SQL

SQL is a language that interacts with databases.
It is the standard language for dealing with relational databases.
It is used to read, update, retrieve, manipulate & store data on a database.
The data is stored in a database and organized by tables.
We write and send queries in SQL to the databases, which receives these queries and make changes. These queries can be traced back to see who made what changes to which table.

SQL is best suited for Larger Datasets (Billions of rows). It doesn't slow down as Excel does. Preparing data for further analysis in another software. We can save and share queries.

Data is a collection of facts, figures and values from different sources, which acts as an information. The sources - Financial, Scientific, Statistical. Transport, Metrological, Geographical etc. 2.5×10^{18} Bytes Everyday.

Database – It is an organized collection of data. It contains one or more tables. It is a location where data is stored in certain format. Database is an integrated collection of related information along with the details so it is available to the several users for the different applications.

Relational Database – A database structured to recognize relations between stored items of information. It uses tables to store information. A relational database is a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables. Ex – Employee Table, Product Table.

Database Management System - It is a collection of programs which enables its users to access database, manipulate data, and help in representation of data.

Types of Database:
1) Distributed Database……2) Object Oriented Database…..3) Centralized Database…..
4) Operational Database…...5) Graph Database….. ……..6) NoSQL Database…..
7) Cloud Database…..**8) Relational Database**

Popular Database Management Program:
1) mongo DB(For NoSQL Database)…..2) MS-Access…..3) MS-SQL Server(For Relational Database, Not Free)…..4) MY SQL(For Relational Database, Open-source & Free)…..5) Oracle (Relational Database)

We can use **MySQL Workbench** (Flexible Features, but some companies don't think it reliable) or we can use **MySQL Command Line Client** (Some companies think it is reliable)

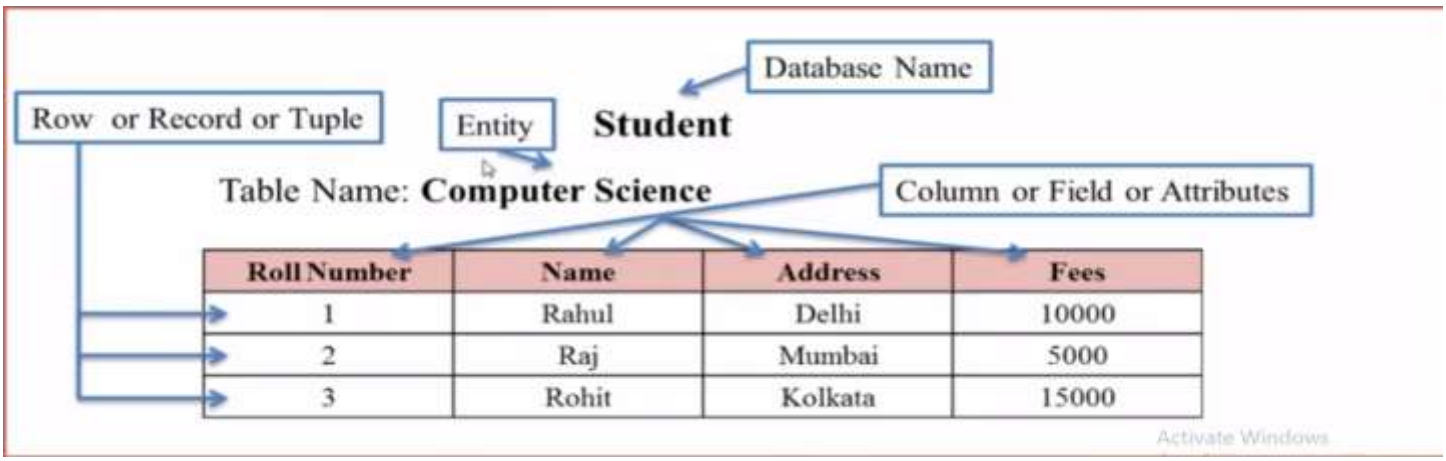
Connect MySQL Database to Python :

In command prompt –
pip install mysql-connector-python # Install DB Connector - It connects python to mysql db

```
# Create MySQL Connection
import mysql.connector as sql
connection = sql.connect        # Use connect function with 3 parameters host,user,password
( host= 'localhost', user= 'root', password: '99900' )
or, ( host= 'localhost', user= 'root', password: '99900' , database='customers')  # To use
customer database

# Create and Show Databases
cursor = connection.cursor( )                    # cursor( ) - to execute mysql queries
cursor.execute("Create database new_db")  # Creating a new DB with cursor.execute statement
cursor.execute("Show Databases")             # Showing the databases
for x in cursor:
        print(x)

# Create and Show Tables
cursor = connection.cursor( )                    # cursor( ) - to execute mysql queries
cursor.execute("CREATE TABLE new_table(id int(3), name varchar(20), city varchar(20))")
cursor.execute("Show Tables")
for x in cursor:
        print(x)

# Insert Records in Table
cursor.execute("Insert Into new_table(id, name, city) values(20,'ram','delhi')")

# Insert Multiple Records
query = "Insert Into new_table (id, name, city) values (%s, %s)"
values = [(1,'A', 'X'), (2,'B','Y'), (3,'C','Z')]
cursor.execute(query,values)

#Show Records of Table
cursor.execute("Select * from new_table")
cursor.fetchall()                                # To show all records
or, cursor.fetchone()                            # To show record one by one

# Showing Records with a condition
cursor.execute("Select * from new3 where name='A'")
cursor.fetchall()

# Showing particular columns
cursor.execute("Select col_name from new_table")
cursor.fetchall()

# Deleting a Table
cursor.execute("Drop Table new_table")
```

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views



CREATE DATABASE

Syntax: CREATE DATABASE database_name;
Example: create database School;

Guidelines for creation of Database:
1. Name should start with an alphabet.
2. Blank space and single quotes are not allowed.
3. Reserve words of that RDBMS/DBMS cannot be used as database name.
4. SQL is not case sensitive language. Means CREATE DATABASE and create database both are same.

Using Database

Syntax: USE database_name;
Example: USE School;

Remove Database

Syntax: DROP DATABASE database_name;
Example: DROP DATABASE School;

Backup Database

To create a full backup of the existing SQL database.

