Databases for developers

Final project report

Eshop

Simon Bucko

Group 12

20.11.2023

1. Table of contents

2. Introduction

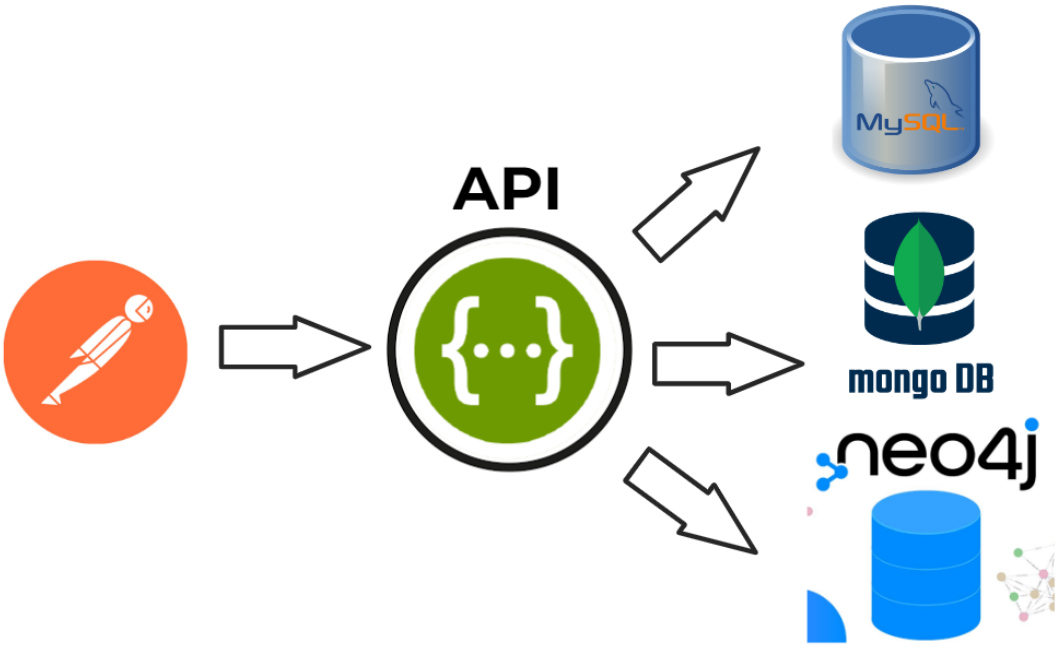
2.1. Problem description

As part of the final project for the databases for developers course, we were given a task to build a project, that would be complex enough to involve most of the syllabus and to consist of 3 databases, that we learned about during the course:

- MySQL
- MongoDB
- Neo4j

To reach this goal, I have decided to create an eshop project. Eshops, in general, include a lot of moving parts and a lot of data, which makes it perfect for me to decide the scope of my eshop to include just enough to cover the course syllabus.

Before starting the project, I needed to decide, what my project would look like. From the final project requirements, eshop needed to have 3 databases. Due to this requirement and the fact that I wanted to work alone on this project, I knew I would not have enough time to build a working frontend for my application. Because of that, I decided to mainly focus on the databases and create an API for each of them. This way I could also create documentation in swagger, and I could also map all endpoints to the postman so that I could keep interacting with the system. These decisions eventually led to a schema of my system, that looks like this:



2.2. Explanation of choices for databases, programming languages, and other tools

When building the project, I needed to use several tools, and programming languages and needed to choose databases. In this section, I would like to describe the reasoning behind using specific tools and technologies.

2.2.1. MySQL

MySQL is an open-source relational database management system (RDBMS) that is widely used for organizing and managing structured data. As an RDBMS, MySQL stores data in tables, which consist of rows and columns, allowing for efficient storage, retrieval, and manipulation of information.

MySQL is known for its scalability, performance, and reliability, making it suitable for a variety of applications, from small-scale projects to large enterprise-level systems. It supports standard SQL (Structured Query Language) for querying and manipulating data, and it offers features such as indexing, transactions, and data replication.

I have decided to use MySQL for the following reasons. Firstly, it is a database that we were using during the course, and thus it made it easy for me to work with it during my project. Secondly, I could reuse some of the scripts I built during lectures while working on exercises. Thanks to that I could achieve fast progress on my project. Lastly, MySQL has a big community of people, which made it easy for me to find help, when I needed it.

2.2.2. MongoDB

MongoDB is an open-source, cross-platform NoSQL database management system that diverges from traditional relational databases by adopting a document-oriented approach. In MongoDB, data is stored in flexible BSON (Binary JSON) documents, resembling JSON-like structures with key-value pairs. This departure from the tabular structure of relational databases provides a more dynamic and schema-less method for data storage.

As a NoSQL database, MongoDB does not rely on a fixed schema, allowing for greater flexibility in handling unstructured or semi-structured data. This characteristic is particularly advantageous when dealing with evolving data structures and diverse data types.

MongoDB is designed to be horizontally scalable, enabling it to efficiently manage increasing data volumes by adding more servers to a distributed database system. This scalability makes it well-suited for large-scale applications with growing data demands.

I have decided to use MongoDB for the following reasons. Firstly, it is a database that we were using during the course, and thus it made it easy for me to work with it during my project. Secondly, MongoDB has a big community of people, which made it easy for me to find help, when I needed it. Lastly, MongoDB has been around for some time, which means that their product is mature with a lot of features I could use while building my eshop.

