# Final Project Report

## Exploring MySQL and Redis

Members: D.J. Bucciero, Annie Tao

Class: Advanced Computer Systems

## Introduction

The objective of this project is to explore and analyze MySQL and Redis, two infrastructure-level softwares. This entails learning how to use the software, configuring the software, and using the workbench and analysis software that is often included with infrastructure-level software.

The device being used for analysis has an Intel Core i7-8705G with a clock rate of 3.1 GHz. It has 8 GB of DDR4 ram installed with a clock rate of 2400 MHz. In order to fully test the two software that were selected, there will be performance analysis of two databases which will be elaborated on further within each software's section.

In order to maintain consistency, the first database tested will be a database that is used primarily for teaching purposes often provided by software manufacturers. For each software this will be referred to as the "Primary Sample Database." This will be used as a tool to show and learn how both software are capable of analyzing the performance of a database and its server.

The second database tested for each software will still have a relatively simple structure but will vary in its number of entries. For simplicity purposes this will be referred to as the "Secondary Sample Database." This will allow tests to show how a differing workload will affect performance and to compare database optimization. Using this new workload and two hardware environments will allow for analysis to demonstrate the software's capability and performance. Testing for both software will also include a basic review of graphical user interface and our assessment of the accessibility of the software in conjunction with its performance.

## What is an infrastructure-level software?

An infrastructure-level software in the context of MySQL and Redis is a database management service. MySQL is used to deploy cloud-native applications while Redis is an in-memory data structure storage system. MySQL is intended to be an accessible approach to SQL which can be very daunting at first glance. Redis is designed to pioneer its own way and refers to itself as the "Most loved NoSQL database in the world," Essentially they have different approaches to the same problems and challenges. MySQL and Redis are both very popularly used by many small and large scale industrial applications.

MySQL focuses on being accessible and capitalizes on its scalability and complexity by advertises the large companies that use it such as Netflix, Uber, Amazon,  Slack, etc. It pursues a clean user interface which is important for a fixed data schema. It is most known for its ability to handle complex lookups and heavy transactions. Redis focuses on its simplicity and efficiency rather than its cleanliness and complexity. It is known to be much more efficient for simple and quick lookups regardless of scale, it is also known to be much better if your application uses non-structured data. It is also worth noting that Redis is used to support Github, and is also recognized for having taken over support of Twitter and Pinterest which were formerly supported by MySQL. The community appears to be shifting in favor of Redis due to its ability to scale, while MySQL appears to be slowly losing control of the market due to its inability to complete many simple operations.

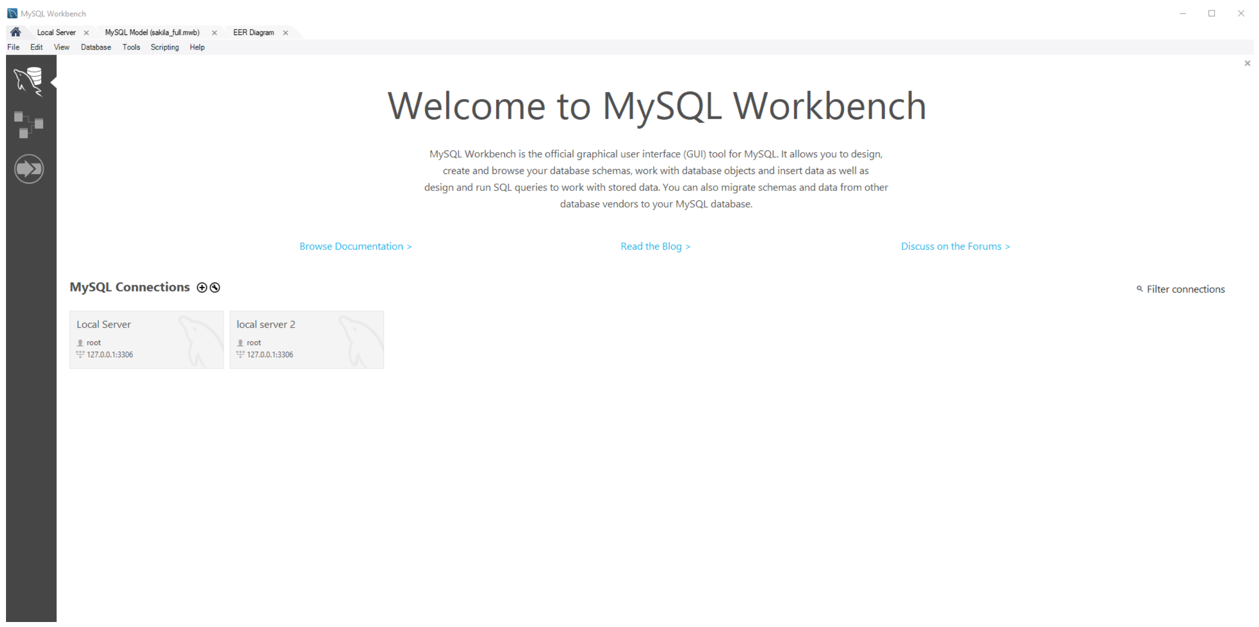
From the research done on community reception of both software, keeping data as objective as possible, the expectation is that both will be efficient and effective, but will offer different approaches with their own flaws and challenges. We expect MySQL to be accessible and well structured excelling in complex operations. We expect Redis to focus on its simplicity and to work faster in specific cases with non-structured data and simple operations. Redis should also scale up more effectively in comparison to MySQL if the database is properly set up.