Syntax:
BACKUP DATABASE database_name
TO DISK = ‘filepath’;
Example: BACKUP DATABASE School
TO DISK = ‘D:\backups\testDB.bak’;

To create a differential backup which only backs up the parts of the database that have changed since the last full database backup. A differential back up reduces the back-up time (since only the changes are backed up).
Syntax:
BACKUP DATABASE database_name
TO DISK = ‘filepath
WITH DIFFERENTIAL;
Example: BACKUP DATABASE School
TO DISK = ‘D:\backups\testDB.bak’
WITH DIFFERENTIAL;

CREATE TABLE

A Table is a collection of data in a tabular form.

Syntax 1 : CREATE TABLE table-name
(Col_name1 data-type (size),
  Col_name2 data-type (size),
  Col_name3 data-type (size));
Example 1: CREATE TABLE Students
(Name varchar (20),
  Roll_No number (5),
  Address varchar (30));

Guidelines for creation of Table:
1. Table name should start with alphabet.
2. In table name, blank spaces and single quotes are not allowed.
3. Reserve words of that RDBMS/DBMS cannot be used as table name.
4. Proper data-type and size should be used.
5. Unique column name should be specified.

Create Table Using Constraints

SQL Constraints are used to specify rules for the data in a table.
Column Level constraints apply to a column. Table Level constraints apply to the whole table.

Types of 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 - Uniquely identifies a row/record in another table.
CHECK - Ensures that all values in a column satisfy a specific condition.
DEFAULT - Sets a default value for a column when no value is specified
INDEX - Used to create and retrieve data from the database very quickly

Syntax : CREATE TABLE table-name
(Col_name1 data-type (size) [constraints],
  Col_name2 data-type (size) [constraints],
  Col_name3 data-type (size) [constraints]);
Example : CREATE TABLE Students
(Name varchar (20) NOT NULL,
  Roll_No number (5) PRIMARY KEY,
  Address varchar (30))
  UNIQUE (City);

Auto Increment Field

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
Syntax : CREATE TABLE table-name
(Col_name1 data-type (size) AUTO_INCREMENT,
  Col_name2 data-type (size) [constraints]);

Create Table Using Another Table

Syntax: CREATE TABLE new-table-name AS
SELECT Col_1, Col_2
FROM existing-table-name;

Delete Table

Syntax: DROP TABLE table-name;
Example: DROP TABEL Students;

Truncate Table

It is used to delete the data inside the table, but not the table itself.
Syntax: TRUNCATE TABLE table-name;

ALTER TABLE

It is used to add, delete, or modify columns in an existing table. It is also used to add or drop various constraints on an existing table.

ADD Column - To add a new column in the existing table.
ALTER TABLE table-name
ADD column-name datatype(size);

DROP Column - To delete a column from the existing table.
ALTER TABLE table-name
DROP COLUMN column-name;

Alter/Modify Column Datatype - To change the data-type of a column in a table.
ALTER TABLE table-name
MODIFY COLUMN column-name datatype;

Describe Table

Syntax: DESC table-name;
Example: DESC Students;

It is used to describe a table. DESC describes the structure of the table not the information (rows) inside table. DESC is short form of describe.

```
mysql> CREATE DATABASE my_db;
Query OK, 1 row affected (0.00 sec)

mysql> USE my_db;
Database changed
mysql> CREATE TABLE my_tab
    -> (
    -> name varchar (50),
    -> roll int (4)
    -> );
Query OK, 0 rows affected (0.51 sec)

mysql> DESC my_tab;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| name  | varchar(50) | YES  |     | NULL    |       |
| roll  | int(4)      | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.03 sec)

mysql>
```

Show Database and Show Table

Syntax: SHOW DATABASES;
This command is used to show all the databases names.

Syntax: SHOW TABLES;
This command is used to show all the tables of the current database.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| db                 |
| my_db              |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.00 sec)

mysql>
```

```
mysql> SHOW TABLES;
+----------------+
| Tables_in_my_db |
+----------------+
| my_tab         |
+----------------+
1 row in set (0.00 sec)

mysql>
```

DATA-TYPES

INT or INTEGER :

1. It holds whole number between -32,768 and 32767 either it negative or positive.
2. It cannot hold decimal numbers.
3. The maximum number of digits specified in parenthesis.

Syntax: COLUMN_NAME INT (size);
Example: roll INT (5);

DEC or DECIMAL :

It holds fixed point numbers. Size is the total number of digits and p is the total number of digits after decimal. The decimal point and the negative '-' sign are not counted in size. If p is '0', values have no decimal point. The maximum number of size for Decimal is 65 and for p 30. If p omitted, the default is '0'.
If size is omitted the default is 10.

Syntax: Column_name DECIMAL (size, p);
Example: price DECIMAL (7,2);

CHAR or CHARACTER

It holds a fixed length string (can contain letters, numbers and special characters).
The fixed size is specified in parenthesis. It can store up-to 255 characters.

Syntax: Column_name CHAR (20);
Example: name CHAR (20);

VARCHAR or VARIABLE CHARACTER

It holds a variable length string (can contain letters, numbers, and special characters).
The maximum size is specified in parenthesis. It can store up to 255 characters.
Note : If we put a greater value than 255 it will be converted to a text type.

Syntax: Column_name VARCHAR (size);
Example : name VARCHAR (50);

DATE

It displays Date values in yyyy-mm-dd format.

Syntax: Column_Name DATE;
Example : age DATE;

DATETIME

It displays DATETIME values in yyyy-mm-dd hh:mm:ss format.
Syntax: Column_name DATETIME;
Example: Date_of_join DATETIME;

TIMESTAMP

It also displays date and time (for current date and time).
Syntax: Column_name TIMESTAMP;
Example: login_dt TIMESTAMP;

Example

Stu_id	Name	Address	DOB	Fees

Stu_id — INT
Name — VARCHAR
Address — TEXT
DOB — DATE
Fees — DEC

DATE

MySQL comes with the following data types for storing a date or a date/time value in the database:
DATE - format : YYYY-MM-DD
DATETIME - format: YYYY-MM-DD HH:MI:SS
TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
YEAR - format YYYY or YY

We can compare two dates easily if there is no time component involved! To keep our queries simple and easy to maintain, do not allow time components in the dates!

Syntax: SELECT * FROM table-name WHERE date-column='2008-11-21';

VIEW

A view is a virtual table, which contains rows and columns just like a real table.

Syntax:
CREATE VIEW *view_name* AS
SELECT *column1*, *column2* FROM *table_name*
WHERE *condition*;

Syntax: To see a created VIEW
SELECT * FROM view_name;

Syntax: To update a View using 'CREATE OR REPLACE' command
CREATE OR REPLACE VIEW existed_*view_name* AS
SELECT *column1*, *column2*, column3 FROM *table_name* (column3 is added to the existed view)
WHERE *condition*;

Syntax: To delete a View
DROP VIEW view_name;

DML - Data Manipulation Language

Type 'help contents' in cmd to show which command is within which category.

