York U EECS 4314: Advanced Software Engineering

MySQL Conceptual Architecture

Ante Pimentel (212166187)
Dusan Birtasevic (210451383)
Francis Okoyo (213811336)
Hashim Al-Helli (212172359)
Richard Van (212918652)
18th October, 2017

Abstract

This document focuses solely on the conceptual architecture of MySQL version 8.0.2. A detailed description of the high level architecture is given followed by a general analysis of the subsystems. Within the descriptions, architecture styles and design patterns are identified with the use of diagrams and use cases. Additionally, concurrency present in MySQL is discussed followed by a discussion on the division of responsibilities for developers and their associated implications. Lessons learned and recommendations for future versions are also provided.

Table of Contents

Introduction **2**

System Functionality **2**

1) General RDBMS Architecture 2
2) MySQL Architecture 3
    2.1) Application Layer 3
    2.2) Logical Layer 3
        2.2.1) Query processor 4
            2.2.1.1) Embedded DML precompiler 6
            2.2.1.2) DDL Compiler 6
            2.2.1.3) Query Parser 6
            2.2.1.4) Query Preprocessor 6
            2.2.1.5) Security Integration Manager 6
            2.2.1.6) Query Optimizer 6
            2.2.1.7) Execution Engine 6
        2.2.2) Transaction management 7
            2.2.2.1) Transaction Manager 7
            2.2.2.2) Concurrency-Control Manager 7
        2.2.3) Recovery management 7
            2.2.3.1) Log Manager 7
            2.2.3.2) Recovery Manager 7
        2.2.4) Storage management 8
            2.2.4.1) Storage manager 8
            2.2.4.2) Buffer manager 8
            2.2.4.3) Resource manager 8
    2.3) Physical layer 8

Concurrency **9**

MySQL Server Connections 9
Simultaneous Queries and Transactions 9

Developer Division of Responsibilities **10**

Top-Level 10
Application Layer 11
Logical Layer 11
Physical Layer 11

Use Cases **12**

Lessons Learned **12**

References **13**

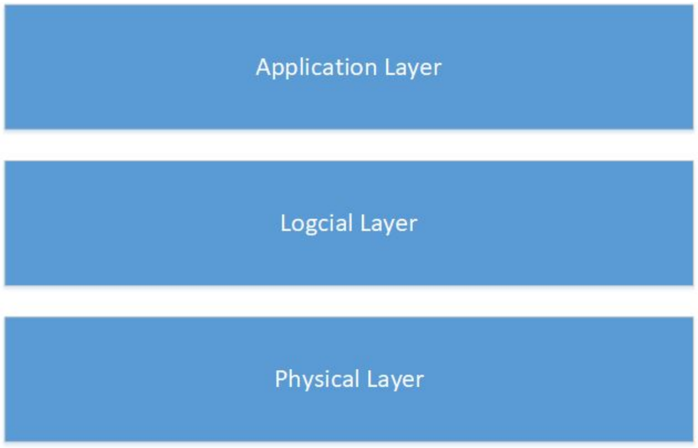
Introduction

This report was prepared for EECS 4314: Advanced Software Engineering during the fall 2017 semester. The authors of the document illustrate the conceptual architecture of MySQL version 8.0.2 through an analysis of the system functionality, pointing out concurrency present in MySQL, revealing the implications of developer divisions of responsibilities, defining use cases, and recommending improvements via lessons learned throughout the production of the report. Information derived for this report was obtained by searching the MySQL documentation pages within MySQL's reference manual, performing general search queries through various academic journals and databases, and by referring to the lecture slides for EECS 4314 provided by professor Zhen Ming (Jack) Jiang. To better illustrate the conceptual architecture of MySQL and the relationship of its subsystem sequence diagrams and state diagrams are provided, which were created using Microsoft Visio.

System Functionality

1) General RDBMS Architecture

In general, RDBMS can be depicted as a layered style architecture which consists of 3 layers. Application, Logical and Physical.



