MySQL Conceptual Architecture

Ryan Bannon (mrbannon@uwaterloo.ca)
Alvin Chin (achin@swen.uwaterloo.ca)
Faryaaz Kassam (kassam@swen.uwaterloo.ca)
Andrew Roszko (andrewroszko@hotmail.com)

CS 798
Professor Ric Holt
2002-01-27

Table of Contents

1.0 Abstract

This paper presents a proposed conceptual architecture for the MySQL Relational Database Management System (RDBMS). The paper first goes into a brief examination of general RDBMS's. This research serves as a "roadmap" for further architectural discovery. It is a good assumption that the architecture for MySQL is similar to that of a general RDBMS architecture. Beyond this, MySQL documentation and literature specific to MySQL was examined to assist in defining the actual conceptual architecture of MySQL.

The MySQL architecture was first broken down into three layers: an application layer, a logical layer, and a physical layer. The logical layer, which is focused upon in detail, was broken down into four components: a query processor, a transaction management system, a recovery management system, and a storage management system.

It should be noted that recovering a conceptual architecture has a noticeable envelope for inaccuracy; putting the pieces of the puzzle together depends not only on facts, but also on relative intuition and logic, which may vary between the realization of the researchers and the designers. Thus, there should never be a guarantee that a conceptual architecture is completely correct; rather, someone who had no part in development should use the conceptual architecture as a general interpretation of the true architecture for MySQL.

2.0 Introduction and Overview

The purpose of this report is to extract the conceptual architecture of the MySQL open-source RDBMS 4.0, based from existing MySQL documentation and the general architecture of RDBMS's. There is no actual architectural diagram for the MySQL RDBMS provided by the designers. The conceptual architecture shows the information about what the system does and how this is broken into interacting parts. The MySQL RDBMS is a very popular database system that is being used all over the world for its performance, simplicity and robustness. (It should be noted that this document attempts to obtain the conceptual architecture of MySQL 4.0, which incorporates transaction and recovery functionality.)

This report is organized into the following sections. Section 1.0 is an Abstract that provides a brief overview of the key points in this report. Section 2.0 is the Introduction and Overview (this section) that basically summarizes the purpose of the report, its organization and conclusions. Section 3.0, following this section, is the General RDBMS Architecture that provides a high-level overview of the architecture of a RDBMS. From here, section 4.0 is the MySQL Architecture. This section provides the details of the high-level layered architecture of an RDBMS pertaining specifically to MySQL. Within each layer, we go deeper into the sublayers that characterize the interactions in the MySQL RDBMS system. Finally, we drill a level deeper and provide a diagram of the components and interaction between them. These components are described in detail as to their functionality and their relationship with other components in the same layer. This diagram is extracted based from reading documentation and books on MySQL and then mapping them onto the general architecture of an RDBMS found from RDBMS concept books. For each layer, we discuss how the architecture we prescribe is subject to system evolvability. Our conceptual architecture is using the logical view of the 4+1 Architectural Views paper by Phillipe Kruchten. Scenarios are then presented to the reader in section 5.0, which provide a walkthrough of the conceptual architecture to illustrate the interaction between the various components within the layer and communication among layers. Data dictionary section 6.0 provides a glossary of terms used within this report and their meaning. Finally, references are listed from which the reader can search and explore more information, and from which our research is based on.

Extracting the conceptual architecture from a RDBMS such as MySQL is not a trivial task. It involves reading lots of documentation on the system itself as well as external references to the theoretical structure or general structure of the system. There really is no guided way to extract a conceptual architecture. The conceptual architecture is just a mental view as to how the system works and is basically a graphical representation of the system to help understand its functionality.