2.2.3. Neo4j

Neo4j is an open-source graph database management system that employs a graph-based model for efficiently storing and querying highly interconnected data. Unlike traditional relational databases, Neo4j represents data as a graph, where nodes signify entities, relationships define connections between nodes, and properties provide additional information. Relationships are treated as first-class citizens in the data model, enabling efficient traversal and querying of complex connections.

Neo4j utilizes the Cypher query language, specifically designed for expressing graph patterns and navigating relationships. The database is scalable horizontally, allowing for the distribution of data across multiple servers to handle larger datasets and increased query loads. Neo4j adheres to ACID principles, ensuring data consistency and reliability in transactions, which is useful when building an eshop.

I have decided to use Neo4j mainly because it is a database that we were using during the course, and thus it made it easy for me to work with it during my project.

2.2.4. Express

Express, commonly known as Express.js, is a lightweight and flexible web application framework for Node.js. It simplifies the development of server-side web applications and APIs by providing a minimalistic, unopinionated structure. With features like a robust routing system, developers can define how the application responds to client requests, handling various HTTP methods and URL patterns.

I have decided to use Express mainly because it is a framework I have the most experience with when building backend applications or APIs, thus making it very suitable for this backend-heavy project. Lastly, I have also done research before starting the project and found a lot of libraries, that could help me with working with databases as well as building a documentation of my API endpoints.

2.2.5. Javascript

JavaScript is a high-level and dynamically typed programming language. The language supports asynchronous programming, utilizing features like callbacks, promises, and async/await to efficiently handle tasks such as data fetching from servers. JavaScript is also object-oriented, following a prototype-based approach, and with the introduction of ECMAScript 6 (ES6), it includes class-based syntax for more traditional object-oriented programming. JavaScript is also employed on the server side through platforms like Node.js, allowing developers to build scalable and efficient server applications using the same language.

Because of the possibility of using it as a server language and because I wanted to use Express for my API, Javascript came as the ultimate language for me to write the whole project in. Furthermore, building a project in JavaScript is fast, which was also important for me.

2.2.6. Sequalize

Sequelize is a npm Object-Relational Mapping (ORM) library for interacting with relational databases. As an npm (Node Package Manager) package, Sequelize simplifies the process of managing database operations by providing an abstraction layer over SQL queries. It supports various database systems, including PostgreSQL, MySQL, SQLite, and MSSQL, making it a versatile choice for developers working with different database engines.

Because of its ease of interacting with MySQL database, I decided to choose this ORM as my primary ORM language for this project, when working with MySQL database. Furthermore, Sequalize is also one of the most used and loved ORMs out there, which makes it easy to search for help when needed.

2.2.7. Mongoose

Mongoose is a npm package designed as an Object-Document Mapper (ODM) specifically for MongoDB. Functioning as an intermediary layer between JavaScript code and MongoDB, Mongoose introduces a structured, schema-based approach to database interactions. Models, constructed from these schemas, serve as the means to interact with the database, offering methods for tasks such as querying, saving, updating, and deleting documents.

Because Mongoose provides a layer between JavaScript code and MongoDB, it was a perfect fit for my project, as it provided me with an easy way to design my MongoDB as well as interact with it with queries.

2.2.8. Neode

Neode is a Node.js npm package designed to provide an Object-Graph Mapper (OGM) for interacting with the Neo4j graph database, functioning as an abstraction layer between Node.js applications and Neo4j. It offers an intuitive and expressive API for performing CRUD operations, traversing relationships, and executing complex queries. Neode encourages a schema-driven approach, making it easier to model and maintain the structure of graph data, while also supporting flexibility through its dynamic schema capabilities.

Because of its abstractions, I decided to use it as my OGM to simplify my queries towards my Neo4j database. Furthermore, after my research, it turned out to be the only well-documented and widely used OGM in the world of graph databases.

2.2.9. Postman

Postman is a widely used collaboration platform and API development tool that simplifies designing, testing, and documenting APIs. It supports a variety of request types, authentication methods, and environments, making it a versatile tool for testing APIs under different conditions. Beyond testing, Postman provides capabilities for automated testing, monitoring, and the creation of API documentation, streamlining the entire API development lifecycle.

Because of its comprehensive features and my previous experience with the Postman, I have decided to use it as my tool to test, document, and interact with my eshop project.