The **application layer** is essentially an interface that is used by the outside world to communicate and interact with the database server. For example; let's say we have and employee database. This database may contain information on several employees. Information like phone numbers, addresses, etc. there exists a secretary (user) that would like to access said information. To do so, he/she would interact with the application layer. The **logical layer** is where all the magic happens. Data received from the application layer is handled by this layer. The logical layer is broken into smaller sub layers that work together to process the requests from the application layer. It is important to note that in general, the design of the logical layer in RDBMS' vary from version to version. The **physical layer** is responsible for storing information and data received from the storage manager (bottom sublayer of the logical layer). Usually stores Data files, Statistical Data and Log information.

2) MySQL Architecture

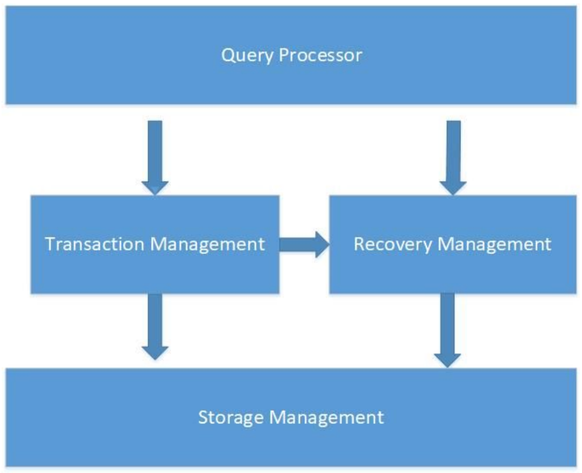
The MySQL layered architecture is similar to that of the generalized architecture. With that said there are still some differences which we will go into detail on in this section.

2.1) Application Layer

Enables clients and users to interact with the MySQL database system. Thus MySQL follows a client server architecture. The application layer consists of the Administrative interface and utilities, client interface and utilities, query interface. These 3 parts give us insight as to what kind of users will be using the system. **Administrators** use administrative utilities for management of the system. Things like server shutdown, creation or termination of databases, managing or transferring databases. These users have complete control over the database. **Clients** (sophisticated users) interact with MySQL through the client interface and utilities like MySQL API's which allow them to connect to the database server. **Query** (naïve) users interact with MySQL through the query interface, these users issue SQL statements (commands/queries) to the server and examine the results.

