Assignment 2

Advanced database and big data

Advanced Database and Big Data

U14440

Kieran Best

Reem Khider

Table of Contents

[Introduction 1](#_Toc134745478)

[Implementation (SQL Databases) 1](#_Toc134745479)

[Data Preparation 1](#_Toc134745480)

[Technique 2](#_Toc134745481)

[SQL Queries 3](#_Toc134745482)

[MongoDB Database 6](#_Toc134745483)

[Data Preparation 6](#_Toc134745484)

[Technique 7](#_Toc134745485)

[MongoDB Queries 8](#_Toc134745486)

[Results 9](#_Toc134745487)

[Conclusion 10](#_Toc134745488)

[Splitting of Tasks 10](#_Toc134745489)

[Links 10](#_Toc134745490)

# Introduction

This report aims to describe the process of creating a PHP application that utilizes both MySQL and MongoDB databases. The application was designed to manage private information for a company, including employee job details, salaries, and department locations. Both databases were utilized to store and retrieve data, and the PHP application provided a user-friendly interface for inputting and displaying the data.

# Implementation (SQL Databases)

### Data Preparation

To prepare the dataset given for SQL usage, the first step was to analyse and normalise the data to be able to simplify the accessibility of data. It was decided to separate the job attribute to a table and perform a 1NF and create a relational model by linking between tables using primary and foreign keys to reference them. The same applied to the address and the car table. Our initial normalised logical data model can be seen in Figure 1.

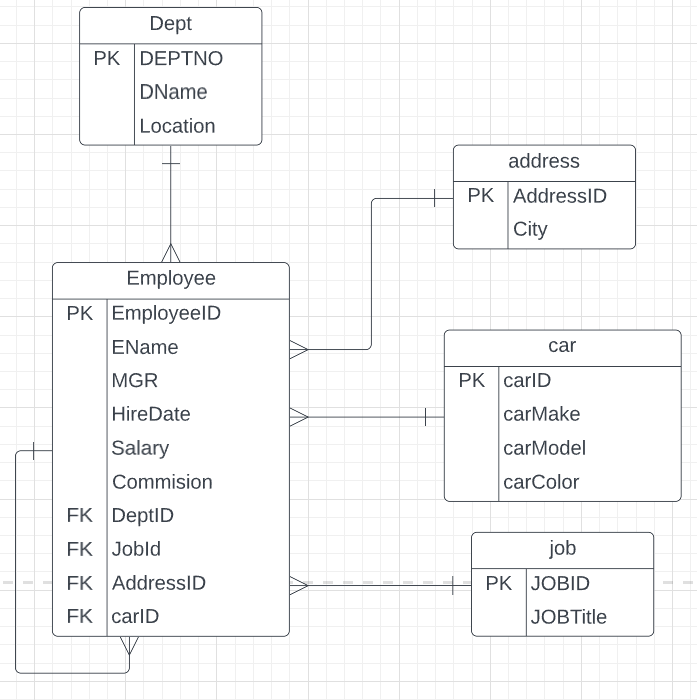


Figure 1: Normalised LDM

We decided that all connections from the “Employee” collection should be ‘many to 1’ as many employees may hold the same qualities as other employees and these should be referenced using foreign keys to save memory if the dataset was to become much larger.

* Many employees may work in the same department
* Many employees may live in the same city
* Many employees may have the same company car
* Many employees may have the same job

From there creating the database in MySQL and inserting data, the creation and insertion script can be found here:

<https://github.com/Reem-313/Database_employee/blob/main/emloyeedb.sql>

### Technique

Following the normalization of data, we proceeded to create the front-end of the application using PHP to connect to the database and enable user to add data to the database by taking their input. This was done by firstly establishing a connection to the MySQL database (Figure 2), then creating a form that handles user input and pass it to the PHP script to perform the insertion query to add the desired data to the connected database (Figure 3 & 4).



Figure 2: Connection TO SQL Database

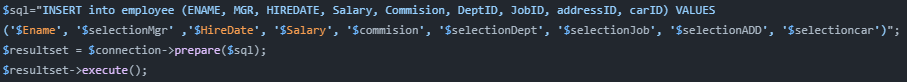


Figure 3: Insertion of Employee into SQL Database

***A screenshot of a computer

Description automatically generated with medium confidence***

Figure 4: Form to Insert Employee

A screenshot of a computer

Description automatically generated with medium confidence

Figure 5: Form to Insert Department

From here this developed to allow the user to insert an employee details with or without a commission or company car and the ability to add to the department table. We then created a table to display all entries from the employee and department table, that is refreshed once a new entry is inputted by the user.

