RDBMS Using MYSQL

UNIT - I

What is MySQL?

MySQL is currently the most popular database management system software used for managing the relational database. It is open-source database software, which is supported by Oracle Company. It is fast, scalable, and easy to use database management system in comparison with Microsoft SQL Server and Oracle Database. It is commonly used in conjunction with PHP scripts for creating powerful and dynamic server-side or web-based enterprise applications.

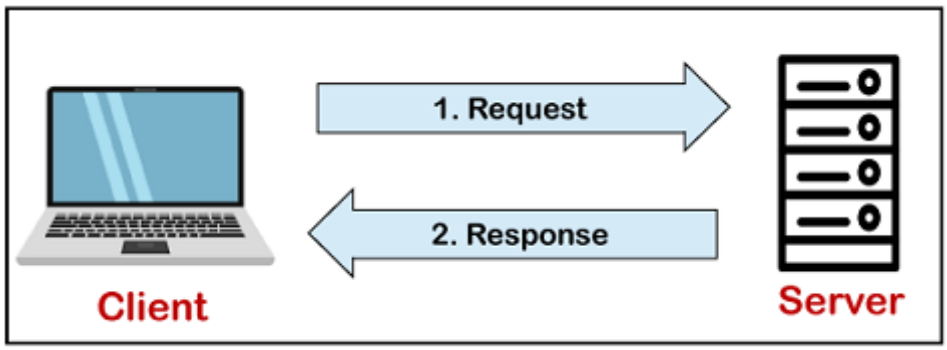
It is developed, marketed, and supported by **MySQL AB, a Swedish company**, and written in C programming language and C++ programming language. The official pronunciation of MySQL is not the My Sequel; it is ***My Ess Que Ell****. However, you can pronounce it in your way.* Many small and big companies use MySQL. MySQL supports many Operating Systems like Windows, Linux, MacOS, etc. with C, C++, and Java languages.

MySQL is a Relational Database Management System (RDBMS) software that provides many things, which are as follows:

- It allows us to implement database operations on tables, rows, columns, and indexes.
- It defines the database relationship in the form of tables (collection of rows and columns), also known as relations.
- It provides the Referential Integrity between rows or columns of various tables.
- It allows us to updates the table indexes automatically.
- It uses many SQL queries and combines useful information from multiple tables for the end-users.

How MySQL Works?

MySQL follows the working of Client-Server Architecture. This model is designed for the end-users called clients to access the resources from a central computer known as a server using network services. Here, the clients make requests through a graphical user interface (GUI), and the server will give the desired output as soon as the instructions are matched. The process of MySQL environment is the same as the client-server model.



The core of the MySQL database is the MySQL Server. This server is available as a separate program and responsible for handling all the database instructions, statements, or commands. 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.

A client can use any MySQL GUI. But, it is making sure that your GUI should be lighter and user-friendly to make your data management activities faster and easier. Some of the most widely used MySQL GUIs are MySQL Workbench, SequelPro, DBVisualizer, and the Navicat DB Admin Tool. Some GUIs are commercial, while some are free with limited functionality, and some are only compatible with MacOS. Thus, you can choose the GUI according to your needs.

Reasons for popularity

MySQL is becoming so popular because of these following reasons:

- MySQL is an open-source database, so you don't have to pay a single penny to use it.
- MySQL is a very powerful program that can handle a large set of functionality of the most expensive and powerful database packages.
- MySQL is customizable because it is an open-source database, and the open-source GPL license facilitates programmers to modify the SQL software according to their own specific environment.
- MySQL is quicker than other databases, so it can work well even with the large data set.
- MySQL supports many operating systems with many languages like PHP, PERL, C, C++, JAVA, etc.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL is very friendly with PHP, the most popular language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).

History of MySQL

The project of MySQL was started in 1979 when MySQL's inventor **Michael Widenius** developed an in-house database tool called **UNIREG** for managing databases. After that, UNIREG has been rewritten in several different languages and extended to handle big databases. After some time, Michael Widenius contacted **David Hughes**, the author of mSQL, to see if Hughes would be interested in connecting mSQL to UNIREG's B+ ISAM handler to provide indexing to mSQL. That's the way MySQL came into existence.

“MySQL is named after the daughter of co-founder Michael Widenius whose name is "My".”