## MySQL

MySQL is an open-source relational database management system. The data is organized into tables that are labeled by the users and relationships are set between them. MySQL works with an operating system to implement the database in a computer's system, manages users, allows for network access, and allows for testing database integrity. This software is widely used by many popular websites such as Facebook and Youtube.

### Setup Process and GUI

The setup for MySQL is relatively straightforward. It is an open source software with an installer available and provided by Oracle. In that installation process there are options to select or deselect various features. The most important feature besides MySQL itself is MySQL Workbench. For all intent and purposes the MySQL Workbench is your command center for all MySQL servers. It features a clean cut interface and there is also plenty of tutorials and documentation for setup of both the software and servers readily available online.



Welcome to MySQL Workbench

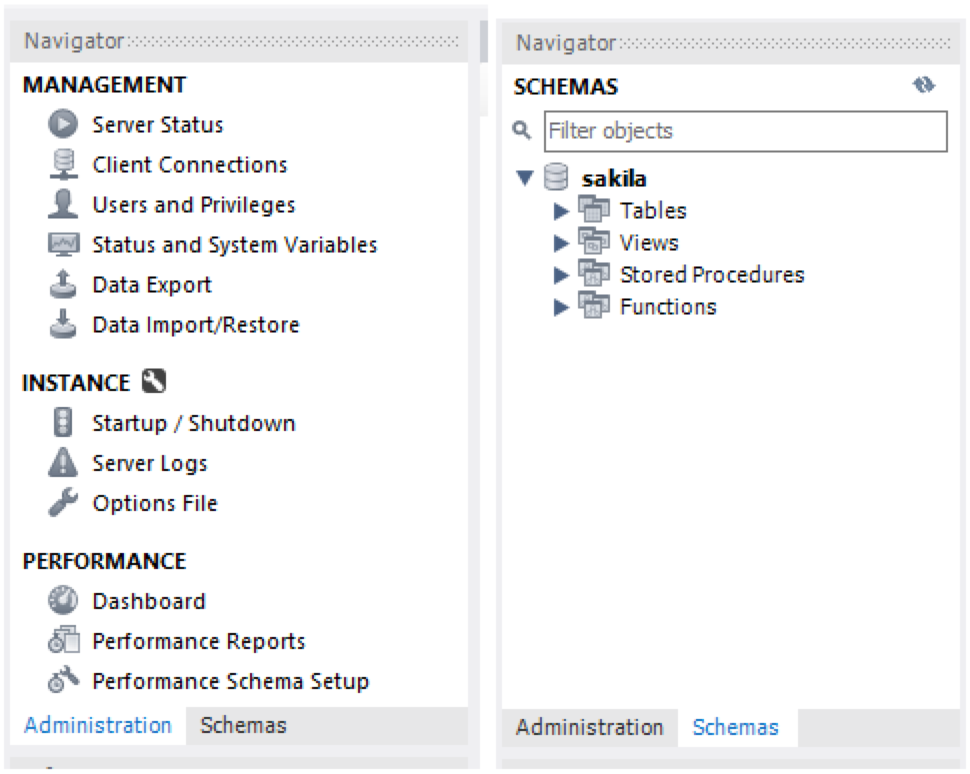
MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

Local Server
root
127.0.0.1:3306

local server 2
root
127.0.0.1:3306

From the Workbench software you can select servers and view which schema are being used. This will be any combination of databases selected, and the "sys" schema which is used for performance metrics and general configuration of the server. Once a server is opened and schema/administration are displayed on the left side of the screen, the right side of the screen will be composed of various tabs. These tabs can have scripts to run on your servers, they can have performance metrics for your servers, and they can have diagrams of your database schema.

In order to configure or open tabs you must select “administration” or “schemas” then select what you would like to view on the left side of your GUI.



From the administration tab, there are a multitude of tools at your disposal for both database and server analysis, and the schema tab features a view of your database structure(s). It is also worth noting that almost all data is exportable and importable, including performance reports, status reports, and your database itself.

## Performance Reports and Analysis

Inside the MySQL workbench software, there is a section under administration referred to as "Performance Reports." This section is used for database optimization and analysis of your database in order to determine where there may be inefficiency. There are 7 categories of reports with two to five subcategories of metrics in each used in order to analyze your database and server usage.