3. Relational database

3.1. Intro to relational databases

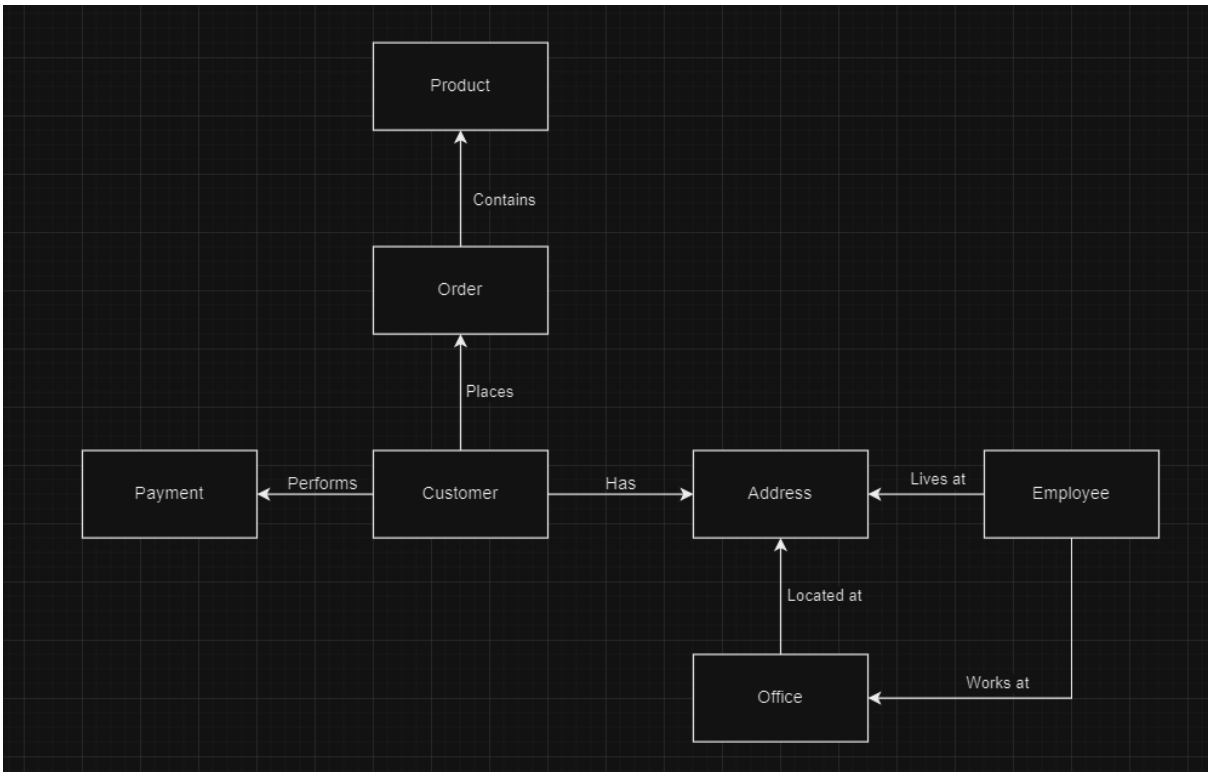
Relational databases are a type of database management system (DBMS) that organizes and stores data in tables consisting of rows and columns. This model is based on the principles of relational algebra, where data relationships are established through defined keys. Each table represents an entity, and the relationships between tables are established by linking common columns, known as keys. The Structured Query Language (SQL) is commonly used to interact with relational databases, allowing users to define, query, and manipulate data.

Key characteristics of relational databases include the ability to enforce data integrity through constraints, support for transactions to ensure atomicity, consistency, isolation, and durability (ACID properties), and the flexibility to perform complex queries across multiple tables using SQL.