MySQL Features

MySQL is a relational database management system (RDBMS) based on the SQL (Structured Query Language) queries. It is one of the most popular languages for accessing and managing the records in the table. MySQL is open-source and free software under the GNU license. Oracle Company supports it.

The following are the most important features of MySQL:

Relational Database Management System (RDBMS)

MySQL is a relational database management system. This database language is based on the SQL queries to access and manage the records of the table.

Easy to use

MySQL is easy to use. We have to get only the basic knowledge of SQL. We can build and interact with MySQL by using only a few simple SQL statements.

It is secure

MySQL consists of a solid data security layer that protects sensitive data from intruders. Also, passwords are encrypted in MySQL.

Client/ Server Architecture

MySQL follows the working of a client/server architecture. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they can query data, save changes, etc.

Free to download

MySQL is free to use so that we can download it from MySQL official website without any cost.

It is scalable

MySQL supports multi-threading that makes it easily scalable. It can handle almost any amount of data, up to as much as 50 million rows or more. The default file size limit is about 4 GB. However, we can increase this number to a theoretical limit of 8 TB of data.

Speed

MySQL is considered one of the very fast database languages, backed by a large number of the benchmark test.

High Flexibility

MySQL supports a large number of embedded applications, which makes MySQL very flexible.

Compatible on many operating systems

MySQL is compatible to run on many operating systems, like Novell NetWare, Windows* Linux*, many varieties of UNIX* (such as Sun* Solaris*, AIX, and DEC* UNIX), OS/2, FreeBSD*, and others. MySQL also provides a facility that the clients can run on the same computer as the server or on another computer (communication via a local network or the Internet).

Allows roll-back

MySQL allows transactions to be rolled back, commit, and crash recovery.

Memory efficiency

Its efficiency is high because it has a very low memory leakage problem.

High Performance

MySQL is faster, more reliable, and cheaper because of its unique storage engine architecture. It provides very high-performance results in comparison to other databases without losing an essential functionality of the software. It has fast loading utilities because of the different cache memory.

High Productivity

MySQL uses Triggers, Stored procedures, and views that allow the developer to give higher productivity.

Platform Independent

It can download, install, and execute on most of the available operating systems.

Partitioning

This feature improves the performance and provides fast management of the large database.

GUI Support

MySQL provides a unified visual database graphical user interface tool named "**MySQL Workbench**" to work with database architects, developers, and Database Administrators. MySQL Workbench provides SQL development, data modeling, data migration, and comprehensive administration tools for server configuration, user administration, backup, and many more. MySQL has a fully GUI supports from MySQL Server version 5.6 and higher.

Dual Password Support

MySQL version 8.0 provides support for dual passwords: one is the current password, and another is a secondary password, which allows us to transition to the new password.

Disadvantages/Drawback of MySQL

Following are the few disadvantages of MySQL:

- MySQL version less than 5.0 doesn't support ROLE, COMMIT, and stored procedure.
- MySQL does not support a very large database size as efficiently.
- MySQL doesn't handle transactions very efficiently, and it is prone to data corruption.
- MySQL is accused that it doesn't have a good developing and debugging tool compared to paid databases.
- MySQL doesn't support SQL check constraints.

Four Different Methods to start and stop mysql in linux

Start MySQL

1. **Using the systemctl Command:**

- If your system uses systemd, you can start MySQL with:

```
sudo systemctl start mysql
```

2. **Using the service Command:**

- On older systems or those using init.d, you can start MySQL with:

```
sudo service mysql start
```

3. **Using the mysql.server Script:**

   - If MySQL was installed from source or through certain packages, you can use the mysql.server script:

```
sudo mysql.server start
```

4. **Directly Executing the mysqld Daemon:**

   - You can also start MySQL directly by running the mysqld daemon, though this is less common and usually done for troubleshooting:

```
sudo /usr/sbin/mysqld
```

Stop MySQL

1. **Using the systemctl Command:**

   - For systems using systemd:

```
sudo systemctl stop mysql
```

2. **Using the service Command:**

   - On systems using init.d:

```
sudo service mysql stop
```

3. **Using the mysqladmin Command:**

   - If you want to stop MySQL from within MySQL itself, you can use mysqladmin:

```
sudo mysqladmin -u root -p shutdown
```

   - You will need to enter the root password when prompted.