The categories are as follows, Memory Usage, Hot Spots for I/O, High Cost SQL Statements, Database Schema Statistics, Wait Event Times, InnoDB Statistics and User Resource Use. Each of these reports has subsections used to analyze use, cost, resources, etc. For example, the "Memory Usage" section has total memory, memory by event, memory by user, memory by host and memory by thread. Essentially all performance metrics are categorized in an accessible GUI with the capability to export all data collected by the MySQL Workbench software.

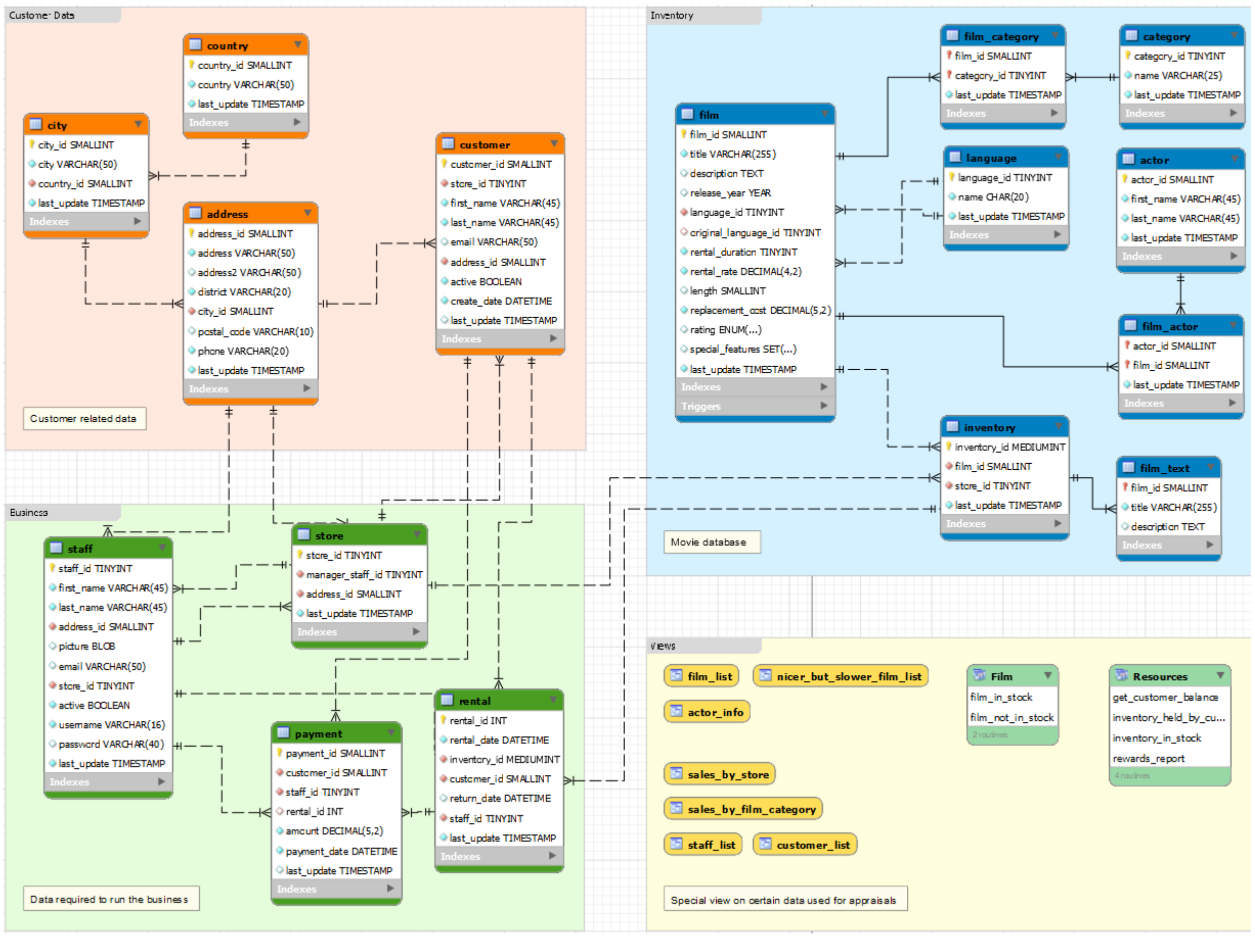
Memory usage as an analysis tool is relatively self evident and obvious, however hot spots and high cost statements are important tools for database optimization. By viewing the data presented by the performance reports, one can understand where unnecessary accesses are being made. This comes in the form of identifying redundancies, inefficient structure, bottlenecking/overworking areas of the database, as well as operations that are not optimized. These tools for optimization are important to use because the optimization of a database forms the foundation for an efficient software, website, etc.

For the purposes of this report, the "Primary Sample Database" section will show how these metrics can be viewed and used while the "Sample Database" section will show how workload can affect these metrics. Ultimately the culmination of these sections will be to show how achieving a path towards further optimization is facilitated by the MySQL Workbench software.

