SQL NOTES

**What is MYSQL ?**

MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database.

Provides for how to manage database and to manipulate data with the help of various SQL queries. These queries are: insert records, update records, delete records, select records, create tables, drop tables, etc.

Written in [C programming language](https://www.javatpoint.com/c-programming-language-tutorial) and [C++ programming language](https://www.javatpoint.com/cpp-tutorial).

Here are five key points about MySQL:

1. **Open-Source**: MySQL is an open-source RDBMS, meaning it's free to use and has a strong community of developers.
2. **Cross-Platform**: It runs on various operating systems, including Windows, Linux, and macOS.
3. **SQL-Based**: MySQL uses Structured Query Language (SQL) for database operations, making it easy to manage data.
4. **Scalable**: It can handle small applications with a few rows of data to large-scale systems with millions of records.
5. **Secure**: MySQL offers robust security features like user authentication and data encryption to protect sensitive information.

The working of MySQL database with MySQL Server are as follows:

1. MySQL creates a database that allows you to build many tables to store and manipulate data and defining the relationship between each table.
2. Clients make requests through the GUI screen or command prompt by using specific SQL expressions on MySQL.
3. Finally, the server application will respond with the requested expressions and produce the desired result on the client-side.

**What is DATABASE ?**

A database is an application that stores the organized collection of records. It allows us to organize data into tables, rows, columns, and indexes to find the relevant information very quickly.

Today, many databases available like MySQL, Sybase, [Oracle](https://www.javatpoint.com/what-is-oracle), [MongoDB](https://www.javatpoint.com/mongodb-tutorial), [PostgreSQL](https://www.javatpoint.com/postgresql-tutorial), [SQL Server](https://www.javatpoint.com/sql-server-tutorial), etc.

**MYSQL DATATYPES :**

A Data Type specifies a particular type of data, like integer, floating points, Boolean, etc.

It defines the possible values, allowed operations, and how the values are stored for that type.

We can determine the data type in MySQL with the following characteristics:

* The type of values (fixed or variable) it represents.
* The storage space it takes is based on whether the values are a fixed-length or variable length.
* Its values can be indexed or not.
* How MySQL performs a comparison of values of a particular data type.

Numeric Data Type :

 **Integers**:

* Types: TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT.
* Used for whole numbers, with varying ranges depending on the type.

 **Floating-Point**:

* Types: FLOAT, DOUBLE.
* Used for numbers with decimals, where precision can vary.

 **Fixed-Point**:

* Types: DECIMAL, NUMERIC.
* Used for exact decimal values, often in financial calculations where precision is critical.

 **Bit**:

* Stores bit values (0 or 1), useful for binary data or flags.

 **Boolean**:

* Typically a TINYINT(1) where 0 is false and 1 is true, often used for true/false conditions.

Date and Time Data Type:

* **DATE**:

Stores a date value in the format YYYY-MM-DD.

Example: '2024-08-22'.

* **TIME**:

Stores a time value in the format HH:MM:SS.

Example: '14:30:00'.

* **DATETIME**:

Combines date and time in the format YYYY-MM-DD HH:MM:SS.

Example: '2024-08-22 14:30:00'.

Useful for tracking events with both date and time.

* **TIMESTAMP**:

Similar to DATETIME but also tracks timezone and can automatically update to the current timestamp on record updates.

Example: '2024-08-22 14:30:00'.

Commonly used for logging and audit trails.

* **YEAR**:

Stores a year value in either 2-digit or 4-digit format (YY or YYYY).

Example: '2024' or '24'.

Useful for storing years alone, such as birth years or production year

String Data Types:

**CHAR**:

* Fixed-length string with a defined length (1-255 characters).
* Example: CHAR(10) always stores 10 characters, padding with spaces if needed.
* Ideal for data like codes or identifiers.

**VARCHAR**:

* Variable-length string with a defined maximum length (1-65,535 characters).
* Example: VARCHAR(100) stores up to 100 characters.
* Efficient for storing text where the length varies.

**TEXT**:

* Used for large text data, can store up to 65,535 characters.
* Variants include TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT for varying sizes.
* Commonly used for storing articles, descriptions, or comments.

**BLOB**:

* Binary Large Object for storing binary data (images, files, etc.).
* Variants include TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB for different sizes.
* Used for raw data storage like multimedia files.