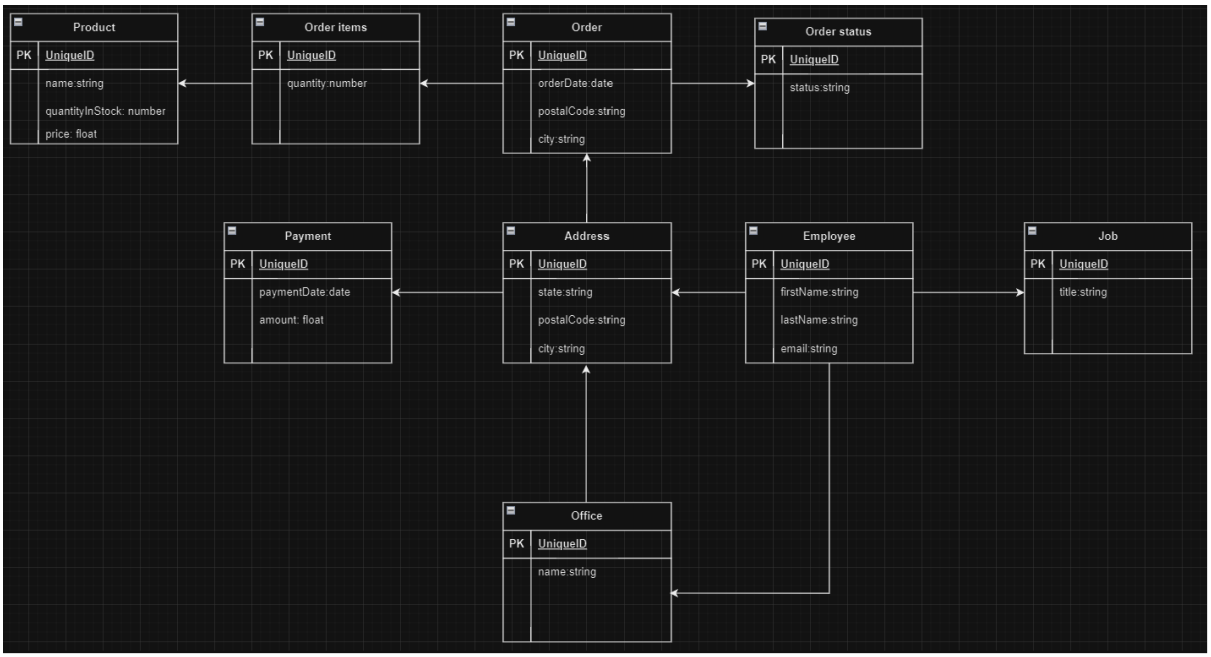
3.2. Database design

3.2.1. Relationship Model

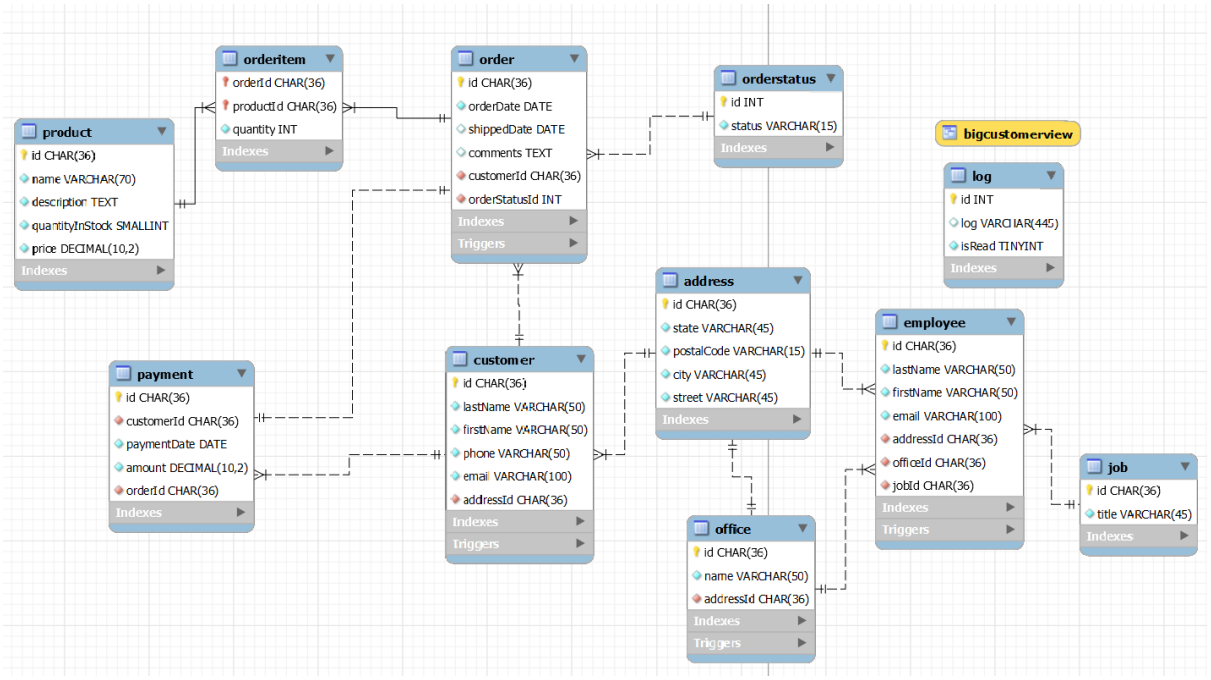
3.2.1.1. Conceptual model



3.2.1.2. Logical model



3.2.1.3. Physical model



3.2.2. Normalization process

Database normalization is a systematic process in database design that aims to reduce data redundancy and improve data integrity by organizing tables and relationships effectively. This process involves breaking down larger tables into smaller, related tables and establishing proper relationships between them. The goal of normalization is to minimize redundancy, dependency issues, and anomalies in the database structure.

There are 5 normalization norms, which explain step-by-step rules to achieve database normalization. However, in my eshop project, I have just fully completed only the first 3 of them, as they are considered the essential ones, and completing the remaining 2 would not bring me any further gain in my database design.

3.3. Physical data model

3.3.1. Primary keys

In a relational database, a primary key serves as a unique identifier for each record within a specific table. This identifier is essential for ensuring data integrity and facilitating relationships between tables.

There are multiple automated generation mechanisms, such as auto-incrementing integers or GUIDs, are commonly used to ensure the uniqueness and consistency of primary key values. This automated approach simplifies the process of managing primary keys. I am personally a huge fan of using GUIDs, as they provide uniqueness and consistency over a long time. Furthermore, from the security point of view, they restrict a potential attacker to get knowledge about how many rows my database has, and it usually saves a roundtrip to a database, when creating a new entity. Despite GUIDs being criticized for not being a natural primary key of the entity, I still find them extremely useful, when creating primary keys for my tables.

Because MySQL does not support GUID data type by default, I was forced to use a VARCHAR(36) to properly store my primary keys as GUIDs, when working with them.

3.3.2. Indexes

Database indexes are essential components that significantly enhance the efficiency of data retrieval operations in a database. Indexes are created on one or more columns of a table, serving as optimized structures that streamline the process of locating and retrieving data.

The indexes are dependent on the type of queries the application is running and how often they are being run. Having this in mind, I have decided to only implement indexes on my Product table. The main reason is, that this table would be mostly queried if I had any frontend, where customers could see the product of my eshop, as well as they would be able to search and sort the results. Because of this requirement, I have decided to implement a full-text search on the name column, to support string queries. In addition to this, I have also created an index on the price column, as I was expecting customers to use search by the most expensive or least expensive value. Using these two indexes combined, I could ensure that I have improved the performance of my queries to the product table. Furthermore, products are not added as frequently as read, and therefore it makes total sense to use indexing in this case scenario.