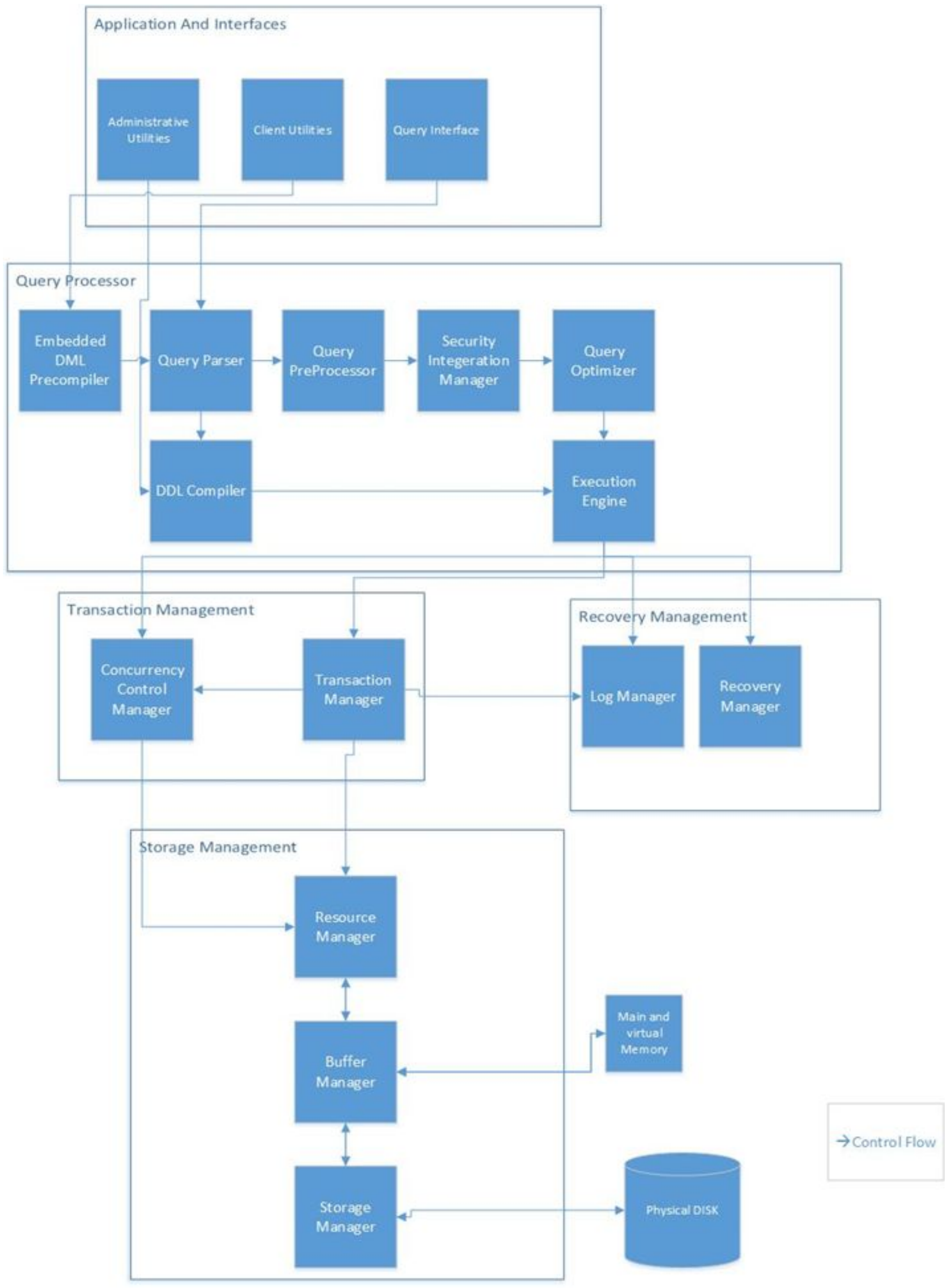
2.2) Logical Layer

Similar to that of the general RDBMS logical layer, the MySQL logical layer consists of 4 sublayers. Query processor, Transaction management, Recovery management, and Storage management. The logical layer follows the pipeline architecture style.



2.2.1) Query processor

The majority of the interactions between the users and the system occur through here. This component can be depicted as a pipe and filter type architecture where the output on the previous component becomes the input of the next.



The query processor consists of the Embedded DML precompiler, DDL compiler, Query parser, Query preprocessor, Security Integration Manager, Query optimizer, and execution engine. Its design is based off of the façade design pattern.

2.2.1.1) Embedded DML precompiler

The embedded DML (Data Manipulation Language) precompiler is responsible for extracting the SQL statement embedded within the client API commands, or to translate the client commands into the corresponding SQL statements[2].

2.2.1.2) DDL Compiler

The DDL (Data Definition Language) compiler compiles the DDL commands. DDL commands are SQL statements that create the structure of the database. DDLs are mostly used by administrators because they are the ones that create the structure of the database[2].

2.2.1.3) Query Parser

MySQL parses queries to create an internal parse tree which is then passed on to other components for processing[7].

2.2.1.4) Query Preprocessor

The query preprocessor takes the query parse tree obtained from the query parser, then uses it to check the SQL syntax and check if the query is valid. If it is a valid query, then the query continues down the pipeline. If not, then the query processing stops and a processing error is issued to the client[2].

2.2.1.5) Security Integration Manager

The job of the security integration manager is to check whether a client has the privileges to execute a certain query against the database[2].

2.2.1.6) Query Optimizer

MySQL uses the query optimizer to determine the most efficient way to execute SQL queries. This may include rewriting the query, the order of scanning the tables, and/or choosing the right indexes to use[2].