3.0 General RDBMS Architecture

It was found in all of the consulted resources that all database systems, can be viewed as a Garlan & Shaw layered architecture at the highest level of abstraction. It has three main components as outlined in the following diagram:

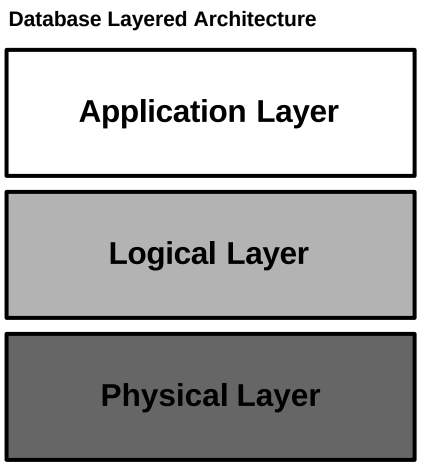

Figure 1: Database Layered Architecture

3.1 Application Layer

The application layer represents the interface for all the users of the system; it essentially provides the means by which the outside world can interact with the database server. In general, it has been found that users can be categorized into four main groups:

1. **Sophisticated users** interact with the system without using any form of application; they form their requests directly with the use of a database query language.
2. **Specialized users** are application programmers who write specialized database applications that do not fit into the traditional data-processing framework.
3. **Naïve users** are unsophisticated users who interact with the system by invoking one of the permanent application programs that have been previously written.
4. **Database Administrators** have complete control over the entire database system. They have a wide variety of responsibilities, including schema definition, the granting of access authorization, as well as the specification of integrity constraints.

All database systems provide extensive services for each of these groups; it is then left to the logical layer to appropriately process the varying requests.

3.2 Logical Layer

The core functionality of the RDBMS is represented in the logical layer of the architecture; it is in this portion of the system that there are a wide variety of vendor specific implementations (see Figure 2). However, upon reading a number of general database management texts, it was found that general core logical functionality can indeed be abstracted. The following diagram is a high level abstraction of the modules found in any robust RDBMS.

High Level Logical MySQL Conceptual Architecture

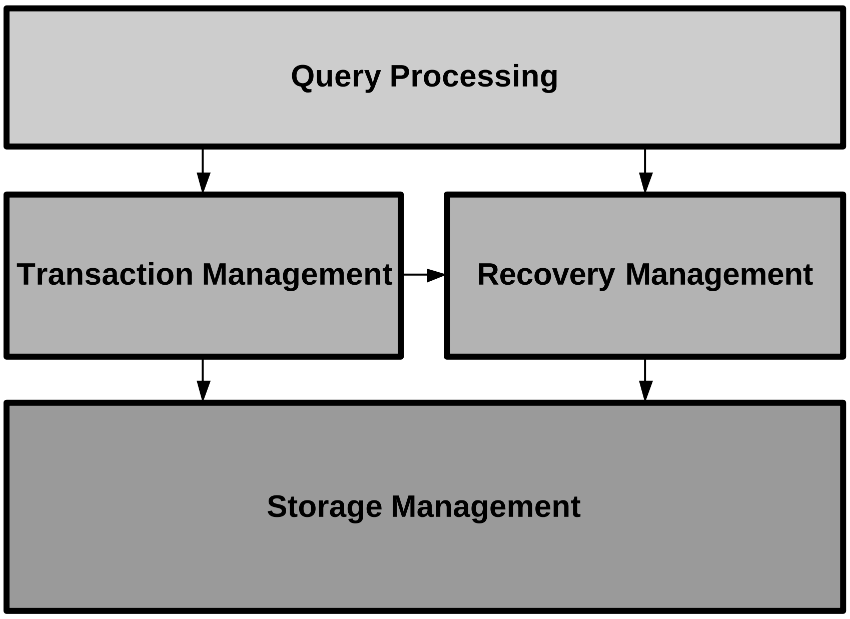

Figure 2: General High Level RDBMS Logical Modules

It can be observed that there are four main modules present in the RDBMS; their implementations and interactions have been summarized based on general documentation from a variety of sources. It is interesting to note that since the major control flow is indeed downwards, the logical layer itself can be considered a Garlan & Shaw layered architecture. It should be noted that the specific implementations and interactions between the subsystems vary greatly from vendor to vendor; in the ensuing section, the MySQL subsystems of these components will be analyzed in detail.

