# MySQL Conceptual Architecture

*EECS4314: Advanced Software Engineering*

*Tabs vs. Spaces*


**Authors**

Kevin Arindaeng

Anton Sitkovets

Glib Sitiugin

Nisha Sharma

Varsha Ragavendran

Ayman Abualsunun

Davood Anbarnam

# Table of Contents

# List of Figures

## Abstract

This document presents the conceptual architecture of the software system known as MySQL Server. An introduction and overview of the high level architecture is presented, and a discussion on the use of common architectural styles of each major subcomponent is provided. Use cases present the interaction between a user and the software system and the subsequent workflow through the various subcomponents in MySQL.

## Introduction

The following report presents the conceptual architecture of MySQL 8.0.2, an open-source Relational Database Management System (RDBMS). It is the top choice for several high-profile e-commerce applications such as Facebook, Twitter, YouTube, etc., and runs under many operating systems including, but not limited to, Linux, Windows, FreeBSD and Unix. The name MySQL is a combination of the name of one of the co-founder's daughter, and the abbreviation for Structured Query Language.

The functionality of MySQL is to provide the computations behind adding, modifying, accessing, managing, and processing a structured collection of data stored in databases. It provides the services for query processing, and data storage management. MySQL is relational, meaning that the data is stored in separate tables based on a structure optimized for speed, instead of one large table. These rules provided by MySQL are enforced by the database to ensure that there is no duplicate or inconsistent data.

The report begins with a look at MySQL's Architecture, illustrating and describing its use of the client-server and repository-style architectures and the interaction between its subsystems. Such subsystems are examined in more detail, where any architectural styles and design patterns are highlighted. A brief summary of MySQL's evolution from the preceding versions to the current version is then given, including the notable changes that came with each update.

After covering the internal interface of the system, the interaction between MySQL and the external environment is discussed. This includes defining the available APIs that can be used by clients to connect to the server and the process of handling clients connecting to the server, mentioning any available Graphical User Interfaces and the contributions made by developers to this open-source project. Some example scenarios are then presented, illustrating the workflow between MySQL's various subcomponents. To close out this section of the report, a glossary for various terms/phrases used throughout the document as well as any naming conventions used.

The report concludes with the major finding of this investigation. The first illustrates the need to break large scale systems like MySQL into smaller subsystems, resulting in a mix of architectural styles amongst such subsystems.

# MySQL Architecture

## High-Level Description

At the highest level, MySQL follows a client-server architecture. This is illustrated in the figure below:

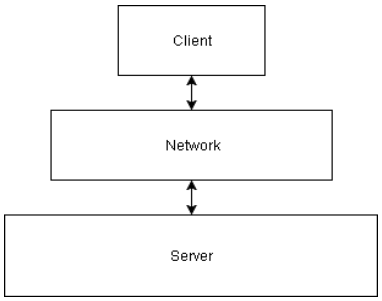


Figure 1 - MySQL's Client-Server architecture

This architecture allows for the bulk of computation to be handled by the server rather than the client application. In the case of MySQL, such computations fall under the more general term of Database Management, specifically Relational Database Management, due to it being a Relational Database Management System (RDBMS). Without going into too much detail, this means MySQL stores data in the form of *tables*, where each table has a relationship to another, and each entry in a table is a *row*.

Such systems typically handle multiple, concurrent requests from a variety of users, where each user either 'reads' or 'writes' to the database. MySQL is no different, hence why in addition to the Client-Server architecture, it also follows a repository-style architecture:

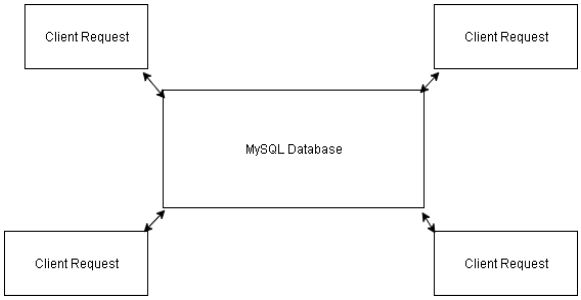


Figure 2 - MySQL's repository style architecture

The operations/computations MySQL carries out with regards to 'reading' and 'writing' are commonly referred to in the DBMS world as CRUD operations: Create, Read, Update, Delete.