4. **Directly Executing the mysqld Daemon Stop Command:**

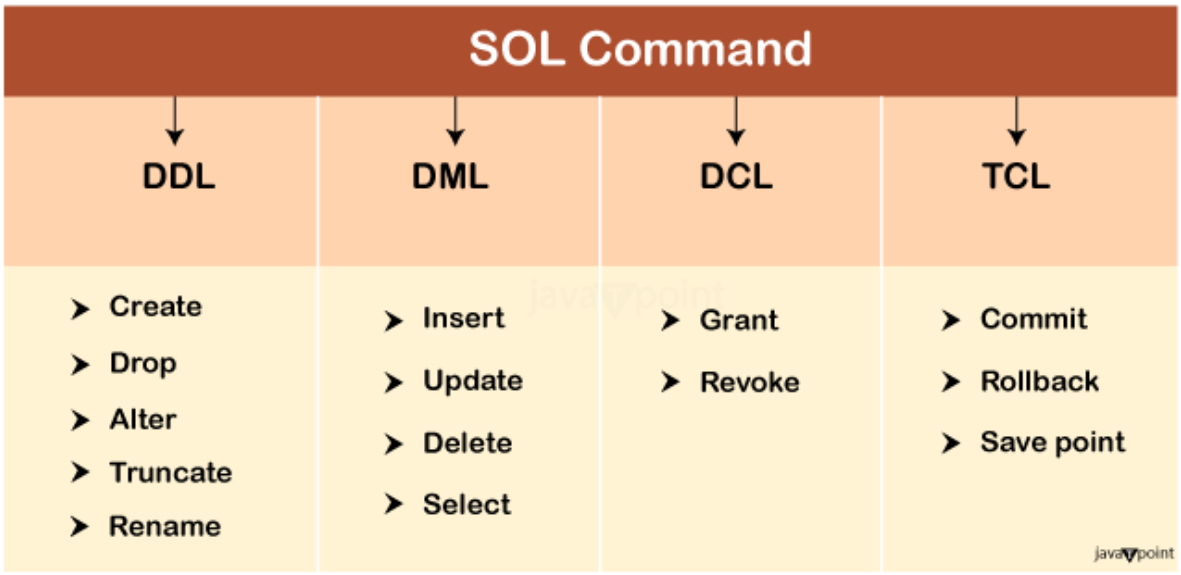
   - This method is less common but can be used to stop the mysqld daemon directly:

```
sudo kill `cat /var/run/mysqld/mysqld.pid`
```

   - Be cautious with this method as it involves manually killing the process.

Types of MySQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



DML Queries

DML (Data Manipulation Language) queries are used to manage and manipulate data within a database. In MySQL, the primary DML operations are `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. Here's a detailed explanation of each:

1. **SELECT**

The `SELECT` statement is used to retrieve data from one or more tables.

Syntax:

```
SELECT column1, column2, ...
FROM table_name;
```

- column1, column2, ... : The columns you want to retrieve.

- table_name : The table from which you want to retrieve data.

Examples:

- Retrieve all columns from a table:

```
SELECT * FROM employees;
```

- Retrieve specific columns:

```
SELECT name, salary FROM employees;
```

2. INSERT

The `INSERT` statement is used to add new rows to a table.

Syntax :

```
INSERT INTO table_name (column1, column2, ...)
```

```
VALUES (value1, value2, ...);
```

- table_name : The table where you want to insert data.

- column1, column2, ... : The columns you want to insert values into.

- value1, value2, ... : The values to be inserted.

Examples:

- Insert a single row:

```
INSERT INTO employees (name, age, salary)
VALUES ('Jane Doe', 29, 55000);
```

- Insert multiple rows:

```
INSERT INTO employees (name, age, salary)
VALUES ('John Doe', 35, 60000),
       ('Alice Smith', 30, 65000);
```

3. UPDATE

The `UPDATE` statement is used to modify existing rows in a table.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

- table_name: The table where you want to update data.

- column1 = value1, column2 = value2, ... : The columns and their new values.

- condition : The criteria to select the rows to be updated.

Examples:

- Update a single column for specific rows:

```
UPDATE employees
SET salary = 70000
WHERE name = 'Jane Doe';
```

- Update multiple columns for specific rows:

```
UPDATE employees
SET salary = 72000, age = 31
WHERE name = 'Alice Smith';
```

4. DELETE

The `DELETE` statement is used to remove rows from a table.