3.3 Physical Layer

The RDBMS is responsible for the storage of a variety of information, which is kept in secondary storage and accessed via the storage manager. The main types of data kept in the system are:

1. **Data files**, which store the user data in the database
2. **Data dictionary**, which stores metadata about the structure of the database
3. **Indices,** which provide fast access to data items that hold particular values
4. **Statistical Data,** which store statistical information about the data in the database; it is used by the query processor to select efficient ways to execute a query.
5. **Log Information,** used to keep track of executed queries such that the recovery manager can use the information to successfully recover the database in the case of a system crash.

4.0 MySQL Architecture

Once the research pertaining to general RDBMS architectures was complete, the specific MySQL documentation could then be examined in order to extract the vendor specific conceptual architecture. The ensuing section begins by describing the detail of the layered architecture from Figure 2 and how the specific MySQL components map onto Figure 1. As mentioned above, however, the core of the functionality is found in the logical layer; the remainder of the section details the key subsystems in this layer. The conceptual architecture of MySQL is illustrated in Figure 3 below.

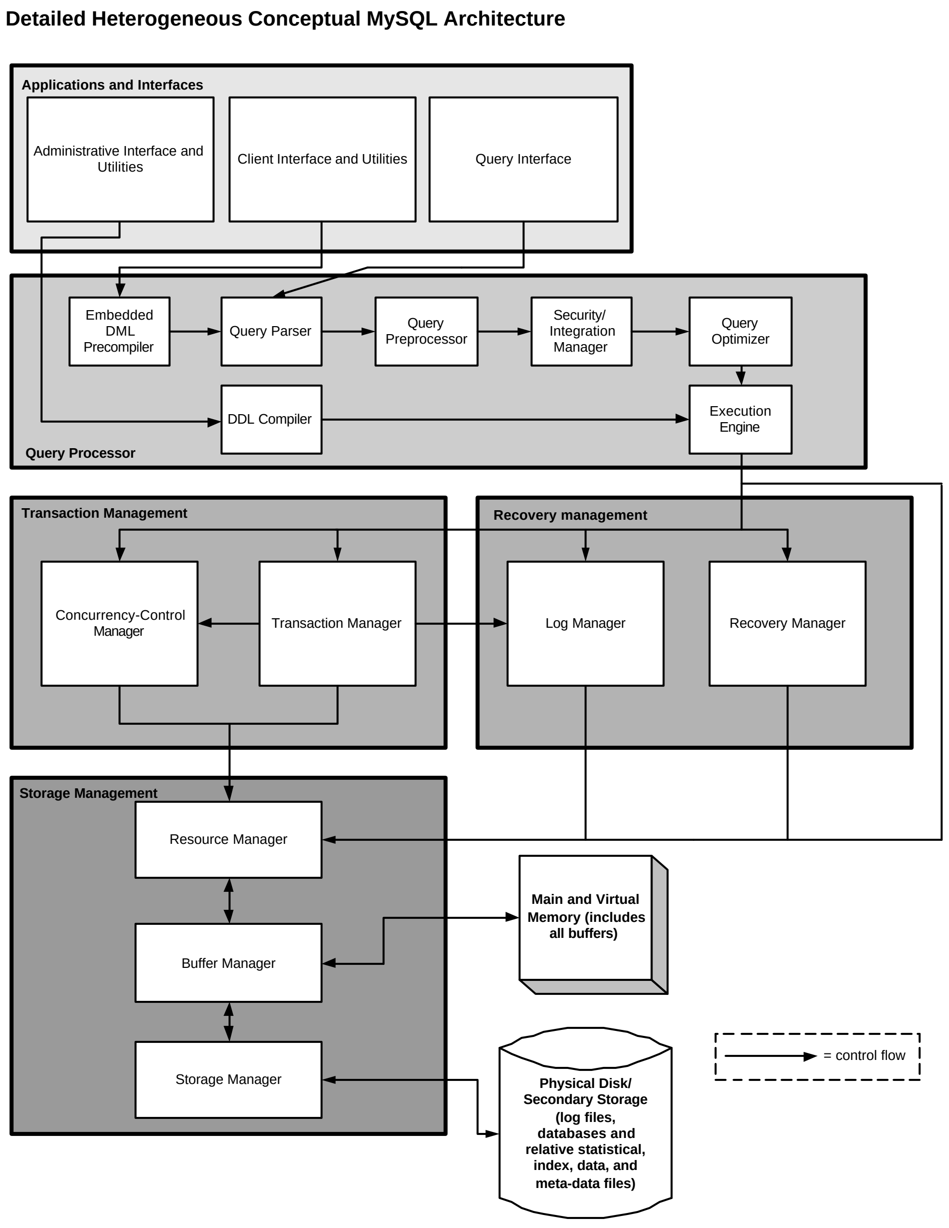
Detailed Heterogeneous Conceptual MySQL Architecture
Applications and Interfaces
Administrative Interface and Utilities
Client Interface and Utilities
Query Interface
Embedded DML Precompiler
Query Parser
Query Preprocessor
Security/ Integration Manager
Query Optimizer
DDL Compiler
Execution Engine
Query Processor
Transaction Management
Recovery management
Concurrency-Control Manager
Transaction Manager
Log Manager
Recovery Manager
Storage Management
Resource Manager
Buffer Manager
Storage Manager
Main and Virtual Memory (includes all buffers)
Physical Disk/ Secondary Storage (log files, databases and relative statistical, index, data, and meta-data files)
= control flow