However, I faced a problem when trying to set up a full-text search on the cloud database, and for some reason, it did not allow me to create a full-text index on the name column in my product table.

3.3.3. Constraints and referential integrity

In the realm of database management, constraints, and referential integrity are pivotal concepts that contribute to the reliability, consistency, and accuracy of data. Constraints serve as rules or conditions applied to the data in database tables, ensuring that it meets predefined criteria. These rules help maintain data integrity and prevent the entry of inconsistent or invalid information.

For most of my foreign key constraints, I have decided to use RESTRICT on the update. The main reason is that I was using GUIDs as my primary keys. The second reason was, that I do not want anyone at any time to change a foreign key as it can potentially create a lot of problems and potential inconsistencies in the database.

Most of my on-delete restrictions also contain RESTRICT on the delete, as I evaluated them as the most appropriate for the relationship and for the business use case. However, in some places, I needed to use NO ACTION, as in some cases I needed to preserve data for auditing purposes, or for archive data for statistical reasons.

Depending on the use case, some of the tables and columns also have constraints on uniqueness. For example, I only allow unique emails on the employee table, as I do not want to have 2 employees in my system, with the same email.

3.4. Stored objects

3.4.1. Stored procedures

Stored procedures are precompiled and stored sets of one or more SQL statements that can be executed as a single unit. These procedures are stored in a database management system (DBMS) and can be called and executed from applications or other stored procedures. Stored procedures offer several advantages in terms of modularity, reusability, security, and performance optimization. Because of their power of being executed as a unit, I have decided to use them when a customer places an order.

Placing an order is an action, that includes multiple SQL queries, and trying to make separate calls from my API to my database for each query would slow down the action of placing the order, which is critical for a business to be relatively fast and with high performance, especially during rush hours, or black Fridays.

Down below you can find a detailed stored procedure for placing an order.

```
CREATE DEFINER=`sibu`@`%` PROCEDURE `placeOrder`(
    IN customer_id CHAR(36),
    IN product_ids VARCHAR(255),
    IN quantities VARCHAR(255),
    IN total_price DECIMAL(10,2)
)
BEGIN
    DECLARE i INT;
    DECLARE product_id CHAR(36);
    DECLARE quantity INT;
    DECLARE order_id CHAR(36) DEFAULT uuid();

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    START TRANSACTION;

    INSERT INTO `order` ( `id`,`orderDate`,`customerId`) VALUES (order_id, current_date(), customer_id);

   SET i = 1;
   WHILE i <= getItemsCount(product_ids) DO
       SET product_id = SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', i), ',', -1);
       SET quantity = SUBSTRING_INDEX(SUBSTRING_INDEX(quantities, ',', i), ',', -1);

       INSERT INTO `orderitem` (`orderId`, `productId`, `quantity`) VALUES (order_id, product_id, quantity);

       -- this will throw error if new quantity < 0
       UPDATE `product` SET `quantityInStock` = `quantityInStock` - quantity WHERE `id` = product_id;

       SET i = i + 1;
   END WHILE;

   INSERT INTO `payment` (`customerId`, `paymentDate`, `amount`, `orderId`) VALUES (customer_id, current_date(), total_price, order_id);

   COMMIT;
END
```

3.4.2. Stored functions

Stored functions, also known as stored procedures with a return value, are database objects that encapsulate a set of SQL statements and return a single value or a table. These functions are stored in a database management system (DBMS) and can be called from within SQL queries, other stored procedures, or applications. Stored functions offer a way to encapsulate reusable logic and calculations within the database.

Because I could not find any built solution in MySQL to count items in an array, I needed to build my own function, which can take an array of strings and return me the count of items in the array. This helper function was extremely helpful in my stored procedure to place an order, as I needed to know how many items had been bought. Down below you can see the implementation of my stored function. To see its usage, please refer to the code in the stored procedure above.

```
CREATE DEFINER=`sibu`@`%` FUNCTION `getItemsCount`(
    input_list VARCHAR(500)
) RETURNS int
    DETERMINISTIC
BEGIN
    DECLARE items_count INT;
    SET items_count = LENGTH(input_list) - LENGTH(REPLACE(input_list, ',', '')) + 1;
    RETURN items_count;
END
```

3.4.3. Views

View is essentially a virtual table derived from the result of a SELECT query. Unlike a physical table, a view doesn't store data itself but rather acts as a dynamic representation of the result set of a specific query. Views serve several purposes, enhancing the management and usability of databases.

Views offer a layer of abstraction over the underlying tables, allowing users to query and interact with data without having to comprehend the intricacies of the SQL statements or the underlying data model. This abstraction promotes simplicity and aids in maintaining a clear separation between the database structure and the user interface.

For my eshop project, I thought about creating a view, that would return the biggest customers of my eshop. This would be a pretty useful query for for example marketing, as they could try to resell something to these biggest customers, or they could get some discount for the royalty and amount of sales they provide. Down below you can find the SQL query I used to create the view for the biggest customers of my eshop.

```
 VIEW `bigcustomerview` AS
    SELECT
        `c`.`id` AS `id`,
        `c`.`firstName` AS `firstName`,
        `c`.`lastName` AS `lastName`,
        `c`.`email` AS `email`,
        SUM(`p`.`amount`) AS `total purchase amount`
    FROM
        (`customer` `c`
        JOIN `payment` `p` ON ((`p`.`customerId` = `c`.`id`)))
    GROUP BY `c`.`id` , `c`.`firstName` , `c`.`lastName` , `c`.`email`
    ORDER BY `total purchase amount` DESC
```

3.4.4. Triggers

Triggers are specialized stored procedures that automatically execute in response to specific events on a particular table or view. These events, or triggers, can include actions such as INSERT, UPDATE, DELETE, or a combination. The primary purpose of triggers is to enforce specific business rules, maintain data integrity, or automate certain tasks without requiring explicit intervention from the application code.