**ENUM**:

* String object that can have one value chosen from a predefined list.
* Example: ENUM('small', 'medium', 'large') stores one of these three values.
* Useful for fields with a limited set of possible values.

**SET**:

* Similar to ENUM, but allows storing multiple values from a predefined list.
* Example: SET('A', 'B', 'C') can store any combination of these values.
* Useful for fields where multiple selections are allowed, like tags or categories.

Binary Large Object Data Types (BLOB):

BLOB types are used to store raw binary data, such as images, videos, audio files, or any other type of file content, without any character set conversion. They are particularly useful when dealing with large and unstructured data that needs to be stored in the database.

**TINYBLOB**:

* Stores small binary data up to 255 bytes.
* Suitable for tiny images, small files, or short binary data.

**BLOB**:

* Stores binary data up to 65,535 bytes (64 KB).
* Commonly used for images, documents, and other medium-sized files.

**MEDIUMBLOB**:

* Stores binary data up to 16,777,215 bytes (16 MB).
* Useful for larger multimedia files like audio or video clips.

**LONGBLOB**:

* Stores large binary data up to 4,294,967,295 bytes (4 GB).
* Ideal for very large files, such as high-resolution images, videos, or large datasets

Spatial Data Types :

Spatial data types are used to store geometric and geographic data, such as maps, location coordinates, and complex shapes. They support spatial indexing and querying, allowing for efficient storage and manipulation of spatial information.

**GEOMETRY**:

* A general type for storing spatial data, including points, lines, and polygons.
* Example: Used for various geometric shapes and spatial data.

**POINT**:

* Stores a single location in a two-dimensional plane using X and Y coordinates.
* Example: POINT(10, 20) represents a point at coordinates (10, 20).

**LINESTRING**:

* Represents a series of connected points (a line).
* Example: LINESTRING(10 20, 30 40, 50 60) creates a line connecting these points.

**POLYGON**:

* Represents a closed shape with multiple sides (a polygon).
* Example: POLYGON((10 20, 30 40, 50 60, 10 20)) defines a polygon with four vertices.

**MULTIPOINT**:

* Stores multiple points as a single spatial value.
* Example: MULTIPOINT((10 20), (30 40), (50 60)).

**MULTILINESTRING**:

* Stores multiple line strings as a single spatial value.
* Example: MULTILINESTRING((10 20, 30 40), (50 60, 70 80)).

**MULTIPOLYGON**:

* Stores multiple polygons as a single spatial value.
* Example: MULTIPOLYGON(((10 20, 30 40, 50 60, 10 20)), ((60 70, 80 90, 100 110, 60 70))).

**GEOMETRYCOLLECTION**:

* Stores a collection of different geometric types in a single value.
* Example: GEOMETRYCOLLECTION(POINT(10 20), LINESTRING(30 40, 50 60)).

**JSON Data Type :**

Ideal for handling JSON documents with validation and optimized storage, enhancing data integrity and access speed.

**Advantages**:

1. **Automatic Validation**: Ensures JSON documents are valid; invalid documents trigger an error.
2. **Optimized Storage**: Stores JSON data efficiently, improving performance over plain text storage.

**Expressions in SQL:**

Expressions in SQL are combinations of symbols and operators that evaluate to a single value.

Types of expressions include:

**Arithmetic Expressions:** Perform mathematical operations (+, -, \*, /).

**String Expressions:** Concatenate strings using concatenation operator (||).

**Comparison Expressions:** Compare values using comparison operators (=, <>, <, >, <=, >=).

**Logical Expressions:** Combine conditions using logical operators (AND, OR, NOT).

Expressions can be used in SELECT statements, WHERE clauses, and other SQL commands to manipulate and filter data.

**Operators in SQL:**

SQL operators are symbols used to perform operations on data.

Types of operators include:

**Arithmetic Operators:** Perform mathematical calculations (+, -, \*, /).

**Comparison Operators:** Compare values (=, <>, <, >, <=, >=).

**Logical Operators:** Combine conditions (AND, OR, NOT).

**Concatenation Operator:** Combine strings (||).

**Assignment Operator:** Assign values (=).

**Bitwise Operators:** Perform operations on bits (&, |, ^).

Operators facilitate data manipulation, filtering, and computation in SQL queries.