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

- table_name : The table from which you want to delete data.

- condition : The criteria to select the rows to be deleted.

Examples:

- Delete specific rows:

```
DELETE FROM employees
WHERE name = 'John Doe';
```

- Delete all rows from a table (without dropping the table):

```
DELETE FROM employees;
```

Important Note : Be careful with `DELETE` and `UPDATE` statements, as they can modify or remove data permanently. Always ensure that the `WHERE` clause is correct to avoid unintended changes.

These DML operations allow you to efficiently manage and manipulate data in your MySQL database.

DDL Queries

DDL (Data Definition Language) queries are used to define and manage database structures such as tables, indexes, and schemas. They include operations like creating, altering, and deleting database objects. Here's a detailed explanation of each DDL query:

1. CREATE

The `CREATE` statement is used to create new database objects such as tables, indexes, and views.

- Create a Table :

```
CREATE TABLE table_name (
    column1 datatype [constraints],
    column2 datatype [constraints],
    ...
);
```

Example:

```
CREATE TABLE employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    name VARCHAR(100) NOT NULL,
    age INT,
    salary DECIMAL(10, 2)
);
```

This creates a new table named `employees` with columns for `id`, `name`, `age`, and `salary`.

- Create an Index :

```
CREATE INDEX index_name ON table_name (column_name);
```

Example:

```
CREATE INDEX idx_salary ON employees (salary);
```

This creates an index on the `salary` column of the `employees` table to speed up queries involving this column.

- Create a View :

```
CREATE VIEW view_name AS
SELECT column1, column2
FROM table_name
WHERE condition;
```

Example:

```
CREATE VIEW
high_salary_employees AS
SELECT name, salary
FROM employees
WHERE salary > 50000;
```

This creates a view named `high_salary_employees` that shows employees with a salary greater than 50,000.

2. ALTER

The `ALTER` statement is used to modify existing database objects. This includes adding or dropping columns, changing data types, and adding constraints.

- Add a Column:

```
ALTER TABLE table_name ADD column_name datatype [constraints];
```

Example:

```
ALTER TABLE employees ADD department VARCHAR(50);
```

This adds a new column `department` to the `employees` table.

- Drop a Column :

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Example:

```
ALTER TABLE employees DROP COLUMN department;
```

This removes the `department` column from the `employees` table.

- Modify a Column:

```
ALTER TABLE table_name MODIFY column_name new_datatype [constraints];
```

Example:

```
ALTER TABLE employees MODIFY salary DECIMAL(12, 2);
```

This changes the data type of the `salary` column to allow for more precision.

- Rename a Column :

```
ALTER TABLE table_name CHANGE old_column_name new_column_name datatype;
```

Example:

```
ALTER TABLE employees CHANGE name full_name VARCHAR(100);
```

This renames the `name` column to `full_name` and changes its datatype to `VARCHAR(100)`.

3. DROP

The `DROP` statement is used to remove existing database objects. This operation is irreversible and deletes both the structure and the data.

- Drop a Table :

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE employees;
```

This removes the `employees` table and all its data.

- Drop an Index :

```
DROP INDEX index_name ON table_name;
```

Example:

```
DROP INDEX idx_salary ON employees;
```

This deletes the `idx_salary` index from the `employees` table.

- Drop a View :

```
DROP VIEW view_name;
```

Example:

```
DROP VIEW high_salary_employees;
```

This removes the `high_salary_employees` view.

4. TRUNCATE

The `TRUNCATE` statement is used to remove all rows from a table but does not delete the table itself. It is faster than `DELETE` because it does not log individual row deletions.

- Truncate a Table :

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE employees;
```

This removes all rows from the `employees` table but keeps the table structure intact.

5. RENAME

The `RENAME` statement is used to rename database objects.

- Rename a Table :

```
RENAME TABLE old_table_name TO new_table_name;
```

Example:

```
RENAME TABLE employees TO staff;
```

This renames the `employees` table to `staff`.

