>>SELECT * FROM Orders WHERE Cust_id=4 AND Item='Keyboard';

```
mysql> SELECT * FROM Orders WHERE Cust_id=4 AND Item='Keyboard';
+----------+----------+--------+---------+
| Order_id | Item     | Amount | Cust_id |
+----------+----------+--------+---------+
|        1 | Keyboard | 400    |       4 |
+----------+----------+--------+---------+
1 row in set (0.00 sec)
```

>> SELECT * FROM Orders WHERE Cust_id=4 OR Item='Keyboard';

```
mysql> SELECT * FROM Orders WHERE Cust_id=4 OR Item='Keyboard';
+----------+----------+--------+---------+
| Order_id | Item     | Amount | Cust_id |
+----------+----------+--------+---------+
|        1 | Keyboard | 400    |       4 |
|        2 | Mouse    | 300    |       4 |
|        4 | Keyboard | 400    |       1 |
+----------+----------+--------+---------+
3 rows in set (0.00 sec)
```

>>SELECT * FROM Orders WHERE NOT Cust_id=4;

```
mysql> SELECT * FROM Orders WHERE NOT Cust_id=4;
+----------+----------+--------+---------+
| Order_id | Item     | Amount | Cust_id |
+----------+----------+--------+---------+
|        3 | Monitor  | 12000  |       3 |
|        4 | Keyboard | 400    |       1 |
|        5 | Mousepad | 250    |       2 |
+----------+----------+--------+---------+
3 rows in set (0.00 sec)
```

5. *Order By Keyword*

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

>>SELECT * FROM Orders ORDER BY Cust_id;

```
mysql> SELECT * FROM Orders ORDER BY Cust_id;
+----------+----------+--------+---------+
| Order_id | Item     | Amount | Cust_id |
+----------+----------+--------+---------+
|        4 | Keyboard | 400    |       1 |
|        5 | Mousepad | 250    |       2 |
|        3 | Monitor  | 12000  |       3 |
|        1 | Keyboard | 400    |       4 |
|        2 | Mouse    | 300    |       4 |
+----------+----------+--------+---------+
5 rows in set (0.02 sec)
```

>>SELECT * FROM Orders ORDER BY Cust_id DESC;

```
mysql> SELECT * FROM Orders ORDER BY Cust_id DESC;
+----------+----------+--------+---------+
| Order_id | Item     | Amount | Cust_id |
+----------+----------+--------+---------+
|        1 | Keyboard | 400    |       4 |
|        2 | Mouse    | 300    |       4 |
|        3 | Monitor  | 12000  |       3 |
|        5 | Mousepad | 250    |       2 |
|        4 | Keyboard | 400    |       1 |
+----------+----------+--------+---------+
5 rows in set (0.00 sec)
```

6. *NULL Values*

A field with a NULL value is a field with no value.

>>SELECT * FROM Customers WHERE Age IS NOT NULL;

```
mysql> SELECT * FROM Customers WHERE Age IS NOT NULL;
+---------+------------+-----------+------+---------+
| Cust_id | First_name | Last_name | Age  | Country |
+---------+------------+-----------+------+---------+
|       1 | John       | Doe       |   31 | USA     |
|       2 | Robert     | Luna      |   22 | USA     |
|       3 | David      | Robinson  |   22 | UK      |
|       4 | John       | Reinhardt |   25 | UK      |
|       5 | Betty      | Doe       |   28 | UAE     |
+---------+------------+-----------+------+---------+
5 rows in set (0.00 sec)
```

7. *Update Statement*

Used to modify the existing records in a table.

>>UPDATE Customers SET Age=27 WHERE Cust_id=3;

```
mysql> UPDATE Customers SET Age=27 WHERE Cust_id=3;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

8. Delete Statement

Used to delete existing records in a table. While deleting make sure to check the WHERE clause. If you omit the WHERE clause, all records in the table will be deleted.

>>DELETE FROM Customers WHERE Cust_id=5

9. Limit Statement

This clause is used to specify the number of records to return. Useful on large tables with thousand of records.

>>SELECT * FROM Customers LIMIT 2;

```
mysql> SELECT * FROM Orders LIMIT 2;
+----------+----------+--------+---------+
| Order_id | Item     | Amount | Cust_id |
+----------+----------+--------+---------+
|        1 | Keyboard | 400    |       4 |
|        2 | Mouse    | 300    |       4 |
+----------+----------+--------+---------+
2 rows in set (0.00 sec)
```

10. Min & Max Functions

>>SELECT MIN(Amount) AS SmallestAmout FROM Orders;