**MySQL Variables :**

Variables are used for storing data or information during the execution of a program.

The main purpose of the variable is to store data in memory and can be used throughout the program.

**User-Defined Variable :**

* Description: Custom variables created by users for temporary storage and manipulation within a session.
* Example:
* SET @myVar = 10;

SELECT @myVar + 5; (Stores 10 in @myVar and adds 5 to it).

**Local Variable :**

* **Description**: Variables defined within stored procedures or functions, only accessible within their scope.
* Example :

CREATE PROCEDURE exampleProc()

BEGIN

DECLARE **localVar** INT DEFAULT 100;

SELECT localVar;

END;

**System Variable :**

* **Description**: Predefined variables that control MySQL server behavior and settings.
* **Example**:

SELECT @@global.max\_connections; (Shows the maximum number of connections allowed).

**MYSQL CREATE DATABASE :**

* **CREATE** **DATABASE** [IF NOT EXISTS] database\_name;
* **CREATE** **DATABASE** [IF NOT EXISTS] employeesdb;

 The **IF NOT EXIST** clause avoids an error when we create a database that already exists.

* Below query that returns the database name, character set, and collation of the database:
* **SHOW DATABASES;**
* to access the database that enables us to create a table and other database objects.
* **USE**employeesdb**;**

#### **All the database names, table names, and table field names are case sensitive.**

**MYSQL SELECT DATABASE :**

You can use SQL command USE to select a particular database.

* **USE**employeesdb**;**

**MYSQL DROP DATABASE :**

* We can drop an existing database in MySQL by using the DROP DATABASE statement with the below syntax:
* **DROP** **DATABASE** [IF EXISTS] database\_name;

we can also use the below syntax for deleting the database. It is because the **schema** is the synonym for the database, so we can use them interchangeably.

* **DROP** **SCHEMA** [IF EXISTS] database\_name;

***there is no difference between DROP SCHEMA and DROP DATABASE—both commands are synonymous and perform the same action of removing a database.***

**MYSQL COPY DATABASE:**

MySQL copy or clone database is a feature that allows us to create a **duplicate copy of an existing database**, including the table structure, indexes, constraints, default values, etc.

**TABLE & VIEWS :**

**A table creation command requires three things:**

* Name of the table
* Names of fields
* Definitions for each field

**CREATE** **TABLE** :

**CREATE** **TABLE** [IF NOT EXISTS] table\_name(

    column\_definition1,

    column\_definition2,

    ........,

    table\_constraints

);

|  |  |
| --- | --- |
| column\_definition | It specifies the name of the column along with data types for each column.  The columns in table definition are separated by the comma operator. |
| table\_constraints | It specifies the table constraints such as PRIMARY KEY, UNIQUE KEY, FOREIGN KEY,  CHECK, etc. |

Create a table name **"employee\_table"** in the database **"employeedb"**

|  |
| --- |
| **CREATE TABLE employee\_table(**  **id int NOT NULL AUTO\_INCREMENT,**  **name varchar(45) NOT NULL,**  **occupation varchar(35) NOT NULL,**  **age int NOT NULL,**  **PRIMARY KEY (id)**  **);** |

|  |
| --- |
| **NOTE:** 1. Here, NOT NULL is a field attribute, and it is used because we don't want this field to be NULL. If we try to create a record with a NULL value, then MySQL will raise an error. 2. The field attribute AUTO\_INCREMENT specifies MySQL to go ahead and add the next available number to the id field. PRIMARY KEY is used to define a column's uniqueness. We can use multiple columns separated by a comma to define a primary key. |

Following command to see the newly created table:

* **SHOW TABLES;**

 Following command to see the information or structure of the newly created table:

* **DESCRIBE** employee\_table;
* **DESC** employee\_table;

**ALTER TABLE :**

* MySQL ALTER statement is used when you want to change the name of your table or any table field. It is also used to add or delete an existing column in a table.
* The ALTER statement is always used with "ADD", "DROP" and "MODIFY" commands according to the situation.

## **ADD A COLUMN IN THE TABLE :**

|  |
| --- |
| **ALTER** **TABLE** table\_name  **ADD** new\_column\_name column\_definition  [ **FIRST** | **AFTER** column\_name ]; |