```
categories:
   Account Management
   Administration
   Compound Statements
   Data Definition
   Data Manipulation
   Data Types
   Functions
   Functions and Modifiers for Use with GROUP BY
   Geographic Features
   Help Metadata
   Language Structure
   Plugins
   Procedures
```

```
   Data Manipulation
   Data Types
   Functions
   Functions and Modifiers for Use with GROUP BY
   Geographic Features
   Help Metadata
   Language Structure
   Plugins
   Procedures
   Storage Engines
   Table Maintenance
   Transactions
   User-Defined Functions
   Utility

mysql> help data manipulation
```

INSERT INTO

This statement is used to insert a new record/row/tuple into a table.

Syntax: INSERT INTO table-name (column1, column2, column3, column4 ...)
VALUES (value1,"value2","value3", value4 ...);
Example: INSERT INTO Students (roll, name, address, mobile)
VALUES (39, ‘Rohit’, ‘Maujpur’, 99900);

Rules :

- Column and Value order should be same.
- Any value that goes into a VARCHAR, CHAR, TEXT, DATE, DATETIME, TIMESTAMP or TIME column has single quotes around it.
- There is no need of quotes for numeric values (INT or DEC).

INSERT INTO Without Specifying Column Name

Syntax: INSERT INTO table-name VALUES (value1, ‘value2’, ‘value3’, value4)
Example: INSERT INTO Students VALUES (39, ‘rohit’, ‘Maujpur’, 8700)

Rules**:**

- The values order should be same as column.
- We need to insert record for each column we cannot leave any column.
- If the values order is not same as column than it shows an error “Column count doesn’t match value count”. Ex – INSERT INTO Students VALUES (39, ‘rohit’, 87000)

INSERT INTO With Changing the Column Order

Syntax: INSERT INTO table-name (column2, column1, column4, column3 ...)
VALUES (‘value2’, value1, value4, ‘value3’ ...);
Example: INSERT INTO Students (name, roll, mobile, address)
VALUES (‘Rohit’, 39, 99900, ‘Maujpur’);

INSERT INTO Insert Data In Only Specified Columns

Syntax: INSERT INTO table-name (column1, column2, column4)
VALUES (value1,"value2", value3);
Example: INSERT INTO Students (roll, name, mobile)
VALUES (39, ‘rohit’, 99900);

INSERT INTO Multiple Rows – To insert multiple records at one time.

Syntax: INSERT INTO table-name (column1, column2, column3, column4)
VALUES (value1,"value2","value3", value4),
(value1,"value2","value3", value4);
Example: INSERT INTO Students (roll, name, address, mobile)
VALUES (11, ‘Rohit’, ‘Maujpur’, 99900)
VALUES (22, ‘Ram, ‘India’, 8700);

SELECT

The Select statement is used to select data from a database and retrieve information.
We can select all data or some column records.

Select All Columns From The Table

Syntax: SELECT * FROM table-name;
Example: SELECT * FROM Students;

Select Particular Columns From The Table

Syntax: SELECT Col_name1, Col_name3 FROM table-name;
Example: SELECT Roll, Address FROM Students;

Select Distinct Values From A Column

Syntax: SELECT DISTINCT Col_name FROM table-name;
Example: SELECT DISTINCT City FROM Students;

SELECT With LIMITS

Select Limit is used to specify certain number of records to display.

MY-SQL :
Syntax: SELECT col_name1, col_name2 FROM table_name
LIMIT records_number;
Example: SELECT Roll, Name FROM Students
LIMIT 3;
Example: SELECT * FROM Students
LIMIT (1,5);
Example: SELECT * FROM Students
WHERE CITY = ‘Delhi’
LIMIT 5;

SQL Server / MS Access Syntax:
SELECT TOP number | percent column_name(s) FROM table-name
WHERE condition;

Oracle :
SELECT Col_name FROM Table-name
WHERE ROWNUM <=number;

SINGLE QUOTES PROBLEM

INSERT INTO Students (2, ‘R.G’s company’, 33445)

To solve this problem there are two ways:

- Use Backslash - ‘R.G\’s company’
- Use Two Time Single Quote - ‘R.G’’s company’

OPERATORS

These operators are used during WHERE query.
= , != , > , < , >= , <= , BETWEEN , LIKE, IN

WHERE

WHERE is used to search for specific data.

WHERE Clause and Equal Operator

Select Data From All Columns

Syntax: SELECT * FROM table_name
WHERE col_name operator "value";
Example: SELECT * FROM Students
WHERE Name = ‘Rohit’;

Select Specific Data From Specific Column

Syntax: SELECT col_name FROM table_name
WHERE col_name operator 'value';
Example: SELECT Name FROM Students
WHERE Roll = 3;

WHERE Clause and Not Equal Operator

Syntax: SELECT * FROM table-name
WHERE Col_name != value;
Example: SELECT * FROM Students
WHERE Roll != 38;

WHERE Clause and Greater Than / Less Than Operator

Syntax: SELECT * FROM Students
WHERE col_name </> value;
Example: SELECT * FROM Students
WHERE Roll < 7;
Exmaple: SELECT * FROM Students
WHERE Name > R;

WHERE Clause and Greater Than = & Less Than = Operator

Syntax: SELECT * FROM table-name
WHERE col_name >= / <= value;
Example: SELECT * FROM Students
WHERE Address >= ‘D’;
ExampleL SELECT * FROM Students
WHERE Address <= ‘D’;

NULL VALUES

IS NULL – This is used to select only the records with Null values in the column.

Syntax: SELECT * FROM table-name
WHERE col_name IS NULL;
Example: SELECT * FROM Students
WHERE Address IS NULL;

IS NOT NULL – This is used to select only the records with NO NULL values in the column.

Syntax: SELECT * FROM table-name
WHERE col_name IS NOT NULL
Example: SELECT * FROM Students
WHERE Mobile IS NOT NULL;

UPDATE

This command is used to modify the rows in a table.
The WHERE clause specifies which record(s) is to be updated. If we omit the WHERE clause, all records in the table will be updated!

Syntax: UPDATE table-name
SET Col_1 = Value_1
SET Col_2 = Value_2
WHERE condition;

Example: UPDATE Students
SET Name = Shiva
SET Mobile = 3437
WHERE Roll = 379;

DELETE

This command is used to delete the rows that are no longer required from the database tables.
The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

Syntax: DELETE FROM table-name
WHERE condition;
Example: DELETE FROM Students
WHERE Roll = 389;

AND OPERATOR

The AND operator displays a record if both the first condition and the second condition are true.

Syntax: SELECT * FROM table-name
WHERE Col_name1 = "value"
AND Col_name2 = "value"
Example: SELECT * FROM Students
WHERE Roll = 39
AND Name='Rohit';

OR OPERATOR

The OR operator displays a record if either the first condition OR the second condition is true.

Syntax: SELECT * FROM table-name
WHERE Col_name1 = "value"
OR Col_name2 = "value";
Example: SELECT * FROM Students
WHERE Roll = 39
OR Name = 'Ram';

NOT OPERATOR

The NOT operator displays a record if the first condition is not true.

Syntax: SELECT * FROM table-name
WHERE NOT Col_name = 'value';

Example: SELECT * FROM Students
WHERE NOT Name = ‘Rohit’;

COMBINATION OF AND & OR OPERATOR

Syntax: SELECT * FROM table-name
WHERE col_name1 = "value"
AND (col_name2 = "value" OR col_name3 = "value");
Example: SELECT * FROM Students
WHERE Address = ‘Delhi’
AND (Roll = 39 OR Name = ‘Rohit’);