One of the use cases I found triggers useful, was to ensure that each time an order's shipped date changed from null to some date, a trigger would automatically set the order status to shipped. This was super convenient as I did not need to explicitly write a code logic for this when marking an order as shipped. Down below you can find the code for the trigger.

```
CREATE DEFINER=`sibu`@`%` TRIGGER `updateOrderStatusWhenOrderShipped` BEFORE UPDATE ON `order` FOR EACH ROW BEGIN
    IF NEW.shippedDate IS NOT NULL AND OLD.shippedDate IS NULL THEN
        SET NEW.orderStatusId = 2;
    END IF;
END
```

3.4.5. Events

Events refer to occurrences or happenings within a database system that can trigger specific actions or processes. These events can range from database-related actions, such as the execution of SQL statements, to system-related activities like scheduled tasks or user interactions. Understanding and managing events is crucial for automating tasks, enforcing business rules, and maintaining the integrity and efficiency of a database system.

In my project, I have used events to clean up old logs each day, which have been read. To create such an event I have used the following SQL script to do so.

```
CREATE EVENT clear_read_logs
ON SCHEDULE EVERY 1 DAY STARTS CURRENT_TIMESTAMP
DO
DELETE FROM logs WHERE isRead = TRUE;
```

3.5. Transactions

Transactions represent a fundamental concept crucial for ensuring the consistency, integrity, and reliability of data operations. A transaction is a sequence of one or more database operations that are executed as a single, indivisible unit. The primary goal of transactions is to maintain the database in a consistent state, even in the presence of failures or errors during the execution of operations.

These features of transactions were extremely useful when dealing with a stored procedure of placing an order. I needed to manipulate multiple tables when placing a new order and an error in any part of these multiple SQL scripts should cause other previous actions to be reverted to their previous state. The persistence of all steps, that happened in the

transaction, only happened at the very end of the transaction by the commit statement. To see the code of this transaction, please refer to the code in the stored procedures section.

3.6. Auditing

For auditing I have decided to implement a solution, where I would delete read logs from my logs table. I have decided to use the log table as a place where I could write some important logs from stored procedures and triggers. After, that my idea was that there could be an external system that could read these logs, write them into an external logging system, and notify an admin if there are any errors or critical logs, that he/she should take a look at. After doing this, this external service would mark read logs as read, after which my auditing system would delete them using an event once per day. This way I would also give time to the admin to take a look at the original logs before they are deleted from the system, or if there was any error in the external system, while it was trying to write the logs to the external logging system, the admin could easily reproduce the problem and find the bug in the system. Unfortunately, I did not have time to implement this external system, that would be performing this job, but at least I have set up the table and events.

3.7. Security

3.7.1. Explanation of users and privileges

To ensure the highest security possible and ensure I followed the best practices when working with MySQL, I have created several users when working with the database:

- admin
- application user
- readonly
- HR manager

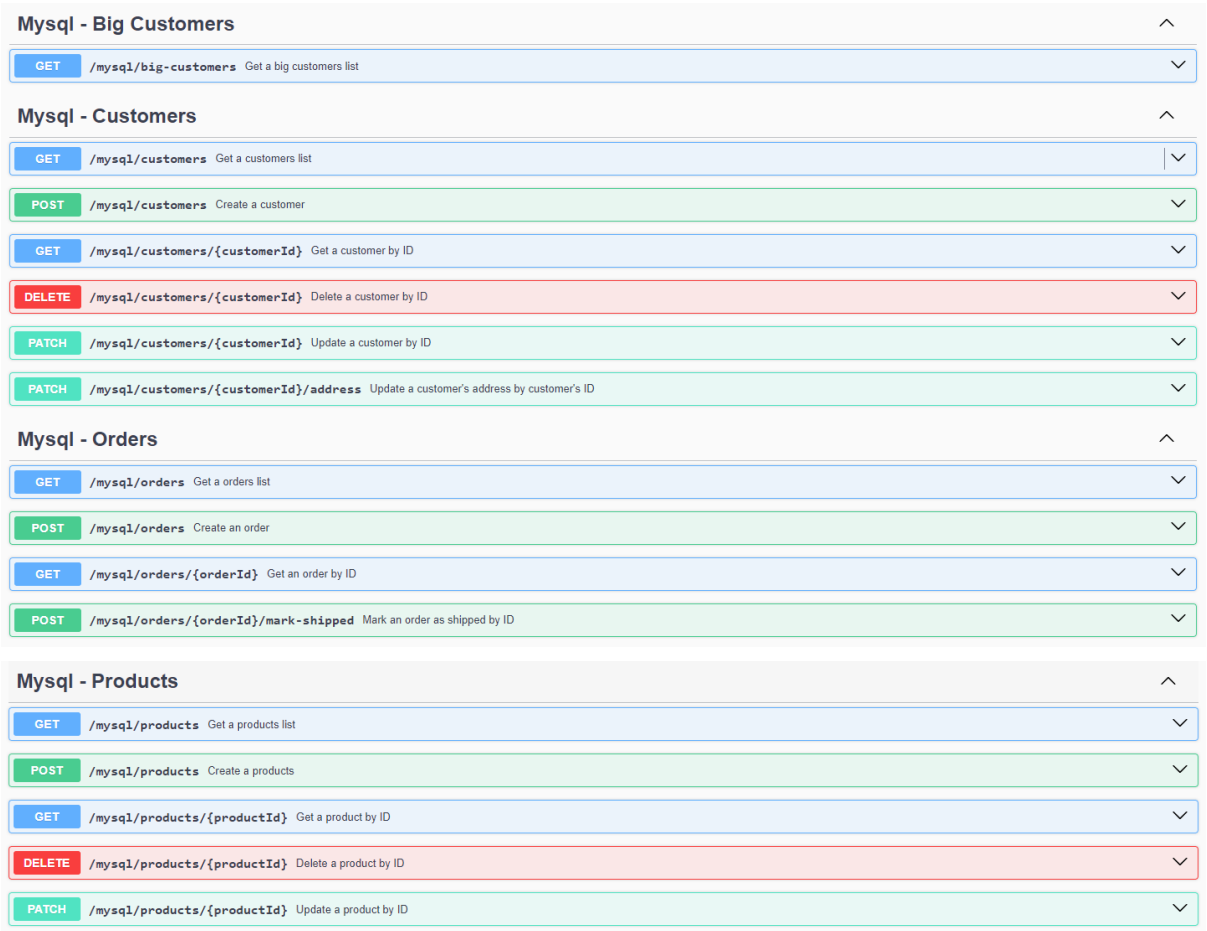
Admin is my super user, which I used to set up all schema, stored procedures, triggers, stored functions, and other users. The application user is a user I have created for my API. This user is only allowed to perform CRUD operations on all tables but is not allowed to perform schema changes, create new users, or drop the whole database. The readonly user was created with the idea of having a user, who is just like the application user, but he/she can only read all tables and nothing else. HR manager is a user, who should be only responsible and allowed to see employee table. Because of that, the HR manager is only allowed to read the data in the employee table and nothing else.

3.7.2. SQL injection prevention

To prevent SQL injection in my application, I have decided to always use the ORM when communicating with my MySQL database instead of writing my own SQL statements. The ORM uses prepared statements under the hood, which ensures that only the intended query will be made to the database and it will prevent a frontend user from creating a malicious SQL query and getting it executed on the database.

3.8. CRUD application for RDBMS

As mentioned in the introduction, I have created an API solution around my RDBMS system. To give an overview of all functionality, that is possible with the system I created the following swagger documentation:



4. Document database

4.1. Intro to document databases

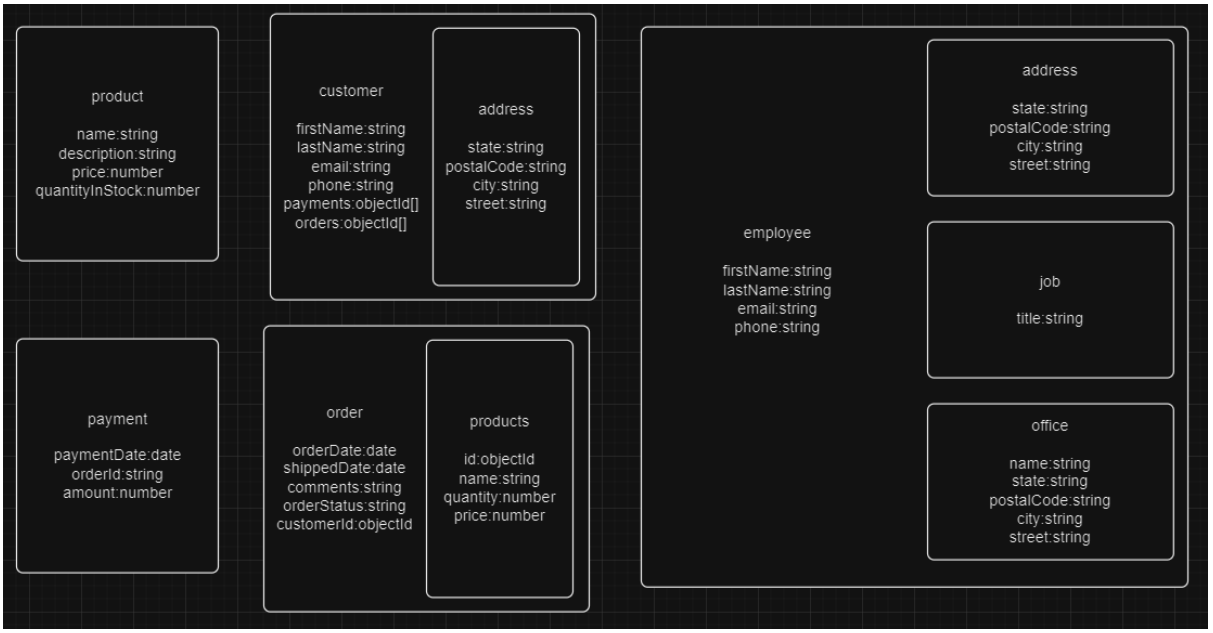
Document databases represent a paradigm shift in database design, moving away from rigid table-based structures to embrace a more flexible and scalable approach. Unlike their relational counterparts, document databases store data in flexible, schema-less documents, typically in JSON or BSON format. Each document encapsulates related data and can vary in structure, enabling the storage of heterogeneous data types within the same database.