2.2.1.7) Execution Engine

The query execution engine executes the SQL query on the database[2].

2.2.2) Transaction management

A transaction is a group of SQL commands that are executed as a single unit of work. The transaction management layer consist of the Transaction Manager, and Concurrency-Control Manager[2].

2.2.2.1) Transaction Manager

Makes sure that transactions are logged and executed atomically meaning a transaction can either success or fail and that no partially completed transactions occur this means that the transaction cannot pass unless all the commands included in it are completed successfully. The transaction manager is also responsible for resolving deadlock situations if they arise. Finally the transaction manager is responsible for issuing COMMIT and ROLLBACK commands[2][7].

2.2.2.2) Concurrency-Control Manager

The concurrency-control manager is responsible for making sure that transactions are executed separately and independently[2].

2.2.3) Recovery management

This layer consist of Log Manager, and Recover Manager.

2.2.3.1) Log Manager

The log manager is responsible for logging every operation executed in the database. It does so by storing the log file on disk. The operations in the log are stored as MySQL commands which in case of a system crash, executing every command in the log will bring back the database to its last stable state[2].

2.2.3.2) Recovery Manager

In event of a crash. The recovery manager is responsible for restoring the database to its last stable state. It uses the log created by the log manager to restore the database by executing every command in it since the log file contains all operations performed since the creation of the database[2].

2.2.4) Storage management

The storage management layer contains Storage Manager, Buffer Manager, and Resource Manager. Storage is done on some type of secondary storage, however access to this storage is not practical due to slow speed. 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.

2.2.4.1) Storage manager

The main job of the Storage Manager is efficient data write to the disk. It does so by writing to the disk all user tables, log information and internal system data[2].

2.2.4.2) Buffer manager

The role of the Buffer Manager is to allocate memory resources and decides how much memory to allocate per buffer and how many buffers to allocate per request[2].

2.2.4.3) Resource manager

The Resource Manager accepts 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[7].

The layered architecture of the logical layer of the MySQL RDBMS allows the system to evolve. If the components of query processor changes, the other layers in the MySQL RDBMS are not affected. This is because the architecture has low sub-component interactions to the layers above and below it, as it 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, query parser, and the execution engine. Hence, if the query preprocessor, security/integration manager, or query optimizer is replaced or changed, this does not affect the result of the query processor[2].

2.3) Physical layer

The final layer of this architecture contains storage engines which are responsible for storing and retrieving data. The main difference between the physical layer of the MySQL architecture and that of the general is its pluggable engine architecture, based off of the strategy design pattern. This allows storage engine to be loaded and unloaded from servers. There are many engines available but the main ones are MyISAM and InnoDB. InnoDB is the default storage engine in MySQL 5.5.5 and later as the CREATE TABLE statement in MySQL creates InnoDB tables by default. Each storage engine has different characteristics and based on the requirement of application we can choose an appropriate storage engine. When you create a table, you can choose what storage engine to use[7].

Concurrency

Concurrency in MySQL is handled by the concurrency control manager in logical layer of the conceptual architecture. Concurrency means allowing simultaneous queries to execute without queries interfering with each other. The concurrency control manager is responsible for ensuring queries run separately. Transactions represent a grouping of commands treated as a single query and are handled similarly to queries[2]. The handling of concurrent queries and transactions depend on the type of storage engine used.

MySQL Server Connections

MySQL allows multiple clients to access the system at the same time. When a client connects to the MySQL server, the server creates a thread to handle authentication and requests made[5]. Each connected client has a separate thread on the server. This is consistent regardless of the storage engine used, and is separate from the storage engine.

Simultaneous Queries and Transactions

In MySQL, storage engines handle how to manipulate data in the database. As storage engines are plugins that can be loaded and unloaded even during runtime, the method in which MySQL handles simultaneous queries is dependent on the storage engine loaded. MySQL’s capability of handling transactions at all depends on the storage engine as well, as some storage engines do not support transactions.