IN OPERATOR

The IN operator is used to specify multiple values inside the WHERE clause. It acts as a short for multiple OR.

Syntax: SELECT * FROM table-name
WHERE Col_name IN ("Value1","Value2"...);
Example: SELECT * FROM Students
WHERE Name IN (‘Rohit’, ‘Ram’);
Example: SELECT * FROM Students
WHERE Name IN (SELECT Name FROM Students2);

NOT IN OPERATOR

Syntax: SELECT * FROM table-name
WHERE Col_name NOT IN (“Value” , “Value”);
Example: SELECT * FROM Students
WHERE Address NOT IN (‘Delhi’);

AGGREGATE FUNCTIONS

COUNT – This function returns the number of rows that match specified criteria.
Note: NULL values are not counted.

Syntax: SELECT COUNT (Col_name)
FROM table-name;
Example: SELECT COUNT (Roll)
FROM Students;

AVERAGE – This function returns the average value of a numeric column.

Syntax: SELECT AVG (Col_name)
FROM table-name;
Example: SELECT AVG (Marks)
FROM Students;

SUM – This function returns the total sum of a numeric column.

Syntax: SELECT SUM (Col_name)
FROM table-name;
Example: SELECT SUM (Marks)
FROM Students;

MIN – This function returns the minimum value of the selected column.

Syntax: SELECT MIN (Col_name)
FROM table-name;
Example: SELECT MIN (Marks)
FROM Students;

MAX – This function returns the largest value of the selected column.

Syntax: SELECT MAX (Col_name)
FROM table-name;
Example: SELECT MAX (Marks)
FROM Students;

BETWEEN

The BETWEEN operator selects the values which are in the range. The values can be numbers, text or dates. It includes the Start and End value.

BETWEEN Numbers

Syntax: SELECT * FROM table-name
WHERE Col_name BETWEEN Value1 AND Value2;
Example: SELECT * FROM Students
WHERE Roll BETWEEN 2 AND 40;
Example: SELECT Name, Address FROM Students
WHERE Roll BETWEEN 2 AND 40;

BETWEEN Text

Syntax: SELECT * FROM table-name
WHERE Col_name BETWEEN ‘Value1’ AND ‘Value2’;

Example: SELECT * FROM Students
WHERE Name BETWEEN ‘A’ AND ‘S’;

BETWEEN Dates

Syntax: SELECT * FROM table-name
WHERE date_col_name BETWEEN ‘yyyy/mm/dd’ AND ‘yyyy/mm/dd’;

BETWEEN with IN

Syntax: SELECT * FROM table-name
WHERE (Col_name BETWEEN ‘Value1’ AND ‘Value2’)
AND col_name IN (Value1 AND Value2);
Example: SELECT * FROM Students
WHERE (Roll BETWEEN 3 AND 40)
AND Address IN (‘Delhi’ AND ‘Mumbai’);

BETWEEN with NOT IN

Syntax: SELECT * FROM table-name
WHERE (Col_name BETWEEN ‘Value1’ AND ‘Value2’)
AND NOT col_name IN (Value1 AND Value2);
Example: SELECT * FROM Students
WHERE (Roll BETWEEN 3 AND 40)
AND NOT Address IN (‘Delhi’ AND ‘Mumbai’);

NOT BETWEEN

The NOT BETWEEN operator selects the values which are not in the range. The values can be numbers, text or dates.

NOT BETWEEN Numbers

Syntax: SELECT * FROM table-name
WHERE Col_name NOT BETWEEN Value1 AND Value2;
(including value1 and value2 depends on RDBMS/DBMS, MySQL does not include)
Example: SELECT * FROM Students
WHERE Roll NOT BETWEEN 2 AND 40;
Example: SELECT Name, Address FROM Students
WHERE Roll NOT BETWEEN 2 AND 40;

NOT BETWEEN Text

Syntax: SELECT * FROM table-name
WHERE Col_name NOT BETWEEN ‘Value1’ AND ‘Value2’;
Example: SELECT * FROM Students
WHERE Name NOT BETWEEN ‘A’ AND ‘S’;

NOT BETWEEN Dates

Syntax: SELECT * FROM table-name
WHERE date_col_name NOT BETWEEN ‘yyyy/mm/dd’ AND ‘yyyy/mm/dd’;

GROUPBY

It is used in SQL to arrange the identical data into groups with the help of some functions.

Syntax: SELECT Col_name(s)
FROM table-name
WHERE condition
GROUP BY Col_name(s);
Example: SELECT COUNT(Roll), City
FROM Students
GROUP BY City;

HAVING

It is used to place conditions where we need to decide which group will be the part of final result-set.

Syntax: SELECT Col_name(s)
FROM table-name
WHERE condition
GROUP BY Col_name(s)
HAVING condition;
Example: SELECT Name, SUM (Marks)
FROM Students
GROUP BY Name
HAVING SUM (Marks)>500;

ORDER BY

This keyword is used to sort the result-set in ascending or descending order.

Syntax: SELECT Col_name1, Col_name2
FROM table-name
ORBER BY Col-name1, Col_name2…….ASC|DESC;

Example: SELECT City, Name
FROM Students
ORDER BY City DESC;
Example: SELECT * FROM Students
ORDER BY City, Name DESC;

ALLIASES

SQL aliases are used to give a table or column of the table, a temporary name. Aliases are often used to make column names more readable. An alias only exists for the duration of the query.
Note**:** It requires double quotation marks or square brackets if the alias name contains spaces:

Synatax: SELECT Col-name AS Alias_name
FROM table-name;
Example: SELECT Roll AS Roll_No. , Name AS Student_Name
FROM Students;

Syntax: SELECT Col_name(s)
FROM table-name AS Alias_name;
Example: SELET Roll, Name
FROM Students AS S;

Example: SELECT Name, CONCAT (Address, ', ', Postal Code, ', ', City, ', ‘, Country) AS Address
FROM Students;

JOIN

Joins in SQL are commands which are used to combine rows from two or more tables, based on a related column between those tables.

Employee Table

EmpID	EmpFname	EmpLname	Age	EmailID	PhoneNo	Address
1	Vardhan	Kumar	22	abc	1234	Delhi
2	Himani	Sharma	32	def	5678	Mumbai
3	Aaayushi	Shreshth	24	ghi	9101	Kolkata
4	Hemanth	Sharma	25	jkl	111213	Bengaluru
5	Swatee	Kapoor	26	mno	141516	Hyderabad

Project Table

ProjectID	EmpID	ClientID	ProjectName	ProjectStart
111	**1**	3	Project1	Jan
222	**2**	1	Project2	Feb
333	**3**	5	Project3	March
444	**3**	2	Project4	Mar

555	**5**	4	Project5	Apr
666	**9**	1	Project6	April
777	**7**	2	Project7	May
888	**8**	3	Project8	Jun

INNER JOIN

This type of join returns those records which have matching values in both tables.

Syntax:
SELECT Table1.Col1, Table1.Col2, Table2.Col1….
FROM Table1
INNER JOIN Table2
ON Table1.MatchingCol = Table2.MatchingCol;

Inner Join

EmpID	EmpFname	EmpLname	ProjectID	ProjectName
1	Vardhan	Kumar	111	Project1
2	Himani	Sharma	222	Project2
3	Aaayushi	Shreshth	333	Project3
3	Aaayushi	Shreshth	444	Project4
5	Swatee	Kapoor	555	Project5

Example:
SELECT Employee.EmpID, Employee.EmpFname, Employee.EmpLname, Project.ProjectID, Project.ProjectName
FROM Employee
INNER JOIN Project
ON Employee.EmpID = Project.EmpID;

Example: Inner Join on 3 tables
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);

LEFT JOIN (Left Outer Join)

All records of left table and corresponding records of right table.
This join returns all the records from the left table and also those records which satisfy a condition from the right table. For the records having no matching values in the right table, the output will contain NULL values.

Syntax:
SELECT Table1.Col1, Table1.Col2, Table2.Col1….
FROM Table1
LEFT JOIN Table2
ON Table1.MatchingCol = Table2.MatchingCol;

Left Join

EmpFname	EmpLname	ProjectID	ProjectName
Vardhan	Kumar	111	Project1
Himani	Sharma	222	Project2
Aaayushi	Shreshth	333	Project3
Aaayushi	Shreshth	444	Project4
Swatee	Kapoor	555	Project5
Hemanth	Sharma	NA	NA

Example:

```
SELECT Employee.EmpFname, Employee.EmpLname, Project.ProjectID, Project.ProjectName
FROM Employee
LEFT JOIN Project
ON Employee.EmpID = Project.EmpID;
```

RIGHT JOIN (Right Outer Join)

All records of right table and corresponding records of left table.
This join returns all the records from the right table and also those records which satisfy a condition from the left table.
For the records having no matching values in the left table, the output will contain NULL values.

Syntax:

```
SELECT Table1.Col1, Table1.Col2, Table2.Col1….
FROM Table1
RIGHT JOIN Table2
ON Table1.MatchingCol = Table2.MatchingCol;
```

Right Join

EmpFname	EmpLname	ProjectID	ProjectName
Vardhan	Kumar	111	Project1
Himani	Sharma	222	Project2
Aaayushi	Shreshth	333	Project3
Aaayushi	Shreshth	444	Project4
Swatee	Kapoor	555	Project5
NA	NA	666	Project6
NA	NA	777	Project7
NA	NA	888	Project8

Example:

```
SELECT Employee.EmpFname, Employee.EmpLname, Project.ProjectID, Project.ProjectName
FROM Employee
RIGHT JOIN Project
ON Employee.EmpID = Project.EmpID;
```

FULL JOIN (Full Outer Join)

All records of left and right table.
This join returns all those records which either has a match in the left table or right table.
For the records having no matching value, the output will contain NULL values.

Syntax:
```
SELECT Table1.Col1, Table1.Col2, Table2.Col1….
FROM Table1
LEFT JOIN Table2
ON Table1.MatchingCol = Table2.MatchingCol;
UNION
SELECT Table1.Col1, Table1.Col2, Table2.Col1….
FROM Table1
RIGHT JOIN Table2
ON Table1.MatchingCol = Table2.MatchingCol;
```

Full Join

EmpFname	EmpLname	ProjectID
Vardhan	Kumar	111
Himani	Sharma	222
Aaayushi	Shreshth	333
Aaayushi	Shreshth	444
Hemanth	Sharma	NA
Swatee	Kapoor	555
NA	NA	666
NA	NA	777
NA	NA	888

Example:
```
SELECT Employee.EmpFname, Employee.EmpLname, Project.ProjectID
FROM Employee
LEFT JOIN Project
ON Employee.EmpID = Project.EmpID;
UNION
SELECT Employee.EmpFname, Employee.EmpLname, Project.ProjectID
FROM Employee
RIGHT JOIN Project
ON Employee.EmpID = Project.EmpID;
```

SELF JOIN

A self-JOIN is a regular join, but the table is joined with itself.

Syntax:
```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

Example:
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City;

LIKE OPERATOR

It is used to search for a specified pattern in a column.