```
mysql> SELECT MIN(Amount) AS SmallestAmount FROM Orders;
+----------------+
| SmallestAmount |
+----------------+
| 12000          |
+----------------+
1 row in set (0.00 sec)
```

>>SELECT MAX(Amount) AS LargestAmount FROM Orders;

```
mysql>
mysql> SELECT MAX(Amount) AS LargestAmount FROM Orders;
+---------------+
| LargestAmount |
+---------------+
| 400           |
+---------------+
1 row in set (0.00 sec)
```

11. Count, Average, Sum Functions

>>SELECT COUNT(Cust_id) FROM Customers Where Country='USA';

```
mysql> SELECT COUNT(Cust_id) FROM Customers Where Country='USA';
+----------------+
| COUNT(Cust_id) |
+----------------+
|              2 |
+----------------+
1 row in set (0.01 sec)
```

>>SELECT AVG(Amount) FROM Orders;

```
mysql> SELECT AVG(Amount) FROM Orders;
+-------------+
| AVG(Amount) |
+-------------+
|        2670 |
+-------------+
1 row in set (0.00 sec)
```

>>SELECT SUM(Amount) FROM Orders;

```
mysql> SELECT SUM(Amount) FROM Orders;
+-------------+
| SUM(Amount) |
+-------------+
|       13350 |
+-------------+
1 row in set (0.00 sec)
```

12. Wild Cards (Like Operator)

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

The percent sign and the underscore can also be used in combinations!

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

>>SELECT * FROM Customers WHERE First_name LIKE 'J%';

Selects all Customers with First_name starting with 'a'.

```
mysql> SELECT * FROM Customers WHERE First_name LIKE 'J%';
+---------+------------+-----------+------+---------+
| Cust_id | First_name | Last_name | Age  | Country |
+---------+------------+-----------+------+---------+
|       1 | John       | Doe       |   31 | USA     |
|       4 | John       | Reinhardt |   25 | UK      |
+---------+------------+-----------+------+---------+
2 rows in set (0.01 sec)
```

>>SELECT * FROM Customers WHERE Last_name LIKE '%a%';

Selects all Customers with First_name having 'a' in any position.

```
mysql> SELECT * FROM Customers WHERE Last_name LIKE '%a%';
+---------+------------+-----------+------+---------+
| Cust_id | First_name | Last_name | Age  | Country |
+---------+------------+-----------+------+---------+
|       2 | Robert     | Luna      |   22 | USA     |
|       4 | John       | Reinhardt |   25 | UK      |
+---------+------------+-----------+------+---------+
2 rows in set (0.00 sec)
```

>>SELECT * FROM Customers WHERE First_name LIKE '_a%';

Selects all Customers with First name that have 'a' in the second position.

```
mysql> SELECT * FROM Customers WHERE First_name LIKE '_a%';
+---------+------------+-----------+------+---------+
| Cust_id | First_name | Last_name | Age  | Country |
+---------+------------+-----------+------+---------+
|       3 | David      | Robinson  |   27 | UK      |
+---------+------------+-----------+------+---------+
1 row in set (0.00 sec)
```

13. IN Operator

The IN operator allows you to specify multiple values in a WHERE clause. This operator is a shorthand for multiple OR condition.

>>SELECT * FROM Customers WHERE Age IN (22,25);

```
mysql> SELECT * FROM Customers WHERE Age IN (22,25);
+---------+------------+-----------+------+---------+
| Cust_id | First_name | Last_name | Age  | Country |
+---------+------------+-----------+------+---------+
|       2 | Robert     | Luna      |   22 | USA     |
|       4 | John       | Reinhardt |   25 | UK      |
+---------+------------+-----------+------+---------+
2 rows in set (0.00 sec)
```

>>SELECT * FROM Customers WHERE Age NOT IN(22,25);

```
mysql> SELECT * FROM Customers WHERE Age NOT IN(22,25);
+---------+------------+-----------+------+---------+
| Cust_id | First_name | Last_name | Age  | Country |
+---------+------------+-----------+------+---------+
|       1 | John       | Doe       |   31 | USA     |
|       3 | David      | Robinson  |   27 | UK      |
|       5 | Betty      | Doe       |   28 | UAE     |
+---------+------------+-----------+------+---------+
3 rows in set (0.00 sec)
```

14. Between Operator

Selects values within a given range. The values can be numbers, text or dates. This operator is inclusive of begin and end values are included.

>>SELECT * FROM Customers WHERE Age BETWEEN 21 AND 25;