Many of the storage engines packaged with the MySQL server, such as InnoDB, MyISAM and Memory, use locks to implement concurrency[5]. When a client acquires a lock, no other client can access the locked data until the lock is released. Locks help achieve concurrency by ensuring only the currently executing query or transaction can access and manipulate locked data.

There are multiple types of locks which restrict access to different parts of the database, such as row locks, table locks, read (shared) locks and write (exclusive) locks. Different types of locks are used by different storage engines to achieve concurrency. Each approach has varying tradeoffs, such as higher speed of database operations at the cost of memory or reduced performance due to the overhead of locking operations[5].

Depending on the storage engine, it is possible to reach a deadlock situation when using locks where two clients hold a lock the other needs[5]. This is handled by the transaction manager which uses functionality from the storage engine to help resolve deadlocks.

Developer Division of Responsibilities

Top-Level

Mysql is a client-server system that uses a three tiered layered architecture at the top level, these layers are: the application layer, logical layer and physical layer [9]. This architecture therefore uses a three tiered approach as opposed to a 2 layered one by splitting the server side into the three layers; applications, logical, and physical. This helps MySql developers avoid the typical disadvantages of a client-server system, which is growing the architecture as computing needs grow and change, and the client-server'us almost non-existing use of object-oriented programming its associated benefits [9]. Designing the architecture this way helps developers as it removes the tight coupling of the user interface and database engine in the classic two-tiered approach to a client-server architecture, which would result in a system that is nearly impossible to scale and modify as the user base and data volume increases [9]. Another way this approach helps developers with scalability is by facilitating application changes in the database across multiple servers, and by taking data from the database to the GUI, and performing operations on the data solely in the GUI [9]. This helps developers as it converts previous thick clients to thin clients who perform operation on the client side as opposed to the server side. Breaking the client-server into three pieces also helps developers with code reuse as common code in data processing(business rules), no longer reside in the user-interface which would make reuse difficult [9]. In general, the most important advantage developers gain is the separation of the database from the user-interface. It is no longer necessary to build database knowledge into the

GUI, instead all knowledge of how to deal with the database can sit in the middle tier [9]. All adjacent layers in the architecture may communicate directly. The logical layer is below the application layer, and developers may therefore use components in the logical layer to request and obtain services from the application layer. This also applies to sub-layers which also follow the layered style, although some non-adjacent layers do communicate to each other directly to improve efficiency. On a deeper level however, there are more divisions of work that impact developers, such as within the application layer.

Application Layer

In this layer clients interact with the MySQL RDBMS, and so all services such as authentication, connection handling and security, (among others), is handled in this layer. The application layer contains three main components; Clients, Query Users, Administrators [1]. Clients use utilities and interfaces such as API's to communicate with MySQL. Query users use the query interface mysql to interact with MySQL RDBMS. Administrators perform server maintenance using administrative utilities and interfaces [1]. Also, with this design clients receive a unique thread for connection when they first connect to the server, and there is no need to create a new one for each forthcoming connection since the thread is cashed [7].

Logical Layer

The logical layer takes all data from the application layer and processes it. MySQL developers divided this layer into subsystems; query processor, recovery management, storage management, and transaction management [1]. They work in conjunction to process all requests received in the application manager for the database server. An important design of the developers was to have the query processor use a pipe and filter architecture style. This makes the query processor easy to understand and simple to reuse, it also allows for specialized analysis(deadlock/throughput), supports concurrent execution, and is easily maintained or enhanced for all current and future developers. Unfortunately choosing this style may lead to increased complexity or loss of performance if developers use a large amount of parsing when writing filters.

Physical Layer

This layer contains all storage engines(referred to as pluggable storage engines) available to developers [10]. MySQL was designed so that developers may use any storage engine for their specific needs. Since each layer is separated in the architecture, implementation details are hidden, the API is easy to use and the developer does not need to manage application coding requirements [10].