Two wildcards used with LIKE operator :
% - The percent sign represents zero, one, or multiple characters.
_ - The underscore represents a single character.

Syntax:
SELECT * FROM table-name
WHERE Col_name LIKE pattern;

‘a%’ - Finds any values that start with "a".
‘%a’ - Finds any values that end with "a".
‘%cd%’ - Finds any values that have "cd" in any position
‘a%z’ - Finds any values that start with "a" and ends with "z"
‘_elhi%’ - Finds any values starts with any character, followed by ‘_elhi’.
‘a_%’ - Finds any values that start with "a" and are at least 2 characters in length.
‘a__%’ - Finds any values that start with "a" and are at least 3 characters in length.

Using [charlist] wildcard
Syntax: Starting with any value b,s or p.
SELECT * FROM table-name
WHERE Col-name LIKE ‘[bsp]%’;

Using [!charlist] wildcard
Syntax: Not starting with any value b,s or p.
SELECT * FROM table-name
WHERE Col-name LIKE ‘[!bsp]%’; or WHERE Col-name NOT LIKE ‘[bsp]’;

EXISTS OPERATOR

The EXISTS operator is used to test for the existence of any record in a subquery.

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

ANY , ALL OPERATOR

The ANY operator returns true if any of the subquery values meet the condition.

Syntax:
SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name operator* ANY
(SELECT *column_name* FROM *table_name* WHERE *condition*);

The ALL operator returns true if all of the subquery values meet the condition.

Syntax:
SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name operator* ALL
(SELECT *column_name* FROM *table_name* WHERE *condition*);

COMMENT

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.
Any text between -- and the end of the line will be ignored.

Example:
SELECT * FROM Students
-- WHERE City = ‘Delhi’ ;

STORED PROCEDURE

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

Syntax:
CREATE PROCEDURE procedure-name
AS
sql-statement
GO;
Example:
CREATE PROCEDURE SelectAllStudents
AS
SELECT * FROM Students
GO;

<u>To Execute</u>

Syntax:
EXEC procedure-name;
Example:
EXEC SelectAllStudents;

Stored Procedure With One Parameter

Syntax:
CREATE PROCEDURE procedure-name @Col_name nvarchar(30)
AS
SELECT * FROM Table-name WHERE Col_name = @Col_name
GO;
Example:
CREATE PROCEDURE SelectAllColumns @City nvarchar(30)
AS
SELECT * FROM Students WHERE City = @City
GO;

To Execute
Syntax:
EXEC procedure-name @Col_name = ‘value’;
Example:
EXEC SelectAllColumn @City = ‘Delhi’;

Stored Procedure With Multiple Parameter

Syntax:
CREATE PROCEDURE procedure-name @Col_1 nvarchar(30), @Col_2 navarchar(20)
AS
SELECT * FROM Table-name WHERE Col_1 = @Col_1 AND Col_2 = @Col_2
GO;
Example:
CREATE PROCEDURE SelectAllColumns @City nvarchar(30), @Name nvarchar(20)
AS
SELECT * FROM Students WHERE City = @City AND Name = @Name
GO;

UNION

The UNION operator is used to combine the results of two or more SELECT statements.
The UNION operator selects only distinct values by default.

Syntax :
SELECT *column_name(s)* FROM *table1*
UNION
SELECT *column_name(s)* FROM *table2*;

UNION ALL
To allow duplicate values, use UNION ALL:

Syntax :
SELECT *column_name(s)* FROM *table1*
UNION ALL
SELECT *column_name(s)* FROM *table2*;

UNION WITH WHERE

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

UNION ALL WITH WHERE

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

SELECT INTO

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

Syntax : To copy all columns
SELECT * INTO newtable
FROM oldtable;

Syntax : To copy into a new table in another database.
SELECT * INTO newtable IN ‘DB_name’
FROM oldtable;

Syntax : To copy few columns into new table
SELECT Col_1, Col_2 INTO newtable
FROM oldtable;

Syntax : To copy with Where condition
SELECT * INTO newtable
FROM oldtable
WHERE Col_name = ‘Value’;

INSERT INTO SELECT

The INSERT INTO SELECT statement copies data from one table and inserts it into another existing table.

Syntax : To copy all columns from one table to another
INSERT INTO table2
SELECT * FROM table1;

Syntax : To copy some columns from one table to another
INSERT INTO table2(Col1, Col2, Col3)
SELECT Col1, Col2, Col3
FROM table1;

Syntax : To copy with Where clause
INSERT INTO table2(Col1, Col2, Col3)
SELECT Col1, Col2, Col3
FROM table1
WHERE Col_name='Value'

CASE

The CASE statement goes through conditions and returns a value when the first condition is met (like an IF-THEN-ELSE statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.
If there is no ELSE part and no conditions are true, it returns NULL.

Syntax;
SELECT Col1, Col2
CASE
  WHEN *condition1* THEN *result1*
  WHEN *condition2* THEN *result2*
  WHEN *conditionN* THEN *resultN*
  ELSE *result*
END
FROM table-name;

HOSTING

If we want our web site to be able to store and retrieve data from a database, our web server should have access to a database-system that uses the SQL language.
If our web server is hosted by an Internet Service Provider (ISP), we will have to look for SQL hosting plans.
The most common SQL hosting databases are MS SQL Server, Oracle, MySQL, and MS Access.

MS SQL Server
Microsoft's SQL Server is popular database software for database-driven web sites with high traffic.
SQL Server is a very powerful, robust and full featured SQL database system.