## MySQL's Subsystems

Developing such a large, concurrent system is no easy task. Doing so inevitably requires the forming of several subsystems/modules, where each module is responsible for one task. These modules, as well as the data flow between them, are shown below in a layered architecture [1]:

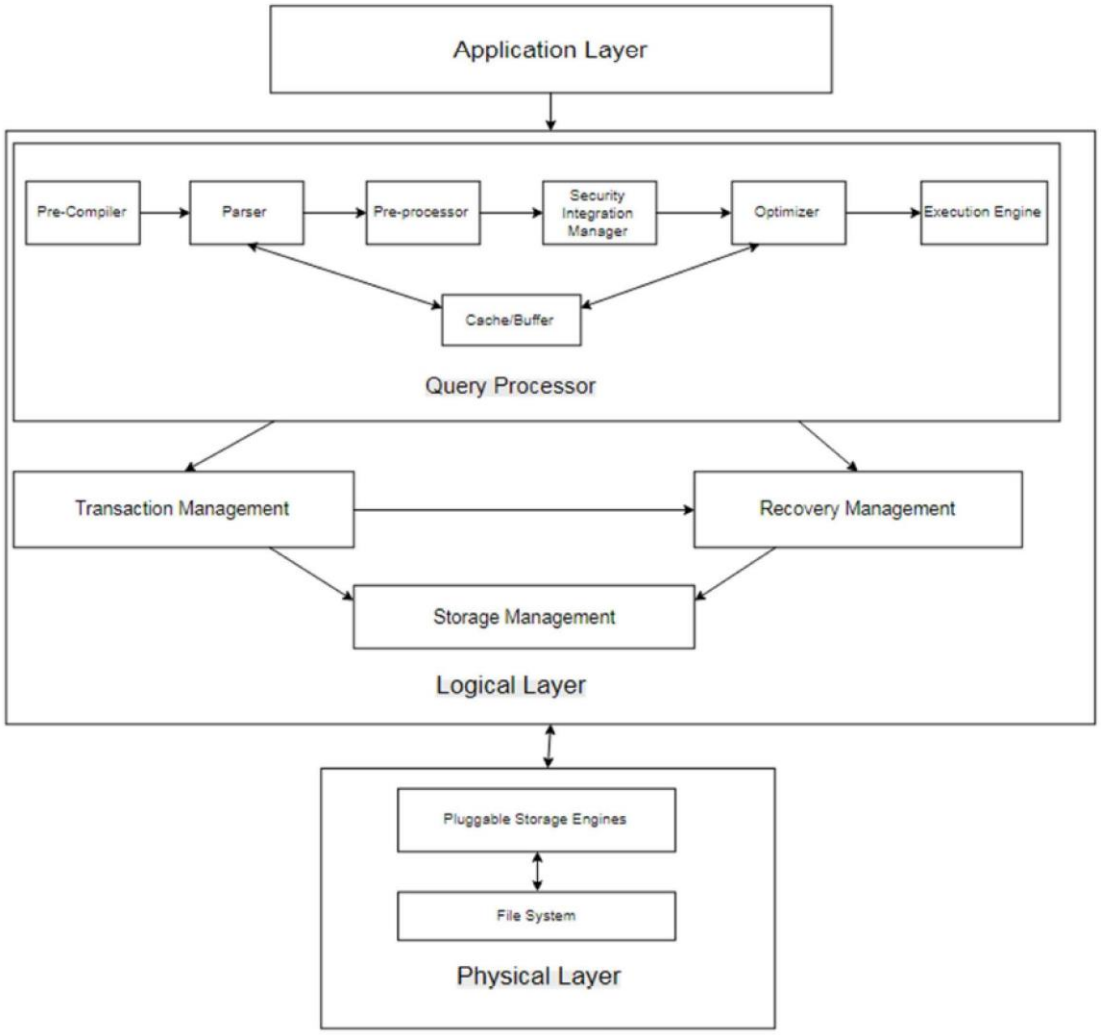


Figure 3 - MySQL Subsystems

The figure shows how MySQL takes a layered approach to managing its subsystems. This approach is referred to as a 'Layered Architecture', and is also used in computer

network protocols. The Logical Layer is where the majority of computation resides, and can be considered the heart of MySQL. The Physical Layer is where the data resides, i.e. disks, file systems and the software modules (storage engines) that they communicate with. The application layer simply refers to the client application.