TCL Queries

TCL (Transaction Control Language) queries are used to manage transactions in a database. Transactions are sequences of one or more SQL operations executed as a single unit, ensuring data integrity and consistency. TCL commands include `COMMIT`, `ROLLBACK` and `SAVEPOINT`. Here's a detailed explanation of each:

1. COMMIT

The `COMMIT` statement is used to save all changes made during the current transaction. Once a transaction is committed, all changes become permanent and visible to other transactions.

- Basic Usage:

```
COMMIT;
```

- **Example:**

```
BEGIN;
INSERT INTO employees (name, age, salary) VALUES ('John Doe', 30, 50000);
UPDATE employees SET salary = salary + 5000 WHERE name = 'John Doe';
COMMIT;
```

In this example, the transaction starts with `BEGIN`, makes some changes to the `employees` table, and then commits those changes. If `COMMIT` is executed, both the `INSERT` and `UPDATE` operations are saved.

2. ROLLBACK

The `ROLLBACK` statement is used to undo all changes made during the current transaction. It restores the database to its state before the transaction began.

- Basic Usage :

```
ROLLBACK;
```

- **Example:**

```
BEGIN;
INSERT INTO employees (name, age, salary) VALUES ('Jane Smith', 28, 55000);
DELETE FROM employees WHERE name = 'John Doe';
ROLLBACK;
```

In this example, the transaction starts, performs an `INSERT` and `DELETE`, and then rolls back. The `ROLLBACK` command undoes both operations, so no changes are saved.

3. SAVEPOINT

The `SAVEPOINT` statement is used to set a point within a transaction that you can roll back to, allowing partial rollback of a transaction.

- Basic Usage :

```
SAVEPOINT savepoint_name;
```

- **Example:**

```
BEGIN;
INSERT INTO employees (name, age, salary) VALUES ('Alice Brown', 25, 60000);
SAVEPOINT before_update;
UPDATE employees SET salary = salary + 10000 WHERE name = 'Alice Brown';
ROLLBACK TO SAVEPOINT before_update;
COMMIT;
```

In this example, a `SAVEPOINT` named `before_update` is set. If something goes wrong with the `UPDATE` operation, you can roll back to `before_update`, undoing the `UPDATE` but keeping the `INSERT`.

Types of Joins

Joins in MySQL allow you to combine rows from two or more tables based on a related column between them. Here are the main types of joins and what they do:

1. INNER JOIN

- Description : Returns records that have matching values in both tables.

- Usage : Use when you need to select records that exist in both tables.

- Syntax :

```
SELECT columns
  FROM table1
  INNER JOIN table2
  ON table1.common_column = table2.common_column;
```

- Example :

Suppose you have two tables: `employees` and `departments`.

```
SELECT employees.name, departments.department_name
  FROM employees
  INNER JOIN departments
  ON employees.department_id = departments.id;
```

2. LEFT JOIN (or LEFT OUTER JOIN)

- Description : Returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side if there is no match.

- Usage : Use when you want to include all records from the left table and only the matched records from the right table.

- Syntax :

```
SELECT columns
  FROM table1
  LEFT JOIN table2
  ON table1.common_column = table2.common_column;
```

- Example :

```
SELECT employees.name, departments.department_name
  FROM employees
  LEFT JOIN departments
  ON employees.department_id = departments.id;
```

3. RIGHT JOIN (or RIGHT OUTER JOIN)

- Description : Returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side when there is no match.

- Usage : Use when you want to include all records from the right table and only the matched records from the left table.

- Syntax :

```
SELECT columns
  FROM table1
  RIGHT JOIN table2
  ON table1.common_column = table2.common_column;
```

- Example :

```
SELECT employees.name, departments.department_name
  FROM employees
  RIGHT JOIN departments
  ON employees.department_id = departments.id;
```

4. FULL JOIN (or FULL OUTER JOIN)

- Description : Returns all records when there is a match in either left or right table. Note that MySQL does not support FULL JOIN directly, but you can simulate it using a combination of LEFT JOIN and RIGHT JOIN.

- Usage : Use when you want to include all records from both tables, regardless of whether there is a match.