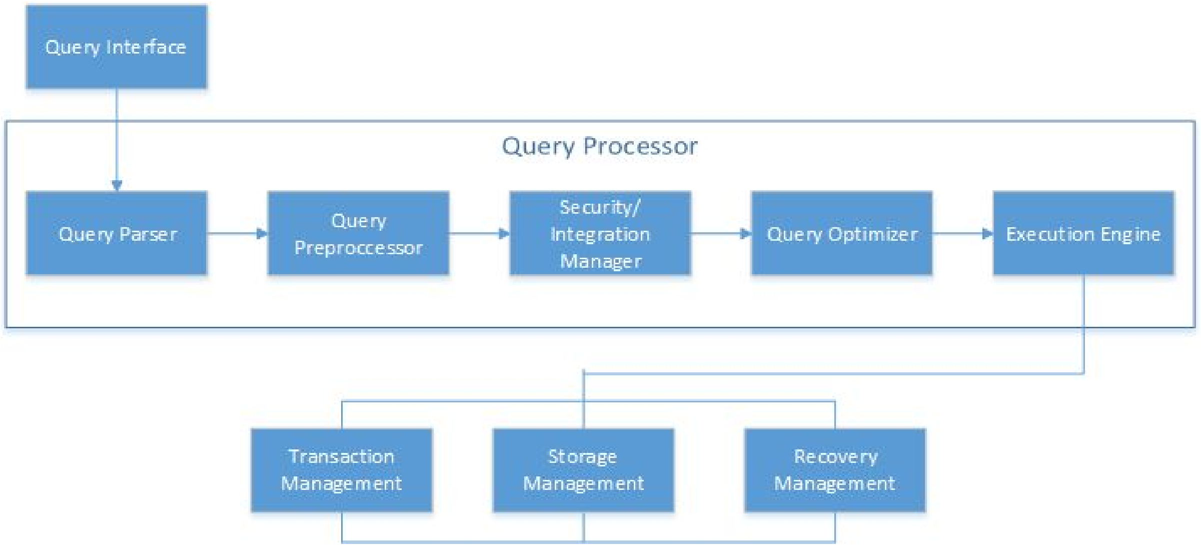
Use Cases

General UPDATE use case:

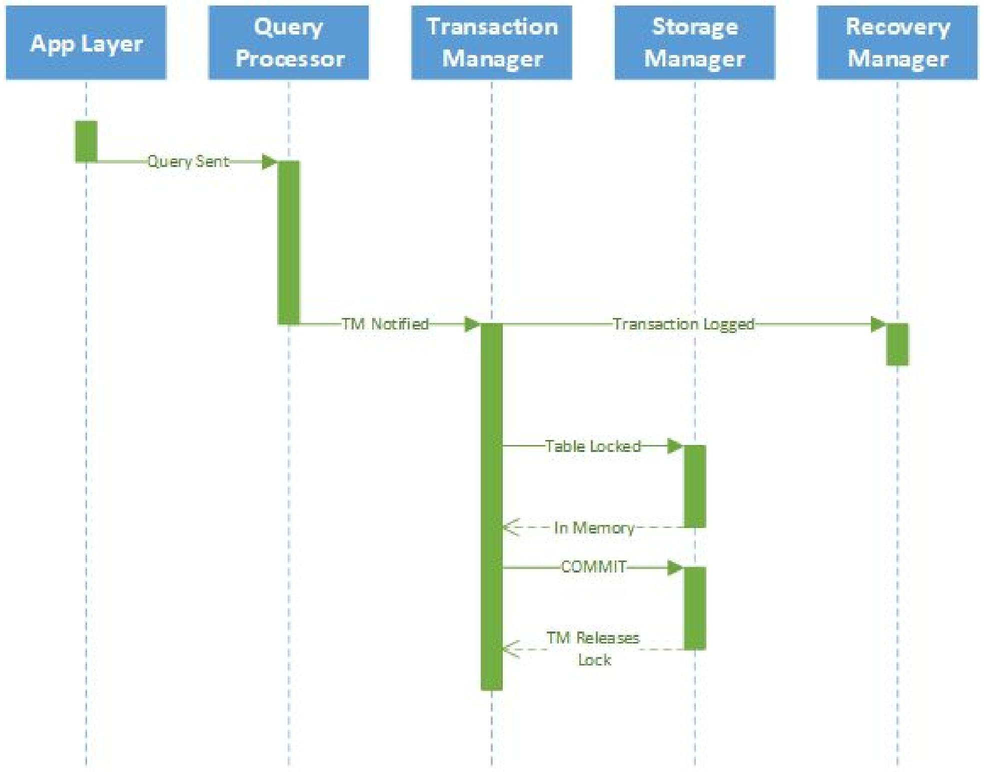
1) Query comes in from query interface
2) Goes through Query Parser to Execution Engine
    a) Creates Parse tree, for other components
    b) Checks semantics and syntax
    c) Checks permissions on user
    d) Optimizes query, checks for existing indexes
    e) Creates Execution plan
3) UPDATE is an action that can be committed, so it is a transaction. The Transaction manager is notified. Recovery Manager is notified, for logging purposes.
4) TM locks the relevant table/row
5) TM communicates with the Buffer Manager in Storage management to load the required data into memory
6) Once the data is in memory, it can be changed and committed back by the TM
7) TM releases the lock on the table/row