## Query Processing

Clients communicate with MySQL by issuing queries. These queries follow SQL syntax and must first be compiled into an executable (i.e machine code). Doing so is a four-step process, accomplished with aid of the tools Lex & YACC [1]:

1. Breaking the query into its constituent components (Tokenizing)
2. Parsing each component/token (Syntax Check)
3. Optimizing the query where appropriate & performing a semantic check
4. Generating the executable

The figure shown below represents the architecture of the above process, referred to as the Hybrid Compiler Architecture, as it comprises of both a linear pipe-and-filter and repository-style architecture:

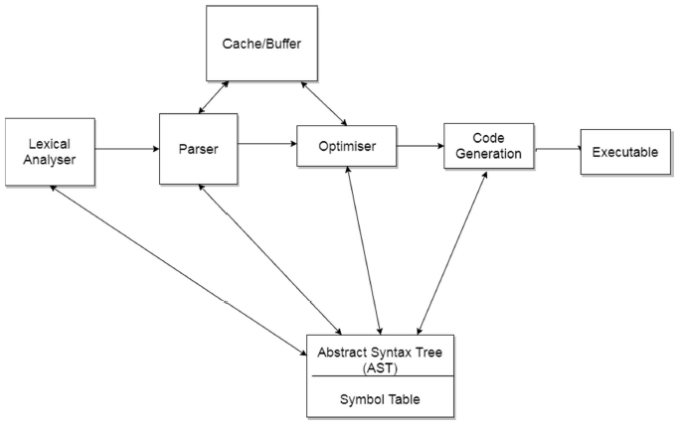


Figure 4 - MySQL compiler architecture

Steps 1 & 2 are handled by the Parser while the Optimizer handles Step 3. The Optimizer's workflow itself consists of Validation and Optimization, The former ensures any referenced data items, e.g. table names, column names, types, etc. are done so correctly. For instance, one cannot select a table "My_Table" if it does not exist. The latter looks to make changes to the query structure to improve time performance, similar to the function of programming language compilers.

The architecture, as a whole, follows the standard for compilers. The crucial difference, one that is unique to MySQL when compared to other RDBMS's, is the presence of a Cache/Buffer.

## Cache/Buffer

This module can be thought of a combination of various cache/buffer sub-modules [2]:

Query Cache

This cache allows for frequently used queries to skip the entire optimization and execution stages by caching the query structure and the results. It is also responsible for purging old data for when the result of a previously stored query has been updated.

Record/Row Cache

This cache aims to improve the sequential reading of tables. It consists of a read-ahead buffer that takes one block of data at a time. This results in less disk reads during the scan and improved performance. This cache is also used for writing data sequentially by first writing the data to the cache and then to the disk.

Key Cache

This is cache is stores frequently used indexes. When an 'index read' is executed, the key cache is first read: if the index exists in the cache we retrieve it from the cache, otherwise retrieve the index block from disk and store it in the cache. The key cache classifies how frequently an index is accessed over time according to the following terms: BLOCK_COLD, BLOCK_WARM, BLOCK_HOT. The 'hotter' a block, the more frequently the data is indexed. Updating the cache is done using a Least-Recently-Used (LRU) algorithm.

Table Cache

This cache is used to minimize time for opening, reading and closing tables. It stores in-memory metadata about the tables using a data chunking mechanism called Unireg. A list of table cache structures are maintained for each execution thread so that each has a local view of the table.

## Transaction Management

It is not always the case that a client application wishes to execute a series of queries sequentially. Rather, there are instances in which the opposite is true: the client application wishes to execute a series of queries in one go. This is referred to as a Transaction. Processing a transaction boils down to determining whether it passes the 'ACID' test or not [3]:

A - Atomicity

Every SQL query in a transaction must be a successful one

C - Consistency

Completing the transaction will not change the stored data in a forbidden manner

I - Isolation

The effects of a transaction are invisible until completed

D - Durability

Changes made to the database are permanent

### Concurrency

This is where the bulk of MySQL's concurrency support can be found. As clients may affect the database in only two ways, i.e. read & write, MySQL implements a locking system where[4]:

- Read locks on a resource are mutually non-blocking: many clients may read from a resource at the same time and not interfere with each other.
- Write locks are exclusive: they block both read locks and other write locks.

These locking mechanisms are applied to a database's primitive structures [4][5]:

Table Locks

Table Locks are the most basic locking strategy available in MySQL. Client applications can only acquire or release locks for themselves.

Row Locks

Row locks offer a greater degree of concurrency than table locks, as any locks obtained by a client application apply only to a row in a table as opposed to the whole table. Use of Row locks is dependent on the storage engine [6].

## Storage Management & Engines

These are the critical software modules responsible for storing and retrieving the data stored in the database tables. These modules receive such requests from MySQL via a storage engine API. Their pluggable nature is unique to MySQL allowing a user to select a specific engine based on their application needs. The pluggable storage engine architecture provides a certain standard set of services common among all the engines offered by MySQL, allowing a user to select one engine for one table, and another for a different table. This is only made possible due to MySQL's ordering of its subsystems, abstracting implementation detail of the various storage engines away from the client application while simultaneously providing all the benefits [7].