## Primary Sample Database - Sakila

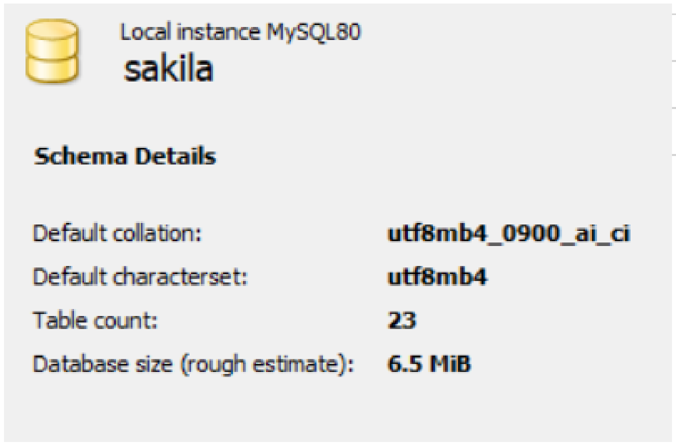
The primary sample Database used to test MySQL is the "Sakila" database, a normalized schema which models a DVD rental store. It happens to be one of the most popular sample databases and was created by MySQL to be used in teaching demonstrations of their software. This makes it a perfect example for testing and initial exploration of the software.

The Sakila database is an example of a well-optimized but also complex enough that it can actually test optimization. It includes information about stores, customers, inventory, as well as in depth information about films, their actors, categories, etc. It is a prime example of a logical schema that is intuitive to understand and learn from. If someone were to ask "What is important about a film?" a mental process of a human would follow the path and process of the Sakila database.



The schema of a database is the basic structure of a database, essentially a “blueprint” of the database’s construction and formatting. Schema can be presented in multiple ways, however the MySQL workbench chooses to offer a visual representation that appears similar to a flowchart. On the “programming side” of development a schema is essentially a set of formulas and integrity constraints. The graphical representation of both the schema and relevant analysis data is far more intuitive from a human perspective and makes sense as a design choice for MySQL. Given that the sample database Sakila uses a tangible but theoretical real life example of a DVD rental service, it proves simple and accessible at an entry level.This is a major strength of both MySQL and its provided sample databases.

**Primary Sample Database Database Data**


Local instance MySQL80
sakila

Schema Details

| | |
|---|---|
| Default collation: | utf8mb4_0900_ai_ci |
| Default characterset: | utf8mb4 |
| Table count: | 23 |
| Database size (rough estimate): | 6.5 MiB |

| Table | Size (MB) |
|---|---|
| Rental | 3 |
| Payment | 2 |

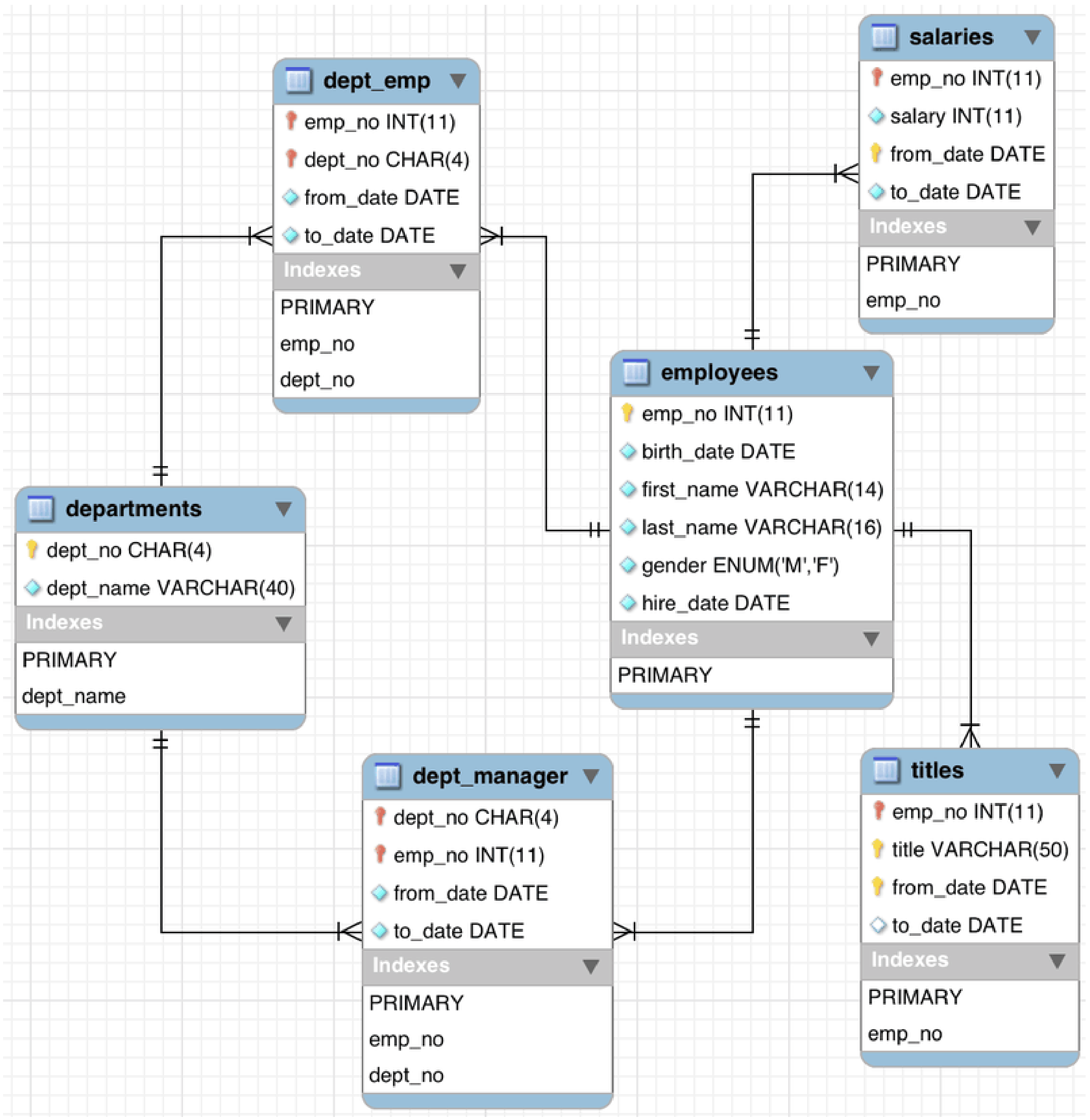
*Schema Inspector and Size of Each Table (not listed tables were listed as 0 MB)*