Figure 3: Detailed Heterogeneous Conceptual MySQL

The architecture depicted in Figure 3 is a view of the control flow of the MySQL system. It is an expansion of the simple architecture described in Figure 2. It should be noted that the architecture described is a layered architecture as described by Garlan and Shaw. There exists a pipeline architecture, also described by Garlan and Shaw, represented in the Query Processing layer between the Embedded DML Precompiler and the Execution Engine.

For sake of simplicity, flow back up the architecture has been left out and should be deemed as implicit. For example, calling a simple SQL command, such as SELECT *, would require information to be brought back up the system. This flow is implied and is not mentioned in the diagram. Each layer in Figure 3 is described below with the core of the functionality found in the Logical Layer. As a result, this is the layer that will be directly explained in further detail.

4.1 Application Layer

The 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 MySQL architecture diagram in Figure 3. 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 *mysqladmin* which performs tasks like shutting down the server and creating or dropping databases, *isamchk* and *myisamchk* which help to perform table analysis and optimization as well as crash recovery if the tables become damaged, and *mysqldump* for backing up the database or copying databases to another server. 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.

4.2 Logical Layer

It was found that MySQL does indeed have a logical architecture that is virtually identical to the one depicted in Figure 2. 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.

4.2.1 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.

4.2.1.1 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.

4.2.1.2 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.

4.2.1.3 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.

4.2.1.4 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.

4.2.1.5 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.

4.2.1.6 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.

4.2.1.7 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 from Figure 3. 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.

4.2.1.8 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.

4.2.2 Transaction Management

4.2.2.1 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.

4.2.2.2 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.

4.2.3 Recovery Management

4.2.3.1 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.

4.2.3.2 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.

4.2.4 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 Storage Manager.

4.2.4.1 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.

4.2.4.2 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.

4.2.4.3 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.

4.2.5 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.

5.0 Use Case Scenarios

In this section, scenarios are used to illustrate sample use cases which walkthrough the interactions between those various layers and the interaction with the components within the layer described in the previous section. A selection of three scenarios will be used to demonstrate the control flow from Figure 3.

5.1 Crash Scenario

One scenario involves the crashing of a MySQL database with the database administrator having to recover this database.

1) The database administrator, at the application layer, uses the *isamchk* or *myisamchk* utilities that have a component to perform crash recovery.
2) This crash recovery utility is interpreted in the logical layer of the MySQL RDBMS by the query processor. Specifically, the command to execute a recovery is translated by the DDL compiler and is executed by the execution engine.
3) The execution engine passes the control that a recovery needs to be done on a particular database to the recovery manager of the recovery management sublayer.
4) The recovery manager finds out that it needs to retrieve the log file for that particular crashed database and it tells the resource manager in the storage management sublayer to grab the log file.
5) The log file pertaining to the crashed database from the physical storage is passed to the buffer manager.
6) The resource manager passes the log over to the recovery manager
7) The recovery manager starts from the beginning of the log and interprets each SQL command in the log.
8) Each SQL command in the log from beginning to end is executed on the database by the execution engine in the query processor.