```
mysql> SELECT * FROM Customers WHERE Age BETWEEN 21 AND 25;
+---------+------------+-----------+------+---------+
| Cust_id | First_name | Last_name | Age  | Country |
+---------+------------+-----------+------+---------+
|       2 | Robert     | Luna      |   22 | USA     |
|       4 | John       | Reinhardt |   25 | UK      |
+---------+------------+-----------+------+---------+
2 rows in set (0.00 sec)
```

>>SELECT * FROM Customers WHERE First_name BETWEEN ‘C’ AND ‘J’;

```
mysql> SELECT * FROM Customers WHERE First_name BETWEEN 'C' AND 'J';
+---------+------------+-----------+------+---------+
| Cust_id | First_name | Last_name | Age  | Country |
+---------+------------+-----------+------+---------+
|       3 | David      | Robinson  |   27 | UK      |
+---------+------------+-----------+------+---------+
1 row in set (0.00 sec)
```

15. Aliases

Used to give a table, or a column in a table, a temporary name for more readable.

>>SELECT Cust_id AS ID FROM Customers;

```
mysql> SELECT Cust_id AS ID FROM Customers;
+------+
| ID   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
+------+
5 rows in set (0.00 sec)
```

>>SELECT Cust_id, CONCAT_WS(‘,’,First_name, Last_name) AS Full_Name from Customers;

```
mysql>
mysql> SELECT Cust_id, CONCAT_WS(',',First_name, Last_name) AS Full_Name from Customers;
+---------+---------------+
| Cust_id | Full_Name     |
+---------+---------------+
|       1 | John,Doe      |
|       2 | Robert,Luna   |
|       3 | David,Robinson |
|       4 | John,Reinhardt |
|       5 | Betty,Doe     |
+---------+---------------+
5 rows in set (0.00 sec)
```

16. INNER Join

An INNER JOIN returns records that have matching values in both tables. If there's no match, the records are not included.

>>SELECT Customers.Cust_id, Customers.First_name, Orders.Item, Orders.Amount FROM Customers INNER JOIN Orders ON Customers.Cust_id=Orders.Cust_id;

```
mysql> SELECT Customers.Cust_id, Customers.First_name, Orders.Item, Orders.A
mount FROM Customers INNER JOIN Orders ON Customers.Cust_id=Orders.Cust_id;
+---------+------------+----------+--------+
| Cust_id | First_name | Item     | Amount |
+---------+------------+----------+--------+
|       4 | John       | Keyboard | 400    |
|       4 | John       | Mouse    | 300    |
|       3 | David      | Monitor  | 12000  |
|       1 | John       | Keyboard | 400    |
|       2 | Robert     | Mousepad | 250    |
+---------+------------+----------+--------+
5 rows in set (0.01 sec)
```

17. LEFT Join (or LEFT OUTER Join)

A LEFT JOIN returns all records from the left table (Customers), and the matched records from the right table (Orders). If there's no match, NULL values are shown for columns from the right table.

>>SELECT Customers.Cust_id, Customers.First_name, Orders.Item, Orders.Amount FROM Customers LEFT JOIN Orders ON Customers.Cust_id=Orders.Cust_id;

```
mysql> SELECT Customers.Cust_id, Customers.First_name, Orders.Item, Orders.A
mount FROM Customers LEFT JOIN Orders ON Customers.Cust_id=Orders.Cust_id;
+---------+------------+----------+--------+
| Cust_id | First_name | Item     | Amount |
+---------+------------+----------+--------+
|       1 | John       | Keyboard | 400    |
|       2 | Robert     | Mousepad | 250    |
|       3 | David      | Monitor  | 12000  |
|       4 | John       | Mouse    | 300    |
|       4 | John       | Keyboard | 400    |
|       5 | Betty      | NULL     | NULL   |
+---------+------------+----------+--------+
6 rows in set (0.00 sec)
```

18. RIGHT JOIN (or RIGHT OUTER JOIN)

A RIGHT JOIN returns all records from the right table (Orders), and the matched records from the left table (Customers). If there's no match, NULL values are shown for columns from the left table.

>>SELECT Customers.Cust_id, Customers.First_name, Orders.Item, Orders.Amount FROM Customers RIGHT JOIN Orders ON Customers.Cust_id=Orders.Cust_id;