### SQL Queries

The following queries have been created to match and display attributes to display specific information about the dataset.

#### Query 1

This query displays the sum of all the salaries for all and each department.

A screen shot of a computer screen

Description automatically generated with low confidence

Figure 6: Input for Query 1

A screenshot of a chat

Description automatically generated with medium confidence

Figure 7: output for query 1

#### Query 2

This query captures how many people work in each department.

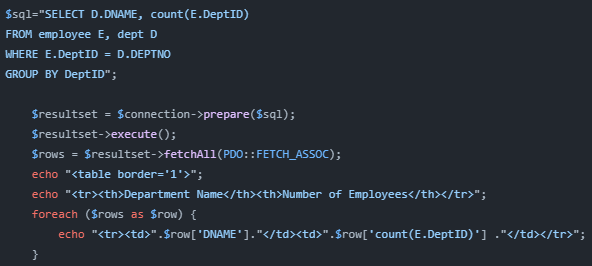


Figure 8: Input for Query 2

***A screenshot of a computer

Description automatically generated with medium confidence***

#### Query 3

This query captures which employees have a salary above 1000, a commission above 0 and have a company car. It outputs their name, salary, commission, and car ID.

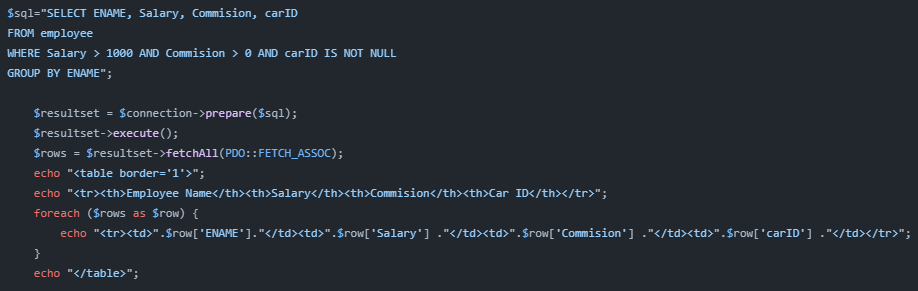


Figure 9: Input for Query 3

A screenshot of a computer

Description automatically generated with medium confidence

Figure 10: output for query 3

## MongoDB Database

### Data Preparation

The format for storing data in MongoDB is in JSON format. To ensure the data was appropriately structured for MongoDB, we denormalized the dataset and created a logical data model that includes cardinalities. This was done while keeping in mind that data in MongoDB is usually outputted in a single format. Figure 11 displays the resulting logical data model.

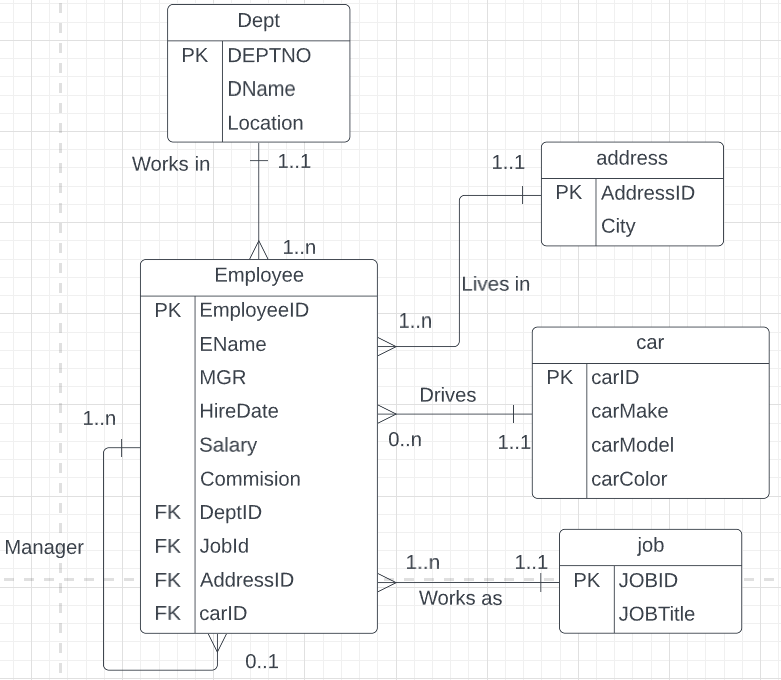


Figure 11: Logical Data Model with Cardinalities

The reason we included cardinalities into this diagram is to be able to fully understand the relations and make a correct modelling decision. From Figure 11 noticing the relations, we decided to implement a full merge to “Employee” document as queries frequently would be about where an employee lives, what is there job, and which department they work in. thus, the joins and the queries could be expensive.