The first database was an example provided by mySQL during installation. The data is very small, estimated to be around 6.5 MiB. This sample is used for teaching and practice of using mySQL. A query was run for each table where 10 random rows of data were fetched. The runtime is shown in the table below. Since the tables are so small, there is little significant difference in the runtimes between tables. Some tables that are not listed as 0 seconds on the table, ranged from 0.0 to the listed seconds after multiple executions of the data fetch.

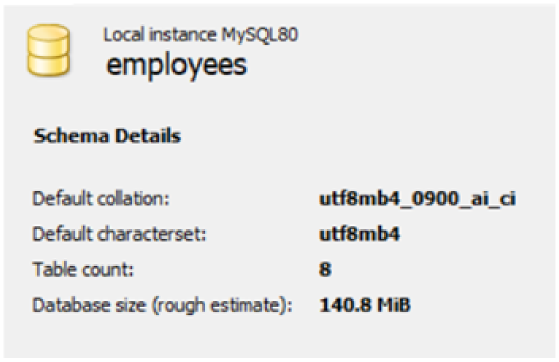
| Table | Seconds |
|---|---|
| Actor | 0.015 |
| Address | 0.031 |
| Category | 0 |
| City | 0.016 |
| Country | 0.015 |
| Customers | 0.032 |
| Film | 0 |
| Film Actor | 0.015 |
| Film Category | 0 |
| Film Text | 0.016 |
| Inventory | 0.016 |
| Language | 0.016 |
| Payment | 0.047 |
| Rental | 0.047 |
| Staff | 0.016 |
| Store | 0 |

## Secondary Sample Database Introduction

The secondary sample Database used to test MySQL is the “employees” database, a normalized schema which models a theoretical business with employees, departments, titles, etc. Similarly to Sakile, the employees database is another example created by MySQL for the purposes of testing the software and teaching the use of the software. The employees database is slightly outdated, as the sakila database used as the primary example was created as a replacement for the Employees database. As a result we expect the sakila database to be slightly more optimized and to perform slightly better when looking at the performance metrics.



## Secondary Sample Database Data


Local instance MySQL80
employees

Schema Details

|  |  |
|---|---|
| Default collation: | utf8mb4_0900_ai_ci |
| Default characterset: | utf8mb4 |
| Table count: | 8 |
| Database size (rough estimate): | 140.8 MiB |

| Table | Size (MB) |
|---|---|
| Salaries | 90 |
| Titles | 20 |
| Dept_Emp | 17 |
| Employees | 15 |

*Schema Inspector and Size of Each Table (not listed tables were listed as 0 MB)*

The second database was an example provided from https://github.com/datacharmer/test_db. This database contains information about 300,000 employee records with 2.8 million salary entries. When the data is exported, it is around 167 MB. The data was provided with .sql files that generated the schema and inputted the data into the schema and tables. The total time to create the schema was 72.725 seconds.

| Table | Seconds |
|---|---|
| Departments | 0.008 |
| Dept_Emp | 0.203 |
| Dept_manager | 0.007 |
| Employees | 0.308 |
| Salaries | 1.581 |
| Titles | 0.344 |