```
mysql> SELECT Customers.Cust_id, Customers.First_name, Orders.Item, Orders.A
mount FROM Customers RIGHT JOIN Orders ON Customers.Cust_id=Orders.Cust_id;
+---------+------------+----------+--------+
| Cust_id | First_name | Item     | Amount |
+---------+------------+----------+--------+
|       4 | John       | Keyboard | 400    |
|       4 | John       | Mouse    | 300    |
|       3 | David      | Monitor  | 12000  |
|       1 | John       | Keyboard | 400    |
|       2 | Robert     | Mousepad | 250    |
+---------+------------+----------+--------+
5 rows in set (0.00 sec)
```

19. FULL OUTER JOIN

The FULL OUTER JOIN returns all the records when there is a match in left table or right table records. FULL OUTER JOIN and FULL JOIN are the same.

>>SELECT Customers.Cust_id, Customers.First_name, Orders.Item, Orders.Amount FROM Customers FULL OUTER JOIN Orders ON Customers.Cust_id=Orders.Cust_id;

20. UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns.
- The columns must also have similar data types.
- The columns in every SELECT statement must also be in the same order.

>>SELECT * FROM Customers UNION SELECT * FROM Orders;

>>SELECT * FROM Customers UNION ALL SELECT * FROM Orders;

It allows the duplicate values.

21. GROUP BY

This statement groups rows that have the same values into summary rows. This statement is often used with aggregate functions (COUNT(), SUM(), MAX(), MIN(), AVG()) to group the result-set by one or more columns.

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

This statement lists the number of customers in each country.

>>SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country ORDER BY COUNT(CustomerID) DESC;

This statement lists the number of customers in each country, sorted high to low.

22. HAVING CLAUSE

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

>>SELECT *column_name(s)* FROM *table_name* WHERE *condition*
GROUP BY *column_name(s)* HAVING *condition* ORDER BY *column_name(s);*

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

>>SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM (Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID) GROUP BY LastName HAVING COUNT(Orders.OrderID) > 10;

23. EXISTS Operator

This operator tests the existence of any record in a subquery. It returns TRUE if the subquery returns one or more records.

>>SELECT *column_name(s)* FROM *table_name* WHERE EXISTS
(SELECT *column_name* FROM *table_name* WHERE *condition*);

>>SELECT SupplierName FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.Supplier ID = Suppliers.supplierID AND Price = 22);

The following SQL statement returns TRUE and lists the suppliers with a product price equal to 22:

24. ANY Operators

This means that the condition will be true if the operation is true for any of the values in the range.

>>SELECT ProductName FROM Products WHERE ProductID = ANY
(SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity equal to 10 (this will return TRUE because the Quantity column has some values of 10).

25. ALL Operators

This means that the condition will be true only of the operation is true for all values in the range.

>>SELECT ProductName FROM Products WHERE ProductID = ALL
(SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

The following SQL statement lists the ProductName if ALL the records in the OrderDetails table has Quantity equal to 10. This will of course return FALSE because the Quantity column has many different values (not only the value of 10).

26. SELECT INTO Statement

This statement copies data from one table into a new table.

>>SELECT *column1*, *column2*, *column3*, ... INTO *newtable* [IN *externaldb*]
FROM *oldtable* WHERE *condition;*

>>SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;

Windows Functions

This allows you to perform calculations across a specific set of rows, known as a window while keeping the detailed data intact. OVER clause is used with window functions to define that window. OVER clause does two things:

- Partitions rows to form a set of rows. (PARTITION BY clause is used).
- Orders rows within those partitions into a particular order. (ORDER BY clause is used).

Syntax:

```
>>SELECT  col_name1,
        window_function(col_name2)
        OVER ([PARTITION BY col_name1] [ORDER BY col_name3]) AS new_col
        FROM table_name;
```

window_function: an aggregate or ranking function

col_name1: column to be selected

col_name2: column on which window function is to be applied

col_name3: column on whose basis partition of rows is to be done.

new_col: Name of new column

table_name: Name of table

Sample Dataset: `Sales`

SalesID	Employee	Region	Month	Sales
1	Alice	North	Jan	500
2	Bob	South	Jan	300
3	Alice	North	Feb	600
4	Bob	South	Feb	400
5	Charlie	North	Jan	700
6	Charlie	North	Feb	800

1. **ROW_NUMBER() – Ranking sales by employee in each region.**
   - Gives unique ranks (no ties)

>> SELECT SalesID, Employee, Region, Month, Sales,

ROW_NUMBER() OVER (PARTITION BY Region ORDER BY Sales DESC) AS Rank

FROM Sales;

Output:

SalesID	Employee	Region	Month	Sales	Rank
6	Charlie	North	Feb	800	1
5	Charlie	North	Jan	700	2
3	Alice	North	Feb	600	3
1	Alice	North	Jan	500	4
4	Bob	South	Feb	400	1
2	Bob	South	Jan	300	2

2. **SUM() – Running total of sales for each employee**
   - Used for cumulative totals

>> SELECT SalesID, Employee, Region, Month, Sales,

SUM(Sales) OVER (PARTITION BY Employee ORDER BY Month) AS RunningTotal

FROM Sales;

Output:

SalesID	Employee	Region	Month	Sales	RunningTotal
1	Alice	North	Jan	500	500
3	Alice	North	Feb	600	1100
2	Bob	South	Jan	300	300
4	Bob	South	Feb	400	700
5	Charlie	North	Jan	700	700
6	Charlie	North	Feb	800	1500

3. **LEAD() and LAG() – Comparing sales of consecutive months**
   - Compare values across rows within a partition

```
>> SELECT SaledID, Employee, Month, Sales,
       LAG(Sales) OVER (PARTITION BY Employee ORDER BY Month) AS PreviousMonthSales
       LEAD(Sales) OVER (PARTITION BY Employee ORDER BY Month) AS NextMonthSales
       FROM Sales;
```

Output:

SalesID	Employee	Month	Sales	PreviousMonthSales	NextMonthSales
1	Alice	Jan	500	NULL	600
3	Alice	Feb	600	500	NULL
2	Bob	Jan	300	NULL	400
4	Bob	Feb	400	300	NULL
5	Charlie	Jan	700	NULL	800
6	Charlie	Feb	800	700	NULL

4. **RANK() vs DENSE_RANK()**
   - RANK(): Skips ranks for ties.
   - DENSE_RANK(): No skipped ranks for ties.

```
>> SELECT SalesID, Employee, Region, Month, Sales,
       RANK() OVER (PARTITION BY Region ORDER BY Sales DESC) AS Rank,
       DENSE_RANK() OVER (PARTITION BY Region ORDER BY Sales DESC) AS DenseRank
       FROM Sales;
```

Output:

SalesID	Employee	Region	Month	Sales	Rank	DenseRank
6	Charlie	North	Feb	800	1	1
5	Charlie	North	Jan	700	2	2
3	Alice	North	Feb	600	3	3
1	Alice	North	Jan	500	4	4
4	Bob	South	Feb	400	1	1
2	Bob	South	Jan	300	2	2

5. Moving Averages – Calculate the moving average of sales for each employee.

>> SELECT SalesID, Employee, Month, Sales,

AVG(Sales) OVER (PARTITION BY Employee ORDER BY Month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS MovingAvg

FROM Sales;

Output:

SalesID	Employee	Month	Sales	MovingAvg
1	Alice	Jan	500	500.00
3	Alice	Feb	600	550.00
2	Bob	Jan	300	300.00
4	Bob	Feb	400	350.00
5	Charlie	Jan	700	700.00
6	Charlie	Feb	800	750.00

6. Calculating Percentile Ranks

>> SELECT SalesID, Employee, Region, Sales,

PERCENT_RANK() OVER (PARTITION BY Region ORDER BY Sales) AS PercentileRank

FROM Sales;

Output:

SalesID	Employee	Region	Sales	PercentileRank
1	Alice	North	500	0.0000
3	Alice	North	600	0.3333
5	Charlie	North	700	0.6667
6	Charlie	North	800	1.0000
2	Bob	South	300	0.0000
4	Bob	South	400	1.0000

7. **Difference Between Current and Previous Sales**

```
>> SELECT SalesID, Employee, Month, Sales,
        Sales – LAG(Sales) OVER (PARTITION BY Employee ORDER BY Month) AS SalesDifference
FROM Sales;
```

Output:

SalesID	Employee	Month	Sales	SalesDifference
1	Alice	Jan	500	NULL
3	Alice	Feb	600	100
2	Bob	Jan	300	NULL
4	Bob	Feb	400	100
5	Charlie	Jan	700	NULL
6	Charlie	Feb	800	100