# Assignment A1

Date:

| | |
|---|---|
| TITLE | Study of open source relational database: MySQL |
| PROBLEM STATEMENT / DEFINITION | To study open source relational MySQL |
| LEARNING OBJECTIVE | To learn and understand the basic database architecture and various components of it |
| LEARNING OUTCOME | The students will be able to<br>• Understand the basic database architecture and the various components of it. |
| S/W PACKAGES & HARDWARE APPARATUS USED | • MySQL<br>• 64-bit Linux based open source OS<br>• 8 GB RAM |

## CONCEPT RELATED THEORY:

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed, and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons:

MySQL is released under an open-source license. So, you have nothing to pay to use it. MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages. MySQL uses a standard form of the well-known SQL data language. MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc. MySQL works very quickly and works well even with large data sets. MySQL is very friendly to PHP, the most appreciated 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). MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

1. Application Layer and Interfaces

MySQL application layer is where the clients and users interact with the MySQL RDBMS. There are three components in this layer as can be seen in the layered. These components illustrate the different kinds of users that can interact with the MySQL RDBMS, which are the administrators, clients and query users. The administrators use the administrative interface and utilities. In MySQL, some of these utilities are MySQL admin which performs tasks like shutting down the server and creating or dropping databases. Clients communicate to the MySQL RDBMS through the interface or utilities. The client interface uses MySQL APIs for various different programming languages such as the C API, DBI API for Perl, PHP API, Java API, Python API, MySQL C++ API and Tcl. Query users interact with the MySQL RDBMS through a query interface that is MySQL. MySQL is a monitor (interactive program) that allows the query users to issue SQL statements to the server and view the results.

2. Logical Layer

It was found that MySQL does indeed have a logical architecture. The MySQL documentation gave an indication as to precisely how these modules could be further broken down into subsystems arranged in a layered hierarchy corresponding to the layered architecture in Garlan and Shaw. The following section details these subsystems and the interactions within them.

### 2.1 Query Processor

The vast majority of interactions in the system occur when a user wishes to view or manipulate the underlying data in storage. These queries, which are specified using a data-manipulation language (ie SQL), are parsed and optimized by a query processor. This processor, depicted in Figure 3 above, can be represented as pipeline and filter architecture in the sense of Garlan and Shaw where the result of the previous component becomes an input or requirement to the next component. The component architecture of the query processor will be explained below.

#### 2.1.1 Embedded DML Precompiler

When a request is received from a client in the application layer, it is the responsibility of the embedded DML (Data Manipulation Language) precompiler to extract the relevant SQL statements embedded in the client API commands, or to translate the client commands into the corresponding SQL statements. This is the first step in the actual processing of a client application written in a programming language such as C++ or Perl, before compiling the SQL query. The client request could come from commands executed from an application interface (API), or an application program. This is prevalent in all general RDBMS's. MySQL has this component in order to process the MySQL client application request into the format that MySQL understands.

#### 2.1.2 DDL Compiler

Requests to access the MySQL databases received from an administrator are processed by the DDL (Data Definition Language) compiler. The DDL compiler compiles the commands (which are SQL statements) to interact directly with the database. The administrator and administrative utilities do not expose an interface, and hence execute directly to the MySQL server. Therefore, the embedded DML precompiler does not process it, and this explains the need for a DDL compiler.

#### 2.1.3 Query Parser

After the relevant SQL query statements are obtained from deciphering the client request or the administrative request, the next step involves parsing the MySQL query. In this stage, the objective of the query parser is to create a parse tree structure based on the query so that it can be easily understood by the other components later in the pipeline.

#### 2.1.4 Query Preprocessor

The query parse tree, as obtained from the query parser, is then used by the query preprocessor to check the SQL syntax and check the semantics of the MySQL query to determine if the query is valid. If it is a valid query, then the query progresses down the pipeline. If not, then the query does not proceed, and the client is notified of the query processing error.

#### 2.1.5 Security/Integration Manager

Once the MySQL query is deemed to be valid, the MySQL server needs to check the access control list for the client. This is the role of the security integration manager which checks to see if the client has access to connecting to that particular MySQL database and whether he/she has table and

record privileges. In this case, this prevents malicious users from accessing particular tables and records in the database and causing havoc in the process.