* “Dept” can be migrated into “Employee” as an employee will only work in 1 department, therefore we can remove the “DeptNO” attribute and merge the department name and its location into nested document.
* “Address” can be migrated into “Employee” as an employee will only live in 1 location, therefore we can remove the “AddressID” attribute as we will no longer need to access a unique address ID
* “Car” can be migrated into “Employee” as an employee may or may not have access to a company car, and if they do, they only have access to 1. This means a singular object can be displayed for each employee, and therefore will no longer need the unique car ID.
* “Job” can be migrated into “Employee” as it is only a single attributed collection, besides from the id, which once migrated can be removed as it is redundant data.

This then resulted in the logical data model shown in Figure 12. The reason we combined it all data into 1 document is because every employee has a 1 to 1 relationship with every attribute, they do not connect to multiple instances of said collections. Therefore, we it was appropriate to remove connections between collections and instead store each employee’s data in 1 document.

A screenshot of a computer

Description automatically generated with low confidence

Figure 12:Denormalized LDM

Once Denormalized it allowed us to alter the existing data into a prefixed format that could then be used to create the template for all new data input. The existing data was manipulated to the following format in Figure 13:

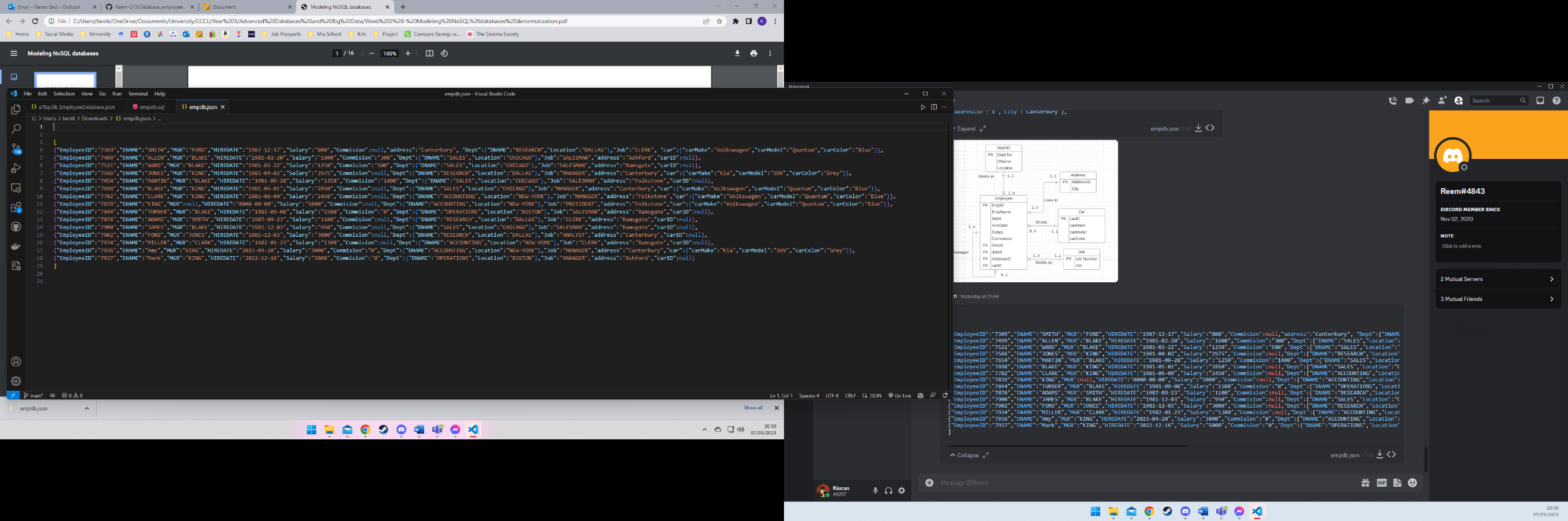


Figure 13: MongoDB JSON Data

### Technique

The same technique was used to create the front end when using MongoDB as was used for SQL. Once the logical data model was updated to suit MongoDB, to start the connection the following code was used in Figure 14.

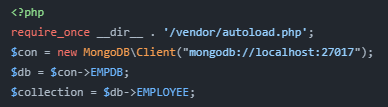


Figure 14: Connection to MongoDB

After establishing the connection, the code shown in Figure 14 was used to display the data inside the php page to allow users to view the contents of the database, this was done through fetching of the data and displaying it in a table as can be seen in Figure 15.