SQL INJECTION

SQL injection is a code injection technique that might destroy your database.
SQL injection is one of the most common web hacking techniques.
SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL in Web Pages

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

Protection From Injection - Use SQL Parameters for Protection

To protect a web site from SQL injection, you can use SQL parameters.
SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

SQL Keywords

Keyword	Description
ADD	Adds a column in an existing table
ADD CONSTRAINT	Adds a constraint after a table is already created
ALTER	Adds, deletes, or modifies columns in a table, or changes the data type of a column in a table
ALTER COLUMN	Changes the data type of a column in a table
ALTER TABLE	Adds, deletes, or modifies columns in a table
ALL	Returns true if all of the subquery values meet the condition
AND	Only includes rows where both conditions is true
ANY	Returns true if any of the subquery values meet the condition
AS	Renames a column or table with an alias
ASC	Sorts the result set in ascending order
BACKUP DATABASE	Creates a back up of an existing database
BETWEEN	Selects values within a given range
CASE	Creates different outputs based on conditions
CHECK	A constraint that limits the value that can be placed in a column
COLUMN	Changes the data type of a column or deletes a column in a table
CONSTRAINT	Adds or deletes a constraint

CREATE	Creates a database, index, view, table, or procedure
CREATE DATABASE	Creates a new SQL database
CREATE INDEX	Creates an index on a table (allows duplicate values)
CREATE OR REPLACE VIEW	Updates a view
CREATE TABLE	Creates a new table in the database
CREATE PROCEDURE	Creates a stored procedure
CREATE UNIQUE INDEX	Creates a unique index on a table (no duplicate values)
CREATE VIEW	Creates a view based on the result set of a SELECT statement
DATABASE	Creates or deletes an SQL database
DEFAULT	A constraint that provides a default value for a column
DELETE	Deletes rows from a table
DESC	Sorts the result set in descending order
DISTINCT	Selects only distinct (different) values
DROP	Deletes a column, constraint, database, index, table, or view
DROP COLUMN	Deletes a column in a table
DROP CONSTRAINT	Deletes a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint
DROP DATABASE	Deletes an existing SQL database
DROP DEFAULT	Deletes a DEFAULT constraint
DROP INDEX	Deletes an index in a table
DROP TABLE	Deletes an existing table in the database
DROP VIEW	Deletes a view
EXEC	Executes a stored procedure
EXISTS	Tests for the existence of any record in a subquery
FOREIGN KEY	A constraint that is a key used to link two tables together
FROM	Specifies which table to select or delete data from
FULL OUTER JOIN	Returns all rows when there is a match in either left table or right table
GROUP BY	Groups the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG
HAVING	Used instead of WHERE with aggregate functions
IN	Allows you to specify multiple values in a WHERE clause

INDEX	Creates or deletes an index in a table
INNER JOIN	Returns rows that have matching values in both tables
INSERT INTO	Inserts new rows in a table
INSERT INTO SELECT	Copies data from one table into another table
IS NULL	Tests for empty values
IS NOT NULL	Tests for non-empty values
JOIN	Joins tables
LEFT JOIN	Returns all rows from the left table, and the matching rows from the right table
LIKE	Searches for a specified pattern in a column
LIMIT	Specifies the number of records to return in the result set
NOT	Only includes rows where a condition is not true
NOT NULL	A constraint that enforces a column to not accept NULL values
OR	Includes rows where either condition is true
ORDER BY	Sorts the result set in ascending or descending order
OUTER JOIN	Returns all rows when there is a match in either left table or right table
PRIMARY KEY	A constraint that uniquely identifies each record in a database table
PROCEDURE	A stored procedure
RIGHT JOIN	Returns all rows from the right table, and the matching rows from the left table
ROWNUM	Specifies the number of records to return in the result set
SELECT	Selects data from a database
SELECT DISTINCT	Selects only distinct (different) values
SELECT INTO	Copies data from one table into a new table
SELECT TOP	Specifies the number of records to return in the result set
SET	Specifies which columns and values that should be updated in a table
TABLE	Creates a table, or adds, deletes, or modifies columns in a table, or deletes a table or data inside a table
TOP	Specifies the number of records to return in the result set
TRUNCATE TABLE	Deletes the data inside a table, but not the table itself
UNION	Combines the result set of two or more SELECT statements (only distinct values)

UNION ALL	Combines the result set of two or more SELECT statements (allows duplicate values)
UNIQUE	A constraint that ensures that all values in a column are unique
UPDATE	Updates existing rows in a table
VALUES	Specifies the values of an INSERT INTO statement
VIEW	Creates, updates, or deletes a view
WHERE	Filters a result set to include only records that fulfill a specified condition

MySQL FUNCTIONS

MySQL has many built-in functions.

MySQL String Functions

Function	Description
ASCII	Returns the ASCII value for the specific character
CHAR_LENGTH	Returns the length of a string (in characters)
CHARACTER_LENGTH	Returns the length of a string (in characters)
CONCAT	Adds two or more expressions together
CONCAT_WS	Adds two or more expressions together with a separator
FIELD	Returns the index position of a value in a list of values
FIND_IN_SET	Returns the position of a string within a list of strings
FORMAT	Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places
INSERT	Inserts a string within a string at the specified position and for a certain number of characters
INSTR	Returns the position of the first occurrence of a string in another string
LCASE	Converts a string to lower-case
LEFT	Extracts a number of characters from a string (starting from left)
LENGTH	Returns the length of a string (in bytes)
LOCATE	Returns the position of the first occurrence of a substring in a string
LOWER	Converts a string to lower-case

LPAD	Left-pads a string with another string, to a certain length
LTRIM	Removes leading spaces from a string
MID	Extracts a substring from a string (starting at any position)
POSITION	Returns the position of the first occurrence of a substring in a string
REPEAT	Repeats a string as many times as specified
REPLACE	Replaces all occurrences of a substring within a string, with a new substring
REVERSE	Reverses a string and returns the result
RIGHT	Extracts a number of characters from a string (starting from right)
RPAD	Right-pads a string with another string, to a certain length
RTRIM	Removes trailing spaces from a string
SPACE	Returns a string of the specified number of space characters
STRCMP	Compares two strings
SUBSTR	Extracts a substring from a string (starting at any position)
SUBSTRING	Extracts a substring from a string (starting at any position)
SUBSTRING_INDEX	Returns a substring of a string before a specified number of delimiter occurs
TRIM	Removes leading and trailing spaces from a string
UCASE	Converts a string to upper-case
UPPER	Converts a string to upper-case