5.2 Update Table Under Transaction Scenario

The transaction is considered to be a query and thus hooks into the application layer at the query interface specified in figure 3.

1) Passes through the pipeline of the query processor outlined above: it is parsed, preprocessed, access rights and integrity constraints are verified, and it is finally optimized before finally reaching the execution engine.
2) Noting that it is a transaction, the execution engine passes the request off to the transaction manager.
3) The transaction manager communicates with the log manager to log the information.
4) Log manager communicates with the buffer manager to update the log.
5) In order to simulate the serialization of the transactions, the transaction manager then communicates with the concurrency control manager (scheduler) in order to obtain an appropriate lock on the table.
6) Once the transaction manager logged the request and locked the appropriate table, the scheduler can then go about performing the update.
7) It issues a sequence of small requests for records of a relation to the resource manager, which knows about the data files, the format and size of the records in those files, as well as the index files.
8) The resource manager translates the requests for data into pages and then passes these requests to the buffer manager.
9) As detailed in the storage management portion of the report, the buffer manager can then read in the appropriate page from the disk via the storage manager.
10) Once the information is in memory, the scheduler can successfully update the data.
11) When the commit transaction command is reached, the scheduler releases the lock on the table.

6.0 Glossary

API (Application Programming Interface): A set of functions in a particular programming language issued by a client that interfaces to a software system.

Atomically : The "all or nothing" execution of transactions.

Disk Controller: A circuit or chip that translates commands into a form that can control a hard disk drive.

DML (Data Manipulation Language): this is a generic language in which the actual SQL statements are embedded in the client or translated from the client code.

DDL (Data Definition Language): This is the language that the RDBMS understands. In MySQL, and all SQL databases, the DDL is SQL itself.

Deadlock A failure or inability to proceed due to two transactions having some data that the other needs.

Index: A means to speed up access to the contents in database tables. The MySQL query optimizer takes advantages of indexes in order to speed up the processing of queries.

Main Memory: The storage device used by a computer to hold the currently executing program and its working data for fast access.

Meta-data: Data whose purpose is to represent the structure and meaning of the actual data.

Query Optimizer: Component in the Query Processor whose primary purpose is to optimize the queries so that they can be processed faster.

Query Precompiler: Processes the client code from the application layer to extract the SQL statements or translate the code into SQL statements.

Query Preprocessor: Performs the checking of syntax and semantics of the query before the query is processed, optimized and executed.

RDBMS (Relational Database Management System): A system that pro vides the management, storage and handling of requests to a relational database.

Transaction: One or more commands grouped together into a single unit of work.

Virtual Memory: Memory, often as simulated on a hard disk, that emulates RAM, allowing an application to operate as though the computer has more memory than it actually does.

7.0 References

1. Atkinson, Leon. Core MySQL: The Serious Developer’s Guide. New Jersey: Prentice Hall Publishing, 2002.
2. Date, C.J. An Introduction to Database Systems. Menlo Park: Addison-Wesley Publishing Company, Inc, 1986.
3. Dubois, Paul. MySQL. New York: New Riders Publishing, 2000.
4. Frost, R.A. Database Management Systems. New York: Granada Technical Books, 1984.
5. Garcia-Molina, Hector. Database System Implementation. New Jersey: Prentice Hall, 2000.
6. Garlan, David. Shaw, Mary. "An Introduction to Software Architecture". Pittsburgh, PA USA: School of Computer Science, Carneigie Mellon University, 1994.
7. Kruchten, Phillippe. "Architectural Blueprints – The 4+1 View Model of Software Architecture". Rational Software Corp, 1995.
8. Silberschatz, Abraham et al. Database System Concepts. New York: McGraw-Hill, c1997.
9. U.S. Department of Commerce. "An Architecture for Database Management Standards". Washington: U.S. Government Printing Office, 1982.
10. "http://www.mysql.com/". MySQL AB, 2002.
11. Yarger, Randy Jay MySQL & mSQL. Sebastopol: O’Reilly & Associates, 1999.