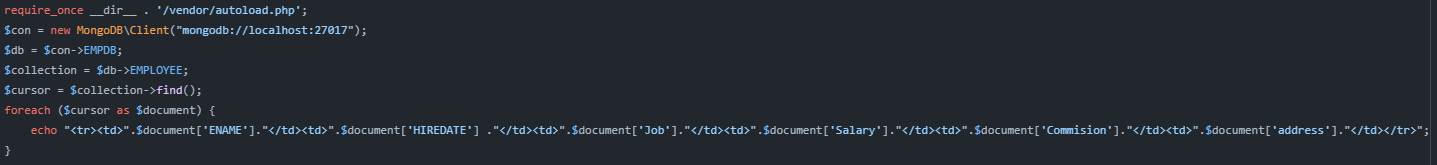


Figure 15: Retrieval of Employee Data

After that creating forms to enable the user to insert and add data into the employee and department tables.

### MongoDB Queries

The following queries have been created to match and display particular key values.

#### Query 1

This query captures all employees that work in accounting, and were hire between 1st June 1981 and 1st June 1982, it then displays their name and salary.

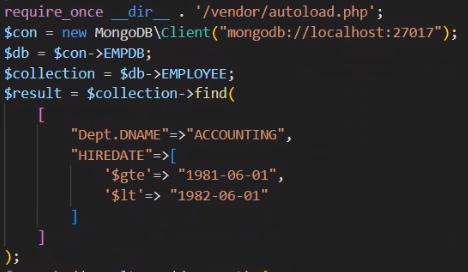


Figure 16: Input for Query 1

***A screenshot of a computer

Description automatically generated with medium confidence***

Figure 17: output for query 1

#### Query 2

This Query captures all employees who are employed as a manager, have a salary of over 2000 and do have a company car. It then displays their name, job, salary and address.

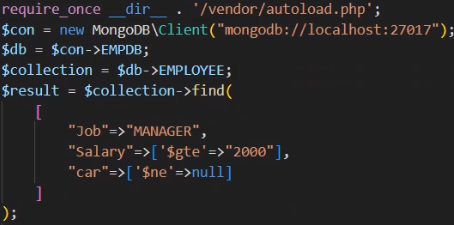


Figure 18: Input for Query 2

A screenshot of a computer

Description automatically generated with medium confidence

Figure 19: output for query 2

# Results

The final deliverable for this project allowed the user to specify which database they would like to work in and enabled them the choice to easily add to both employee and department tables in both SQL and MongoDB data tables.

# Conclusion

In conclusion, SQL databases and MongoDB are both powerful database management systems that have their own strengths and weaknesses. SQL databases are great for handling structured data with strict schemas, enforcing data integrity, and providing strong consistency guarantees. On the other hand, MongoDB excels in handling unstructured or semi-structured data, providing high availability and scalability, and allowing for more flexible data modelling.

One major difference between SQL databases and MongoDB is the way they handle queries. SQL databases use structured query language (SQL) to retrieve and manipulate data which can be complex and time-consuming for large datasets because it uses join to combine data from multiple tables, whereas MongoDB uses a document-oriented query language that allows for more flexible querying of unstructured data making it easier to retrieve and manipulate data in a single query because allows for nested and complex data structures.

In addition, SQL databases often require pre-defined schemas and strict data types, while MongoDB allows for more flexible and dynamic schemas. This means that in SQL databases, queries can be more predictable, but may also require more upfront planning and design. In MongoDB, queries may be more flexible, but may require more care to ensure data consistency and accuracy.

Ultimately, the choice between SQL databases and MongoDB will depend on the specific needs of your application. If your data is highly structured and consistency is critical, then SQL may be the better option. If you have a large amount of unstructured data or need a more flexible data model, then MongoDB may be the better choice. It's important to carefully evaluate your needs and consider the trade-offs of each option before deciding on specific database to use.

# Splitting of Tasks

Tasks were divided between the two of us based on our strengths and weaknesses, while there was some major overlapping in working individually, we did work well independently as well as coherently. Designing of the databases was a joint effort in proposal of ideas and discussing thoroughly to reach an outcome that was both agreed on.

Creating the PHP and database connectivity was primarily Reem’s task, however this was worked dually when obstacles occurred, such as connecting the MongoDB to the front-end in PHP whilst keeping the SQL connection viable.

Designing and creating the front end was a joint effort as we are both fully capable in doing this, therefore just speeding up the process whilst working on this.

Creating the queries was Kieran’s task with Reem’s input on testing to ensure full working queries.

Writing the report was also done by Kieran alongside creation of the project.

# Links

Github Repository - <https://github.com/Reem-313/Database_employee>

Demo video link - <https://www.youtube.com/watch?v=_oLvhxF_zJA>