*Runtime of fetching 10 rows per table*

A query was run for each table where 10 random rows of data were fetched. The runtime is shown in the table above. While this database is not huge, it is significantly bigger than the first sample to where there are significant differences in query as the tables get bigger. Below is a table of several queries being performed on the salary table. The query increasingly fetches more rows of data (power of 10) until the maximum number of available rows are returned. The duration of this operation increases with the size of the requested information and starting from 100,000 rows, the fetch time is large enough to not be rounded to 0 seconds by the program.

| Number of Returns | Duration/Fetch |
|---|---|
| 1 row(s) returned | 1.485 sec / 0.000 sec |
| 10 row(s) returned | 1.469 sec / 0.000 sec |
| 100 row(s) returned | 1.500 sec / 0.000 sec |
| 1000 row(s) returned | 1.610 sec / 0.000 sec |
| 10000 row(s) returned | 1.844 sec / 0.000 sec |
| 100000 row(s) returned | 2.156 sec / 0.062 sec |
| 1000000 row(s) returned | 2.515 sec / 0.579 sec |
| 2844047 row(s) returned | 2.781 sec / 1.672 sec |

## MySQL Conclusion, Results and Review

Overall the experience with MySQL was relatively seamless. It provided very useful onboard measurements and performance analysis for hosting a server with a mostly intuitive GUI. Due to the accessible GUI with the integration for visual representation of performance metrics and schema, it was easier to understand exactly how the server was running. A valuable lesson is to be learned from database analysis, and primarily it is a lesson of relativity. The magnitude is not of concern unless you build inefficiently. When using a structure that follows proper design standards similarly to Sakila or the Employees database your results will be positive regardless of how large the project will scale. If you build your database with inefficiencies, redundancies, high cost statements, etc. there will be more and more issues down the line.

While Sakila performed slightly better relative to size than Employees it was obvious that both databases were highly optimized and were extremely efficient. As a result this project was more of a learning experience of how to use the software to manage and analyze a server, and what to look out for in terms of performance anomalies and issues. When creating an actual database to be used for hosting, it will be apparent which design choices to make based on the results and example provided by the MySQL team at Oracle.

There are many pitfalls to be aware of when designing a SQL database. These issues are more obvious than just memory, cpu, or network usage. It is important to only select data that is needed and avoid extraneous data selection. Tables should be efficiently joined to mitigate unnecessary accesses. Another tricky issue that can occur when designing an SQL database is having too much literal SQL, this means having literal values in operations too frequently. Because literal values cannot be shared and stored, they must be parsed every time. There are many similar examples to overuse of literal SQL so the main point is to understand which operations are expensive and to try and avoid them at all costs. In addition, the database should be designed in a way that there will not be conflicts between different users when accessing the database. Due to exclusively local testing we were unable to demonstrate this issue, but once a server is rolled out to be non-local the multitude of user queries can be problematic if they interfere with each other due to calculations or other unforeseen issues.

## Redis

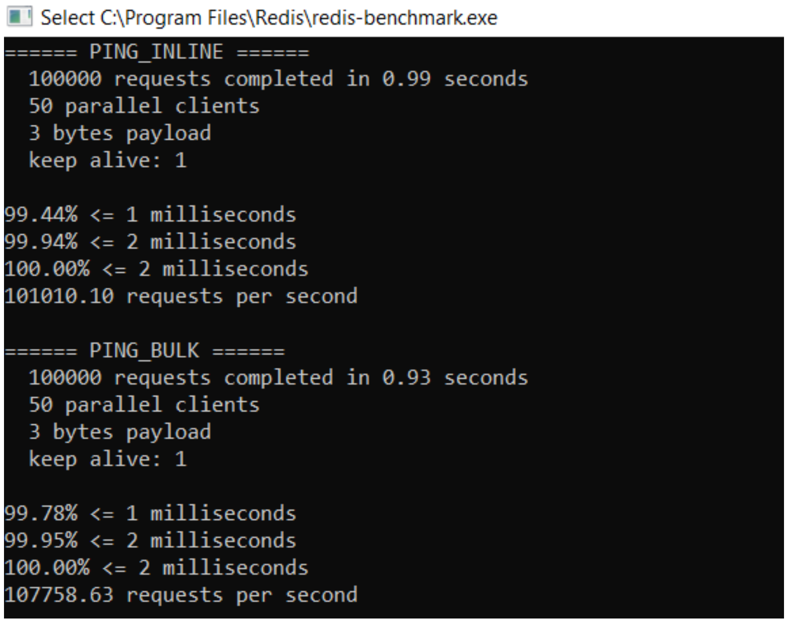
Contrary to the relational DBMS approach of MySQL with a fixed data schema, Redis uses Key-Value storage and is completely data schema free. While this can be criticized as less visually appealing and less easy to understand at first glance, it proves to be highly scalable with proper execution. As a result, many major companies rely on Redis to support their platforms. As Redis continues to develop and improve, many companies have switched from MySQL to Redis for its scalability features.