The two most commonly used storage engines are InnoDB and MyISAM. There are several other storage engines provided such as MEMORY, BLACKHOLE, CSV, ARCHIVE,

PERFORMANCE_SCHEMA, FEDERATED, MRG_MYISAM. Each storage engine is designed with different use cases, and therefore has its own unique advantages [8].

## Evolution of MySQL

The first version of MySQL was released in 1996, and only the binary distributions for Solaris and Linux was provided. Over the next two years, MySQL version 3.x was released and was ported to several other operating systems as the operations of this RDBMS increased. Some of the features released in version 3.x included the optimizer, numerous APIs, ISAM storage engine, and a subset of the SQL language. By 2001, MySQL version 4.x had come out with the InnoDB storage engine. Several new features were added for version 5.x, in which native JSON types, Unicode characters, and stored procedures were supported [9].

MySQL 8.0 changed how metadata, which can be thought of as a database of a database, was stored. For example in a banking database, the metadata may include a table containing all tables that involve loans. Previously, such data was stored in a files and non-transactional tables. Now the data dictionary consists solely of transactional tables, making it more in line with how user data is stored[10].

Based on these trends, it is predicted that MySQL will continue to support a greater variety of data types. Its long-term health, however, will largely depend on the evolution of its Pluggable Storage Engine Architecture.

## External Interfaces

External interfaces refer to how information is transferred to and from a system. For MySQL, it would be how users and developers interact with the database; an example being sending queries. The two main ways of doing so are by connectors and through graphical user interfaces.

At a high level, the architecture of MySQL's external interfaces are based on the client-server style. One of the main features of this style is having a server that handles most of the data processing, and having clients that do little data processing. Here, MySQL Server does most of the data processing, while the clients mainly send queries.

MySQL's external interfaces can also be seen in a repository style. The central data structure in this case would be MySQL server, and the components around it would be the clients. A key feature of repository style architecture is concurrency, which MySQL also supports.

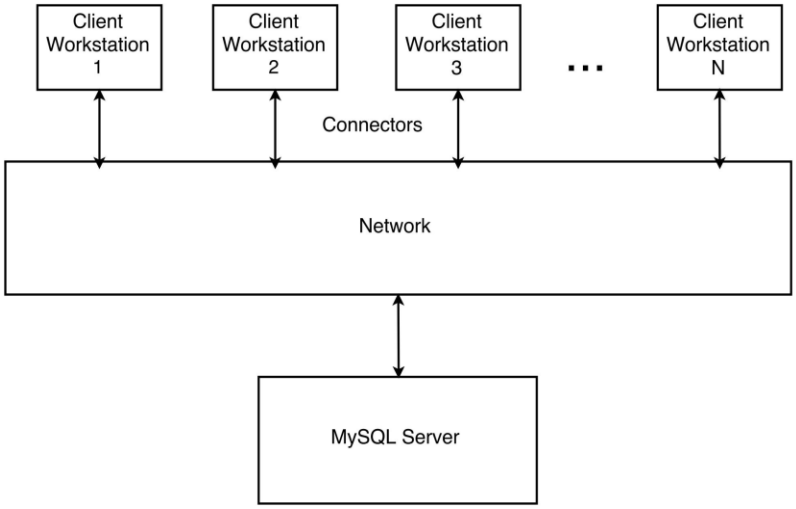


Figure 5 - Architectural diagram of MySQL's External Interfaces

## Connectors and APIs

The most common way that developers use to connect to MySQL Server is through connector APIs. MySQL supports general connectors with APIs to execute MySQL statements for nearly all popular programming languages. This includes C, C++, Java, .NET, and many more. It also supports a shell for other programming languages such as JavaScript, Python, and Visual Studio.

### Connector API

MySQL Server has unique APIs for each programming language. These connectors connect directly to the database system using programmatic interfaces. Most of these database connectors are based on the Open Database Connectivity (ODBC) model.

ODBC is a specification for an API, used to send and retrieve SQL queries from the client to the database [1:28]. The client sends commands to an implementation of a ODBC connector (such as Connector/J, Connector/C, or others depending on the language). This comes with an ODBC API, which acts as an intermediary to communicate to the ODBC library. This library then uses a specialized driver to connect and send the command to MySQL Server. The database retrieves and then interprets the command; processing it and returning the result to the user.