2.1.6 Query Optimizer

After determining that the client has the proper permissions to access the specific table in the database, the query is then subjected to optimization. MySQL uses the query optimizer for executing SQL queries as fast as possible. As a result, this is the reason why the performance of MySQL is fast compared to other RDBMS's. The task of the MySQL query optimizer is to analyze the processed query to see if it can take advantage of any optimizations that will allow it to process the query more quickly. MySQL query optimizer uses indexes whenever possible and uses the most restrictive index in order to first eliminate as many rows as possible as soon as possible. Queries can be processed more quickly if the most restrictive test can be done first.

2.1.7 Execution Engine

Once the MySQL query optimizer has optimized the MySQL query, the query can then be executed against the database. This is performed by the query execution engine, which then proceeds to execute the SQL statements and access the physical layer of the MySQL database. As well the database administrator can execute commands on the database to perform specific tasks such as repair, recovery, copying and backup, which it receives from the DDL compiler.

2.1.8 Scalability/Evolvability

The layered architecture of the logical layer of the MySQL RDBMS supports the evolvability of the system. If the underlying pipeline of the query processor changes, the other layers in the RDBMS are not affected. This is because the architecture has minimal sub-component interactions to the layers above and below it, as can be seen from the architecture diagram. The only sub-components in the query processor that interact with other layers is the embedded DML preprocessor, DDL compiler and query parser (which are at the beginning stages of the pipeline) and the execution engine (end of the pipeline). Hence, if the query preprocessor security/integration manager and/or query optimizer is replaced, this does not affect the outcome of the query processor.

2.2.1 Transaction Manager

As of version MySQL 4.0.x, support was added for transactions in MySQL. A transaction is a single unit of work that has one or more MySQL commands in it. The transaction manager is responsible for making sure that the transaction is logged and executed atomically. It does so through the aid of the log manager and the concurrency-control manager. Moreover, the transaction manager is also responsible for resolving any deadlock situations that occur. This situation can occur when two transactions cannot continue because they each have some data that the other needs to proceed. Furthermore, the transaction manager is responsible for issuing the COMMIT and the ROLLBACK SQL commands. The COMMIT command commits to performing a transaction. Thus, a transaction is incomplete until it is committed to. The ROLLBACK command is used when a crash occurs during the execution of a transaction. If a transaction were left incomplete, the ROLLBACK command would undo all changes made by that transaction. The result of executing this command is restoring the database to its last stable state.

2.2.2 Concurrency-Control Manager

The concurrency-control manager is responsible for making sure that transactions are executed separately and independently. It does so by acquiring locks, from the locking table that is stored in

memory, on appropriate pieces of data in the database from the resource manager. Once the lock is acquired, only the operations in one transaction can manipulate the data. If a different transaction tries to manipulate the same locked data, the concurrency-control manager rejects the request until the first transaction is complete.

### 2.3 Recovery Management

#### 2.3.1 Log Manager

The log manager is responsible for logging every operation executed in the database. It does so by storing the log on disk through the buffer manager. The operations in the log are stored as MySQL commands. Thus, in the case of a system crash, executing every command in the log will bring back the database to its last stable state.

#### 2.3.2 Recovery Manager

The recovery manager is responsible for restoring the database to its last stable state. It does so by using the log for the database, which is acquired from the buffer manager, and executing each operation in the log. Since the log manager logs all operations performed on the database (from the beginning of the database's life), executing each command in the log file would recover the database to its last stable state.

### 2.4 Storage Management

Storage is physically done on some type of secondary storage, however dynamic access of this medium is not practical. Thus, all work is done through a number of buffers. The buffers reside in main and virtual memory and are managed by a Buffer Manager. This manager works in conjunction with two other manager entities related to storage: The Resource Manager and the StorageManager.

#### 2.4.1 Storage Manager

At the lowest level exists the Storage Manager. The role of the Storage Manager is to mediate requests between the Buffer Manager and secondary storage. The Storage Manager makes requests through the underlying disk controller (and sometimes the operating system) to retrieve data from the physical disk and reports them back to the Buffer Manager.

#### 2.4.2 Buffer Manager

The role of the Buffer Manager is to allocate memory resources for the use of viewing and manipulating data. The Buffer Manager takes in formatted requests and decides how much memory to allocate per buffer and how many buffers to allocate per request. All requests are made from the Resource Manager.