- Syntax :

```
SELECT columns
  FROM table1
  LEFT JOIN table2
  ON table1.common_column = table2.common_column
  UNION
  SELECT columns
  FROM table1
  RIGHT JOIN table2
  ON table1.common_column = table2.common_column;
```

- **Example**:

```
SELECT employees.name, departments.department_name
  FROM employees
  LEFT JOIN departments
  ON employees.department_id = departments.id
  UNION
  SELECT employees.name,
departments.department_name
  FROM employees
```

```
RIGHT JOIN departments
ON employees.department_id = departments.id;
```

5. CROSS JOIN

- Description : Returns the Cartesian product of the two tables. This means it will return every combination of rows from both tables.

- Usage : Use when you need all possible combinations of rows from both tables.

- Syntax:

```
SELECT columns
FROM table1
CROSS JOIN table2;
```

- Example :

```
SELECT employees.name, departments.department_name
FROM employees
CROSS JOIN departments;
```

6. SELF JOIN

- Description : A self join is a regular join but the table is joined with itself.

- Usage : Use when you need to compare rows within the same table.

- Syntax :

```
SELECT A.columns, B.columns
FROM table A, table B
WHERE A.common_column = B.common_column;
```

- Example:

Suppose you have an `employees` table where each employee has a `manager_id` that refers to another employee:

```
SELECT A.name AS Employee, B.name AS Manager
FROM employees A
LEFT JOIN employees B
ON A.manager_id = B.id;
```

These are the primary types of joins you'll use in MySQL. Each join type has its use cases and helps to retrieve specific sets of data depending on your needs.

Union

Unions are used to combine the results of two or more `SELECT` queries into a single result set. The queries must have the same number of columns, and the corresponding columns must have compatible data types.

1. UNION

- Description : Combines the results of two or more SELECT queries, removing duplicate rows by default.

- Syntax :

```
SELECT column1, column2 FROM table1
UNION
SELECT column1, column2 FROM table2;
```

- Use: When you want to merge results from multiple queries, ensuring no duplicates in the final output.

2. UNION ALL

- Description : Similar to `UNION`, but includes all rows, including duplicates.

- Syntax :

```
SELECT column1, column2 FROM table1
UNION ALL
SELECT column1, column2 FROM table2;
```

- Use : When you want to merge results from multiple queries and keep all rows, including duplicates.

Key Points:

- The number of columns and their data types must match across all queries in the UNION.

- The column names in the final result set are taken from the first SELECT statement.

- The `UNION` operation can combine results from different tables or even the same table.

Example:

Suppose you have two tables, `employees` and `contractors`, and you want to list all people (both employees and contractors) without duplicates:

```
SELECT name FROM employees
UNION
SELECT name FROM contractors;
```

SELECT name FROM employees

UNION

SELECT name FROM contractors;

```
```

If you want to include duplicates:

```
SELECT name FROM employees
UNION ALL
SELECT name FROM contractors;
```

This brief overview explains how unions are used to merge results from multiple queries, either with or without removing duplicates.

Various logs in MySQL

MySQL maintains several types of logs that are essential for monitoring, troubleshooting, and optimizing database performance.

1. Error Log

- **Description**: Records problems encountered by the MySQL server, such as startup and shutdown issues, and critical errors during operation.
- **Use**: Essential for diagnosing server crashes, startup problems, and other critical issues.
- **Location**: Configurable in the MySQL configuration file (my.cnf), typically under the log_error variable.

2. General Query Log

- **Description**: Logs all SQL queries received by the server, including administrative commands like CREATE, ALTER, and SET.
- **Use**: Useful for tracking client connections and understanding what queries are being executed, helpful in debugging and auditing.
- **Location**: Configurable in my.cnf using the general_log and general_log_file variables.

3. Slow Query Log

- **Description**: Logs queries that take longer than a specified time to execute, based on the long_query_time setting.
- **Use**: Crucial for identifying and optimizing slow-running queries, improving overall database performance.
- **Location**: Configurable in my.cnf using the slow_query_log and slow_query_log_file variables.

4. Binary Log

- **Description**: Records all changes to the database, such as INSERT, UPDATE, DELETE statements, and other data-modifying operations. It does not log SELECT statements.
- **Use**: Used for replication (synchronizing databases across servers) and for data recovery (point-in-time recovery).
- **Location**: Configurable in my.cnf using the log_bin variable.

5. Relay Log

- **Description**: Used in replication, it stores events received from the master server before applying them to the slave database.
- **Use**: Part of the replication process, helping the slave server to keep an exact copy of the master.
- **Location**: Managed automatically by MySQL in the replication process.