[11] [12]

Here is a simplified flow diagram. Some subsections have been cut or condensed to highlight only the relevant parts for this use case.



Here is sequence diagram the steps above. Note that recovery management is used many times through the process, it is only shown in the diagram to avoid clutter. It emphasizes the locking on the table and how the transaction manager controls it.



Lessons Learned

Organizing information into a format we could understand and explain was our main challenge in this assignment. This included deciphering and understanding technical jargon specific to RDBMS and MySQL. Some of our sources conflicted, so it was necessary to synchronize our information and ensure that our report was consistent with all of our sources. All authors gained a greater appreciation of proper documentation. Therefore we recommend that the developers of MySQL provide more documentation on their conceptual architecture including includes diagrams.

References

[1] Nellutia, I. (2017, January 20) MySQL Architecture and Layers. Retrieved from https://qxf2.com/blog/mysql-architecture-and-layers/.

[2] Bannon, R., & Chin, A., & Kassam, F., & Roszko, A. (2002, January 27). MySQL Conceptual Architecture. Retrieved from https://s2.smu.edu/~rkotamarti/mysql.pdf.

[3] MySQL Architecture (2013, December 22). Retrieved from http://mysqladvice.blogspot.ca/2013/12/mysql-architecture.html.

[4] Structure of Database Management System (DBMS). (n.d.). Retrieved from https://www.tutorialcup.com/dbms/structure-of-dbms.htm.

[5] MySQL 8.0 Reference Manual. (2017,.October 13). Retrieved from https://dev.mysql.com/doc/refman/8.0/en/.

[6] MySQL & mSQL. Retrieved from https://docstore.mik.ua/orelly/linux/sql/ch08_04.htm.

[7] Kumar, R. (2016, April 19). Understanding MySQL Architecture. Retrieved from http://www.rathishkumar.in/2016/04/understanding-mysql-architecture.html.

[8] Vuong, T. (2013, August 08). MySQL Architecture and Concepts. Retrieved from https://www.slideshare.net/TuyenVuongDinh/mysql-architectures.

[9] MySQL and mSQL The Three Tier Architecture. Retrieved from https://docstore.mik.ua/orelly/linux/sql/ch08_04.htm

[10] Overview of MySQL Storage Engine Architecture. Retrieved from https://dev.mysql.com/doc/refman/5.7/en/pluggable-storage-overview.html

[11] MySQL Transactions https://dev.mysql.com/doc/refman/5.7/en/glossary.html#glos_transaction

[12] MySQL Buffer Pool https://dev.mysql.com/doc/refman/8.0/en/innodb-buffer-pool-intro.html