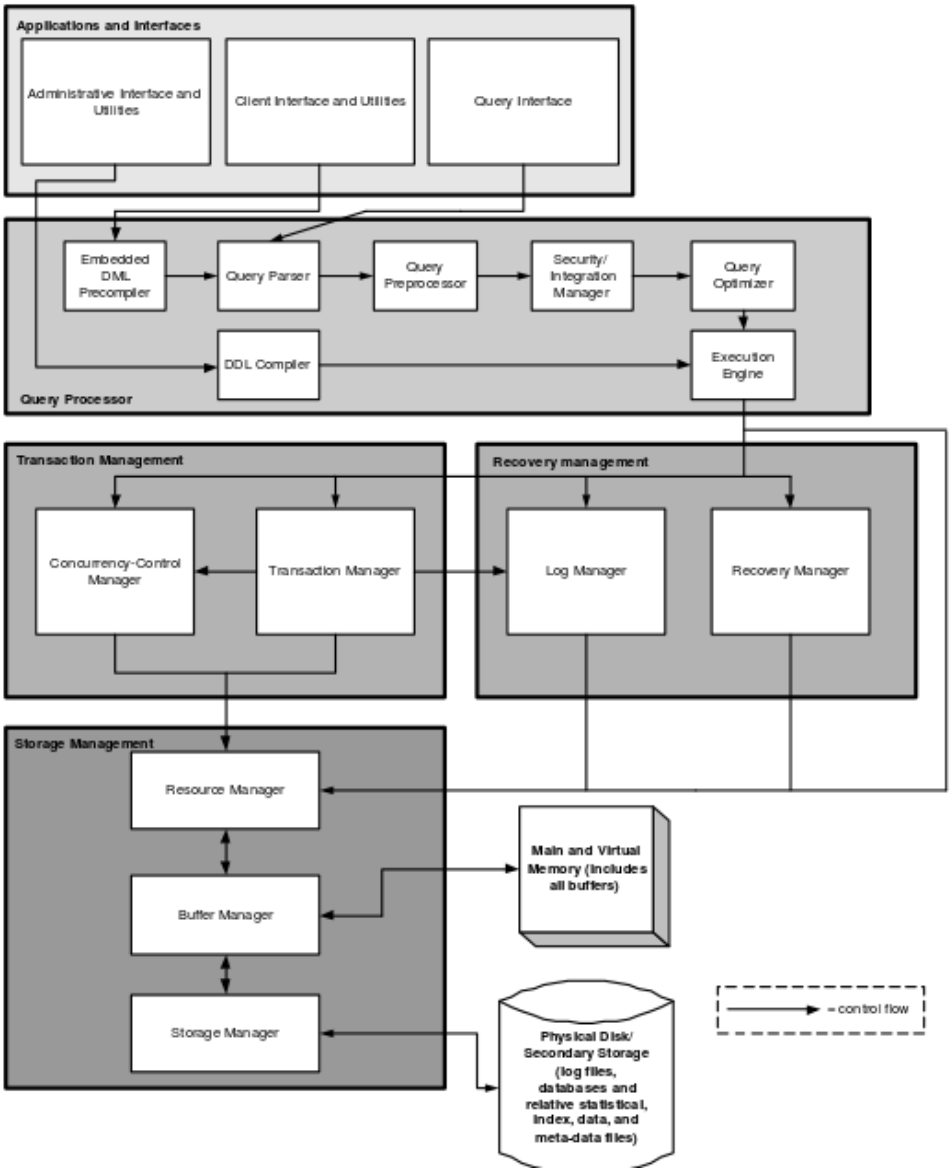
#### 2.4.3 Resource Manager

The purpose of the Resource Manager is to accept requests from the execution engine, put them into table requests, and request the tables from the Buffer Manager. The Resource Manager receives references to data within memory from the Buffer Manager and returns this data to the upper layers.

### 2.5 Evolvability/Scalability

The goals of the Transaction Management subsystem and the Recovery Management subsystem seem to provide non-functional requirements such evolvability and scalability. For example, the different managers provide the necessary abstractions so that the implementation can change while leaving the interface the same, thereby ensuring that the system can evolve to contain better data structures or algorithms. Furthermore, these subsystems provide scalability by being able to handle several transactions from several different users concurrently, or by recovering crashes from several different databases without much effort.

## DATABASE ARCHITECTURE:




## CONCLUSION:

Through this assignment, we have learned the basic database architecture and various components associated with it.