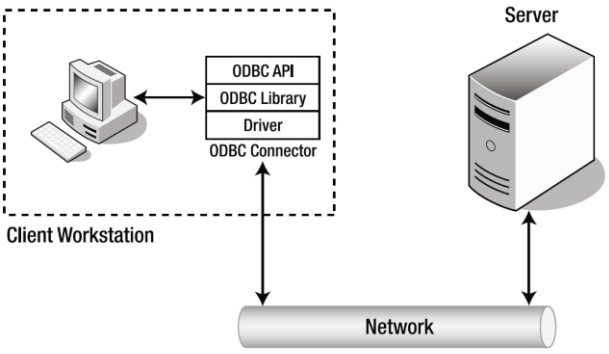


Figure 6 - ODBC Low Level Architecture [1:28]

## Graphical User Interfaces (GUI)

There are graphical user interfaces (GUI) that are developed for MySQL, as an alternative to traditional connectors and API's. These interfaces are useful to view and interact with a user's database graphically, instead of programmatically. It allows the same features as a command line interface, and allows a user to manage their model and administer queries. There is an official integrated environment for MySQL was developed called "MySQL Workbench" [11]. However, there are also many other third-party interfaces are available.

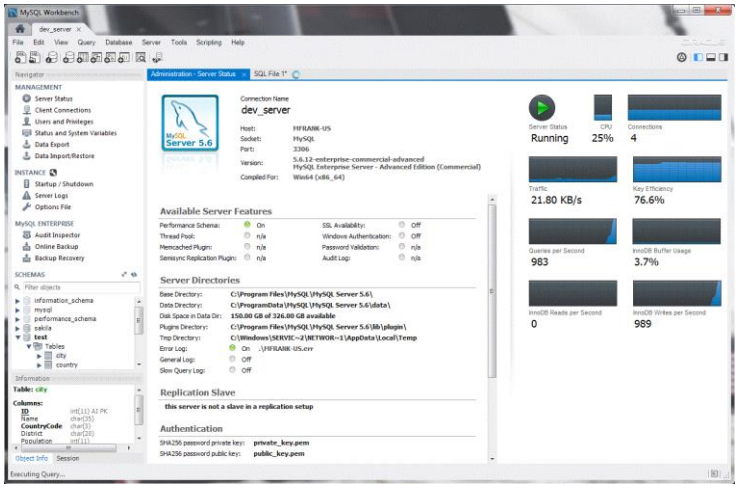


Figure 7 - MySQL Workbench Interface

## Responsibilities amongst Participating Developers

MySQL Server is an open-source project made by Oracle, and is in active development. It is licensed under the GNU General Public License v2.0 [12]. The two main groups contributing to the development of this project are users and contributors.

Users are the community that uses MySQL the most. This includes both non-commercial software developers (MySQL Standard Edition) and commercial software developers (MySQL Enterprise Edition). They mainly provide bug and defects for contributors to fix.

Contributors are a community that provide new features, as well as fixes for bugs and defects. This is comprised of Oracle's MySQL Server development team, as well as any online open-source developer. There are four rules to become a participating open-source developer for MySQL [13].

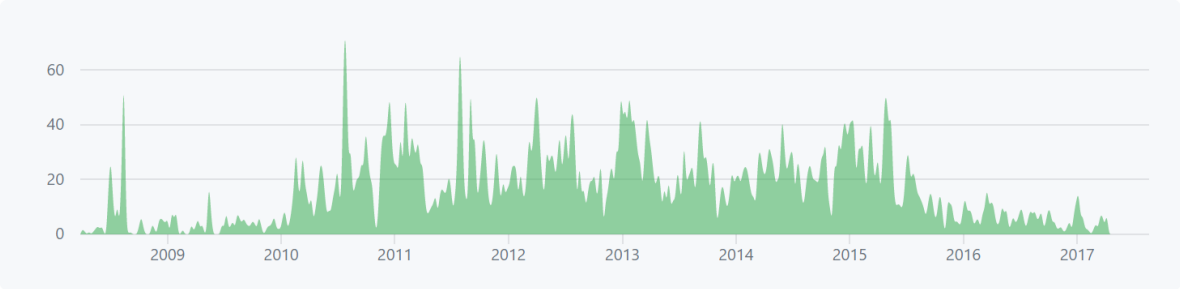


Figure 8 - MySQL Github Commit History [12]

1. Sign and return the Oracle Contributor Agreement (OCA).
2. Follow MySQL Internals Coding Guidelines.
3. Have proper test, architectural, design, and code reviews before submitting a patch.
4. Provide documentation changes if necessary.