### Setup Process and GUI

Redis is also an open source software. Documentation is provided on the Redis website as well as tutorials which are provided on Github and YouTube. The installation process is much more complex through Windows because it is not officially supported, so if you are on a Windows device it is best to either use Ubuntu or a different device with Linux. There is a user interface called Retool or RedisGUI that allows for more configuration and visual representation of commands and databases. It is not on the same level of depth as MySQL Workbench, but nonetheless provided tools necessary for analysis and setup of a database. Due to its lack of support on Windows we found it more intuitive to use “Redis Insight” a website that can be used to connect to databases to perform analysis. Not only was it easier to set up, it was also easier to gather metrics and understand those metrics effectively.

Overall non-relational databases are non-visual in comparison to schema based relational databases, so it is quite unfair to fault Redis for its ability to show you what is actually going on. For non-relational databases, they do offer more tools than other peers that host non-relational databases and do the best they can with their resources. As a whole the limitations on the interface side of development can be mitigated by the performance benefit in larger scale projects.

## Performance Reports and Analysis



Select C:\Program Files\Redis\redis-benchmark.exe

```
====== PING_INLINE ======
  100000 requests completed in 0.99 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.44% <= 1 milliseconds
99.94% <= 2 milliseconds
100.00% <= 2 milliseconds
101010.10 requests per second

====== PING_BULK ======
  100000 requests completed in 0.93 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.78% <= 1 milliseconds
99.95% <= 2 milliseconds
100.00% <= 2 milliseconds
107758.63 requests per second
```

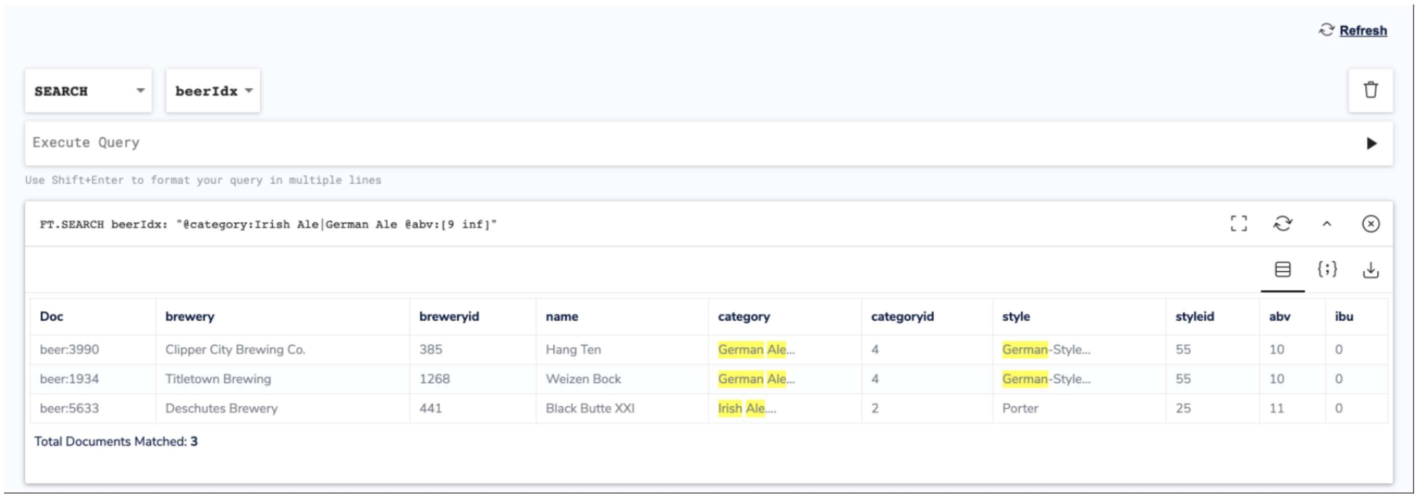
The Windows version of Redis that was used had an executable file called “redis-benchmark” seen above. This program simulates running commands done by X amount of clients at the same time sending Y amount of total queries. This gives the user a general idea of how many requests the server can handle as well as allow them to identify any bottlenecks.

## Primary Sample Database Introduction

The primary sample database used to test Redis is called “Retwis.” By Redis developers this is referred to as their “Hello World.” It is a Twitter-style social network clone, written using Redis and PHP. It is quite controversial in the context of infrastructure level software, as it was used as a proof of concept for the scalability of Redis and was the start of Twitter’s migration to their platform. Essentially Redis proved itself to the industry by showing that their software was capable of supporting even the largest scale platforms highly efficiently. As a result of their success and efficiency, Twitter switched over to Redis a year after Retwis’ release. To give a timeline, Twitter was established in 2006 and depended on MySQL, then Redis was developed and established in 2009, with the transfer occurring in 2010. The transfer was gradual and initially only included the timeline and ads services. As a result, it is fitting to use Retwis as an example to test Redis, as it was used to proclaim Redis’ success to the world.