MySQL Numeric Functions

Function	Description
ABS	Returns the absolute value of a number
ACOS	Returns the arc cosine of a number
ASIN	Returns the arc sine of a number
ATAN	Returns the arc tangent of one or two numbers
ATAN2	Returns the arc tangent of two numbers

AVG	Returns the average value of an expression
CEIL	Returns the smallest integer value that is >= to a number
CEILING	Returns the smallest integer value that is >= to a number
COS	Returns the cosine of a number
COT	Returns the cotangent of a number
COUNT	Returns the number of records returned by a select query
DEGREES	Converts a value in radians to degrees
DIV	Used for integer division
EXP	Returns e raised to the power of a specified number
FLOOR	Returns the largest integer value that is <= to a number
GREATEST	Returns the greatest value of the list of arguments
LEAST	Returns the smallest value of the list of arguments
LN	Returns the natural logarithm of a number
LOG	Returns the natural logarithm of a number, or the logarithm of a number to a specified base
LOG10	Returns the natural logarithm of a number to base 10
LOG2	Returns the natural logarithm of a number to base 2
MAX	Returns the maximum value in a set of values
MIN	Returns the minimum value in a set of values
MOD	Returns the remainder of a number divided by another number
PI	Returns the value of PI
POW	Returns the value of a number raised to the power of another number
POWER	Returns the value of a number raised to the power of another number
RADIANS	Converts a degree value into radians
RAND	Returns a random number
ROUND	Rounds a number to a specified number of decimal places
SIGN	Returns the sign of a number
SIN	Returns the sine of a number
SQRT	Returns the square root of a number

SUM	Calculates the sum of a set of values
TAN	Returns the tangent of a number
TRUNCATE	Truncates a number to the specified number of decimal places

MySQL Date Functions

Function	Description
ADDDATE	Adds a time/date interval to a date and then returns the date
ADDTIME	Adds a time interval to a time/datetime and then returns the time/datetime
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
CURRENT_TIME	Returns the current time
CURRENT_TIMESTAMP	Returns the current date and time
CURTIME	Returns the current time
DATE	Extracts the date part from a datetime expression
DATEDIFF	Returns the number of days between two date values
DATE_ADD	Adds a time/date interval to a date and then returns the date
DATE_FORMAT	Formats a date
DATE_SUB	Subtracts a time/date interval from a date and then returns the date
DAY	Returns the day of the month for a given date
DAYNAME	Returns the weekday name for a given date
DAYOFMONTH	Returns the day of the month for a given date
DAYOFWEEK	Returns the weekday index for a given date
DAYOFYEAR	Returns the day of the year for a given date
EXTRACT	Extracts a part from a given date
FROM_DAYS	Returns a date from a numeric datevalue
HOUR	Returns the hour part for a given date

LAST_DAY	Extracts the last day of the month for a given date
LOCALTIME	Returns the current date and time
LOCALTIMESTAMP	Returns the current date and time
MAKEDATE	Creates and returns a date based on a year and a number of days value
MAKETIME	Creates and returns a time based on an hour, minute, and second value
MICROSECOND	Returns the microsecond part of a time/datetime
MINUTE	Returns the minute part of a time/datetime
MONTH	Returns the month part for a given date
MONTHNAME	Returns the name of the month for a given date
NOW	Returns the current date and time
PERIOD_ADD	Adds a specified number of months to a period
PERIOD_DIFF	Returns the difference between two periods
QUARTER	Returns the quarter of the year for a given date value
SECOND	Returns the seconds part of a time/datetime
SEC_TO_TIME	Returns a time value based on the specified seconds
STR_TO_DATE	Returns a date based on a string and a format
SUBDATE	Subtracts a time/date interval from a date and then returns the date
SUBTIME	Subtracts a time interval from a datetime and then returns the time/datetime
SYSDATE	Returns the current date and time
TIME	Extracts the time part from a given time/datetime
TIME_FORMAT	Formats a time by a specified format
TIME_TO_SEC	Converts a time value into seconds
TIMEDIFF	Returns the difference between two time/datetime expressions
TIMESTAMP	Returns a datetime value based on a date or datetime value
TO_DAYS	Returns the number of days between a date and date "0000-00-00"
WEEK	Returns the week number for a given date
WEEKDAY	Returns the weekday number for a given date
WEEKOFYEAR	Returns the week number for a given date

YEAR	Returns the year part for a given date
YEARWEEK	Returns the year and week number for a given date

MySQL Advanced Functions

Function	Description
BIN	Returns a binary representation of a number
BINARY	Converts a value to a binary string
CASE	Goes through conditions and return a value when the first condition is met
CAST	Converts a value (of any type) into a specified datatype
COALESCE	Returns the first non-null value in a list
CONNECTION_ID	Returns the unique connection ID for the current connection
CONV	Converts a number from one numeric base system to another
CONVERT	Converts a value into the specified datatype or character set
CURRENT_USER	Returns the user name and host name for the MySQL account that the server used to authentica client
DATABASE	Returns the name of the current database
IF	Returns a value if a condition is TRUE, or another value if a condition is FALSE
IFNULL	Return a specified value if the expression is NULL, otherwise return the expression
ISNULL	Returns 1 or 0 depending on whether an expression is NULL
LAST_INSERT_ID	Returns the AUTO_INCREMENT id of the last row that has been inserted or updated in a table
NULLIF	Compares two expressions and returns NULL if they are equal. Otherwise, the first expression
SESSION_USER	Returns the current MySQL user name and host name
SYSTEM_USER	Returns the current MySQL user name and host name
USER	Returns the current MySQL user name and host name
VERSION	Returns the current version of the MySQL database

SQL OPERATORS

SQL Arithmetic Operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

SQL Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL Compound Operators

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals

/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

SQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

SQL DATATYPES

The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.

String data types:

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters – can be from 0 to 65535

BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large OBjects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Numeric data types:

Data type	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)
MEDIUMINT(*size*)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which is 255)
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER(*size*)	Equal to INT(size)
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to

	9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255)
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter
DOUBLE PRECISION(*size*, *d*)	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DEC(*size*, *d*)	Equal to DECIMAL(size,d)

Date and Time data types:

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified u DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAM column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000.