The contributor group also includes product managers and release engineers in the MySQL development team at Oracle. Product managers decide which features or defects to work on, while release engineers focus of providing a working release candidate for public use. Both these groups run blogs with posts on highlights of each release.

## Use Cases

Two typical scenarios of a user executing a query in MySQL are examined below. The first scenario shows MySQL's workflow when a user enters the query "SELECT * FROM *table_name*" and the second scenario shows how MySQL responds to the query being executed consecutively.

Use Case 1

The user wishes to see all table rows and columns in 'MyTable', entering the query "SELECT * FROM MyTable":

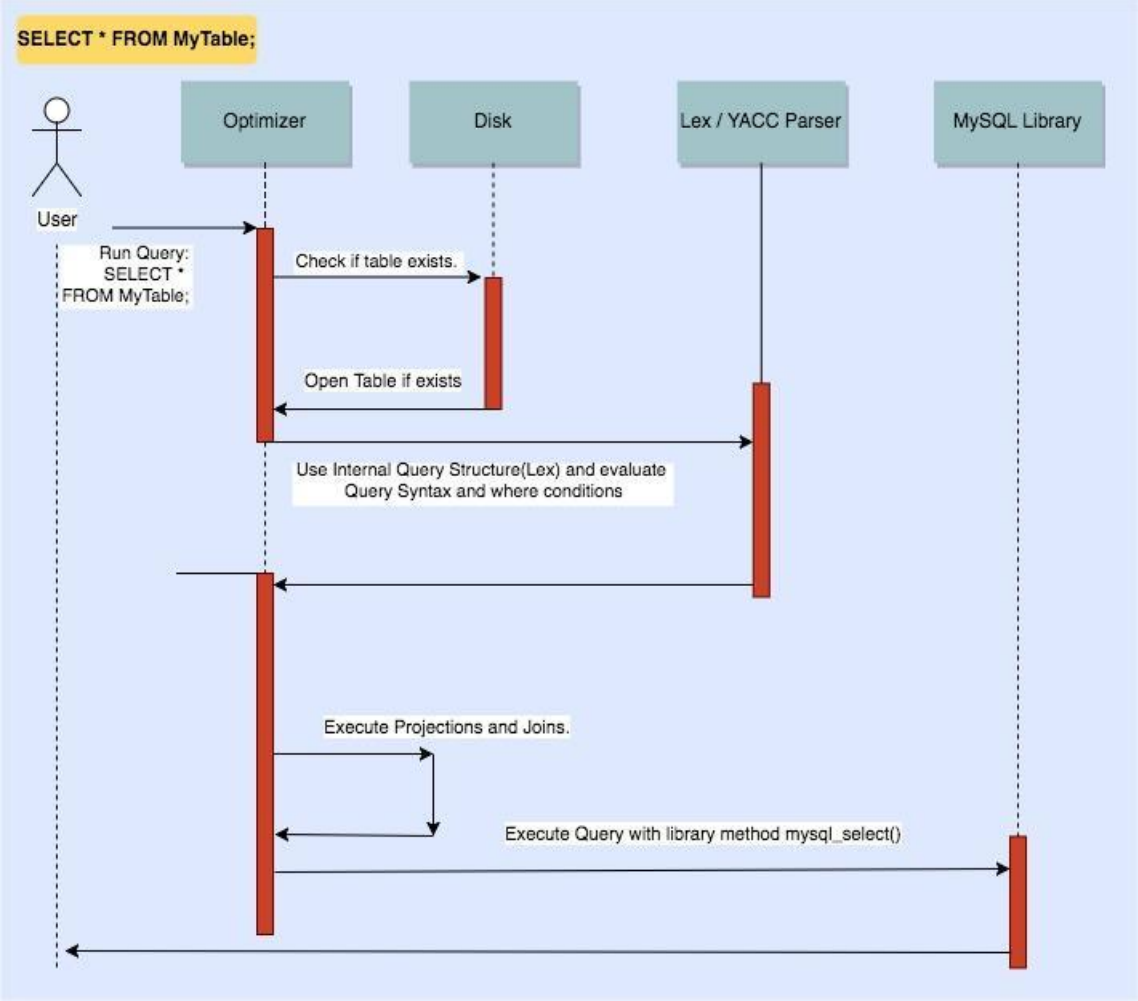


Figure 9 - User selecting all rows & columns from 'MyTable'

Use Case 2

The user wishes to see all table rows and columns in ‘MyTable’, entering the query “SELECT * FROM MyTable”. After some time, the user enters the same query:

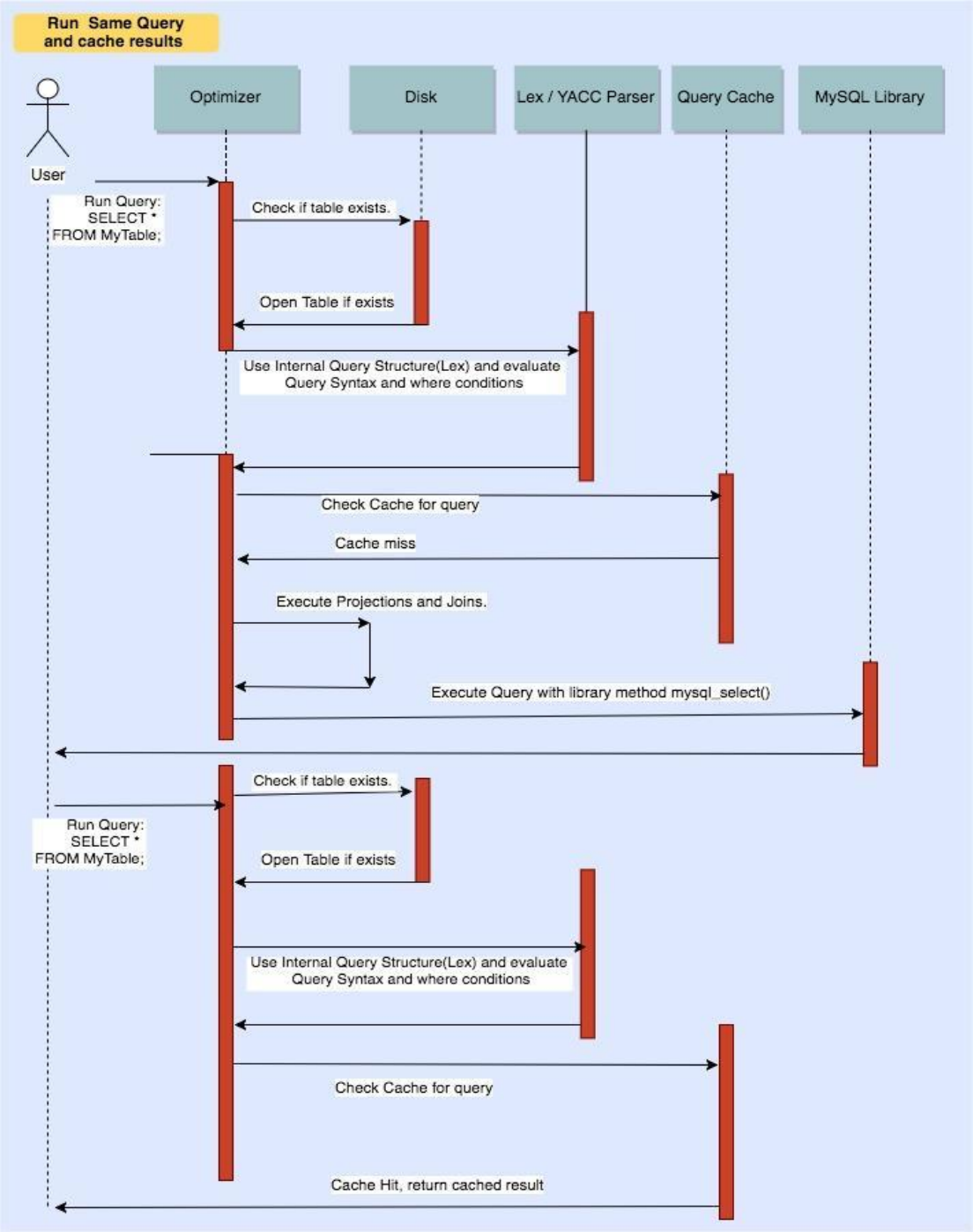


Figure 10 - User running the same command as in Figure 6

## Data Dictionary

Database Management System: A system handling concurrent access to a shared database and creation of a database. Each database consists of tables, which in turn consist of rows (entries) and columns (types)

Relational Database Management System: A DBMS where tables share relationships with each other, e.g. one table is linked to another due to the common field “Student ID”

Query: a unit of work for a database to execute

Transaction: A transaction is a sequence of operations performed as a single logical unit of work.

ACID: A set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, etc.

API: An application program interface (API) is a set of routines, protocols, and tools for building software applications. (Beal "API - application program interface")

Concurrency: Ability of multiple processes to access or change shared data at the same time.