## Secondary Sample Database Introduction

The secondary sample database used to test redis is called “Openbeerdb.” This database is featured in the “Redisearch” section of the redis dataset repository. The main purpose of this database is intended to connect people to their favorite beers. This database is larger than Retwis which is more of a proof of concept, but isn’t incredibly large. It appears that Redis markets itself more as having a multitude of different tools, such as search, graph, etc.  rather than being all inclusive. By testing this database we came to understand one of the many features which Redis can be used to support.



```
FT.SEARCH beerIdx: "@category:Irish Ale|German Ale @abv:[9 inf]"
```

| Doc | brewery | breweryid | name | category | categoryid | style | styleid | abv | ibu |
|---|---|---|---|---|---|---|---|---|---|
| beer:3990 | Clipper City Brewing Co. | 385 | Hang Ten | German Ale... | 4 | German-Style... | 55 | 10 | 0 |
| beer:1934 | Titletown Brewing | 1268 | Weizen Bock | German Ale... | 4 | German-Style... | 55 | 10 | 0 |
| beer:5633 | Deschutes Brewery | 441 | Black Butte XXI | Irish Ale.... | 2 | Porter | 25 | 11 | 0 |

Total Documents Matched: 3

## Redis Conclusion, Results and Review

Overall our experience with Redis was positive. There were some difficulties with setup and integrating the Redis GUI. Redis as a whole offered slightly less tools for analysis onboard their GUI and relied more on user commands which could be tricky to figure out at first glance. Fortunately there was ample documentation to help us determine the capabilities of the software, and many sample databases to choose from in order to understand Redis' features.

Retwis and OpenbeerDB both performed exceptionally well even when compared to the similarly sized databases tested in MySQL. It was obvious that both databases were highly optimized and the server hosting software, Redis, was also highly optimized. The idea of a non-relational database can be daunting or overwhelming as it was for our first experiences, but was ultimately rewarding in the end. The project was a learning experience for how to manage servers and analyze their performance, as a result there were growing pains especially with a less intuitive software, but ultimately a comparable end result.

There are difficulties associated with the design of a non-relational database. This became more obvious once actually experiencing the use of the software. There were less debugging tools present to facilitate the issues which could come in even greater magnitude than with a relational database. Ultimately we feel that because of this collaboration could be difficult and that the complexity and inconsistency of a non-relational database could make the process suboptimal for smaller projects. As a result we feel that it would be worth undertaking the responsibility to integrate Redis for certain aspects of a platform if the performance increase would be substantial. It would take more work to integrate but would result in a better end product.

## Conclusion:

Relational databases look nice, make sense, and are in theory fast and organized, but if properly designed they are outscaled by non relational “NoSQL” databases. It is a more difficult task to design a NoSQL, non-relational, database using Redis, but if done properly it simply scales better. This is why many triple A companies have shifted towards this approach as they have the resources to design a more complex database in order to achieve these efficient results.

Both MySQL and Redis are great platforms and fit most projects. Certain projects that are particularly demanding may benefit from using Redis as a platform, but there will be certain design challenges associated with making that switch. A relational SQL database, such as the ones used by MySQL can be very intuitive and easy to understand especially for inexperienced programmers. A visual representation of a database can be very thematic and conceptually easier to understand due to how tangible it can be. Unfortunately this idea of a relational database can become a limiting reagent in the realm of optimization as the project scales. As a result Redis can become a more applicable solution if you can handle non-relational databases and the associated challenges.

In MySQL you should be aware of certain pitfalls that may occur when designing your platform. Selecting more data than needed can be a major issue causing issues with optimization. Inefficient joins between tables can be a difficulty, as overuse of references is an extra workload for the system. Because literal issues cannot be cached or stored, too many literal statements in your SQL can greatly decrease efficiency, a limitation only present if you’re using SQL. In addition, conflicts between user queries can also be an issue when rolling out a platform. While many of these issues sound confusing or daunting, many of the issues can be resolved through the onboard performance and analysis sections of the MySQL Workbench. Consistency and efficiency is something that is sought after by every platform, MySQL does a great job of offering the tools, visuals, and analysis necessary to achieve that goal.

One of the major pitfalls in designing with Redis or other non-relational design is how complex it can be at first glance. It is certainly less intuitive without proper training and exposure and can be intimidating or overwhelming when first attempting to create something new on a completely new platform. The consistency of joins, similarly to MySQL can also be a design issue. Conceptually due to the lack of structure, it can be harder to plan or prepare ideas for your software, which is a major limiting factor for simpler projects that pursue non-relational design. In addition, collaboration can be clunky and counterintuitive at times which is ironic given that Redis boasts its ability to scale properly. It becomes sort of a paradox in that it functions better for larger platforms, but in many ways is just as hard or even harder to design. The main takeaway or conclusion that we could draw from our research, is that it is worth pursuing a non-relational database structure for larger projects, but be prepared for potential design difficulties as they could get very complex very quickly. Both solutions fit an overwhelming majority of platforms, certain niche cases would favor one platform over another based on size, platform and other database metrics.