Linear Pipe-And-Filter: A software architecture that consists of any number of components (filters) that transform or filter data, before passing it on via connectors (pipes) to other components in a sequential manner

Repository Architecture: A software design architecture that restricts working directly with the data and creates new layers for database operations, business logic and the application’s UI.

Server: A computer or computer program that manages access to a centralized resource or service in a network.

## Naming Conventions

API: Application Programming Interface

CRUD: Create, Read, Update, Delete

RDBMS: Relational Database Management System

DBMS: Database Management System

SQL: The Structured Query Language

JSON: JavaScript Object Notation

GUI: Graphical User Interface

## Conclusions

Developing large-scale software systems is no easy task. To make it more feasible and manageable, such a system must be broken into smaller, more manageable subsystems and modules. As seen in the document, this leads to each subsystem and module implementing a variety of architectural styles to achieve their respective goals. Hence, large scale systems like MySQL cannot be described as following architecture A or B. The opposite, in fact, holds true: MySQL, and other large pieces of software, are comprised of a mix of architectural styles.

## Lessons Learned

**Kevin Arindaeng:** Contributing to MySQL as an open-source developer requires numerous of steps, such as having proper test, architectural, design, and code reviews, resulting in high software quality.

**Anton Sitkovets:** MySQL is unique because it caches the query structure as well the query result, making it more time-efficient than the competition.

**Glib Sitiugin:** Concurrency locks do not necessarily have to be applied to the entire set of data.

**Nisha Sharma:** It is possible to mix architectural styles together.

**Varsha Ragavendran:** MySQL's pluggable storage engine architecture provides benefits by allowing a client to load engines based on the application's requirement, thus working towards a client's advantage.

**Ayman Abualsunun:** Systems can be made with more than one programming language.

**Davood Anbarnam:** Now I understand the true power of documentation.

# References


[1] C. Bell, *Expert MySQL*. Berkeley, CA: Apress, 2012

[2] "MySQL :: MySQL 5.5 Reference Manual :: 8.10 Buffering and Caching", *Dev.mysql.com*, 2017. [Online]. Available: https://dev.mysql.com/doc/refman/5.5/en/buffering-caching.html. [Accessed: 18- Oct- 2017].

[3] "ACID", *En.wikipedia.org*, 2017. [Online]. Available: https://en.wikipedia.org/wiki/ACID. [Accessed: 18- Oct- 2017].

[4] Schwartz, Baron, et al. High performance MySQL. Beijing, OReilly, 2012

[5] "MySQL :: MySQL 5.5 Reference Manual :: 14.8.1 InnoDB Locking", *Dev.mysql.com*, 2017. [Online]. Available: https://dev.mysql.com/doc/refman/5.5/en/innodb-locking.html. [Accessed: 18- Oct- 2017].

[6] "A few notes on locking in MySQL - ovais.tariq", *ovais.tariq*, 2017. [Online]. Available: http://www.ovaistariq.net/612/a-few-notes-on-locking-in-mysql/#.WdPiOWhSyM8. [Accessed: 18- Oct- 2017].

[7] "MySQL :: MySQL 5.5 Reference Manual :: 15.11 Overview of MySQL Storage Engine Architecture", *Dev.mysql.com*, 2017. [Online]. Available: https://dev.mysql.com/doc/refman/5.7/en/pluggable-storage-overview.html [Accessed: 18- Oct- 2017].

[8] "MyISAM vs InnoDB", *d4nyll*, 2014. [Online]. Available: http://blog.danyll.com/myisam-vs-innodb/. [Accessed: 18- Oct- 2017].

[9] S. Pachev, *Understanding MySQL Internals*. Sebastopol: O'Reilly Media, Inc., 2009.

[10] "MySQL :: MySQL 8.0 Reference Manual :: Chapter 14 MySQL Data Dictionary", *Dev.mysql.com*, 2017. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/data-dictionary.html [Accessed: 18- Oct- 2017].

[11] "MySQL :: MySQL Workbench", *Mysql.com*, 2017. [Online]. Available: https://www.mysql.com/products/workbench/. [Accessed: 17- Oct- 2017].

[12] "mysql/mysql-server", *GitHub*, 2017. [Online]. Available: https://github.com/mysql/mysql-server. [Accessed: 17- Oct- 2017].

[13] "Contributing Code to MySQL | Oracle Community", *Community.oracle.com*, 2017. [Online]. Available: https://community.oracle.com/docs/DOC-914911. [Accessed: 17- Oct- 2017].