The cornerstone of document databases is their ability to handle semi-structured and unstructured data with ease. This is particularly advantageous in scenarios where the data's structure is not predetermined or where the schema may evolve. Document databases empower developers to work with data in a way that mirrors the natural structure of their applications, fostering agility and adaptability.

One of the prominent features of document databases is their support for nested and hierarchical data structures. This facilitates the representation of complex relationships within a single document, reducing the need for multiple tables and complex joins. As a result, document databases excel in scenarios where data relationships are inherently hierarchical or nested, such as in content management systems, e-commerce platforms, and real-time analytics.

4.2. Database design

When designing a document database I have used a physical model of my MySQL database and tried to denormalize it. Using this technique I could get rid of some of the relationships between entities and embed them into one single document. This not only simplified the whole database but also allowed me to create documents, which are much more optimized for querying and do not require complex joins across multiple tables. The final database models can be found below:



4.3. CRUD application for the document database

As mentioned in the introduction, I have created an API solution around my MongoDB system. To give an overview of all functionality, that is possible with the system I created the following swagger documentation:

5. Graph database

5.1. Intro to graph databases

Traditional relational databases, while effective for many use cases, often struggle to efficiently model and traverse complex relationships. Graph databases address this limitation by providing a specialized and efficient approach to representing and querying interconnected data.

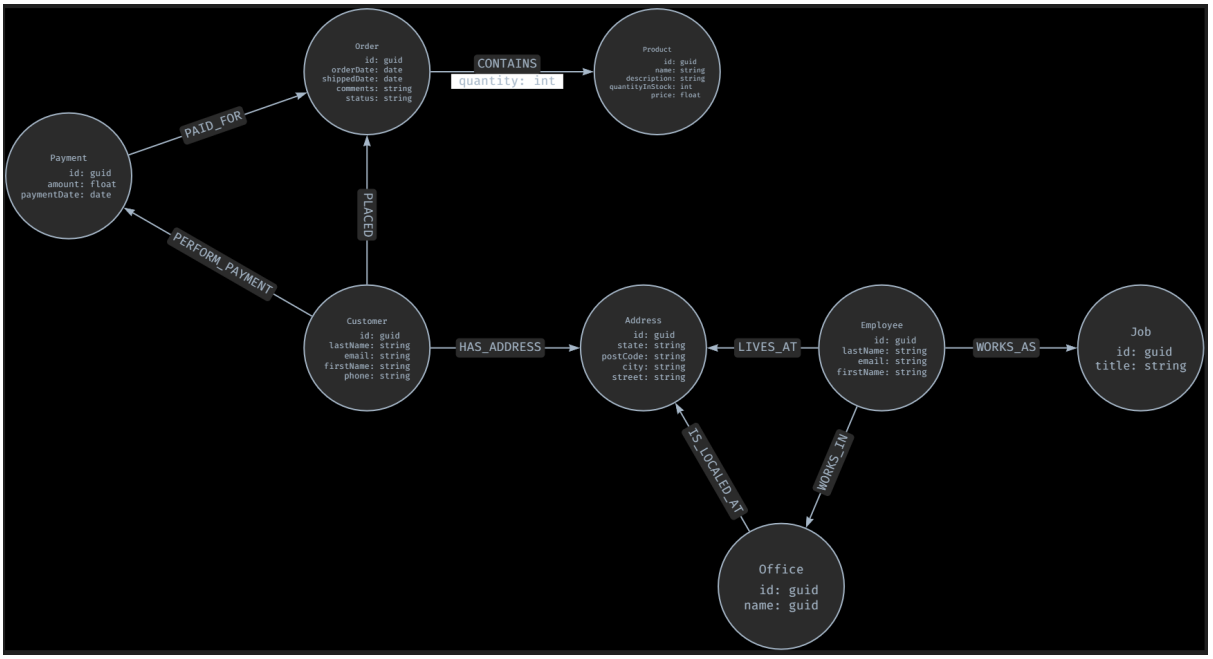
Graph databases are built on the principles of graph theory, a mathematical discipline that studies the relationships between entities. In a graph database, data is structured as nodes, which represent entities, and edges, which denote relationships between these entities. This elegant and intuitive model allows for the representation of complex networks, making graph databases particularly well-suited for scenarios where relationships are central to the data.

One of the defining characteristics of graph databases is their ability to traverse relationships with exceptional speed and efficiency. Traditional databases, reliant on complex JOIN operations, can struggle as the number of relationships grows. Graph databases, on the other hand, employ specialized algorithms that excel at navigating intricate networks of connections, making them ideal for use cases such as social networks, recommendation engines, fraud detection, and network analysis.

The flexibility of the graph data model is another key advantage. Unlike rigid, tabular structures of relational databases, graph databases can easily adapt to changing data models. Nodes and relationships can be added, modified, or removed without the need for extensive schema modifications, providing a level of agility that aligns well with the dynamic nature of many contemporary applications.

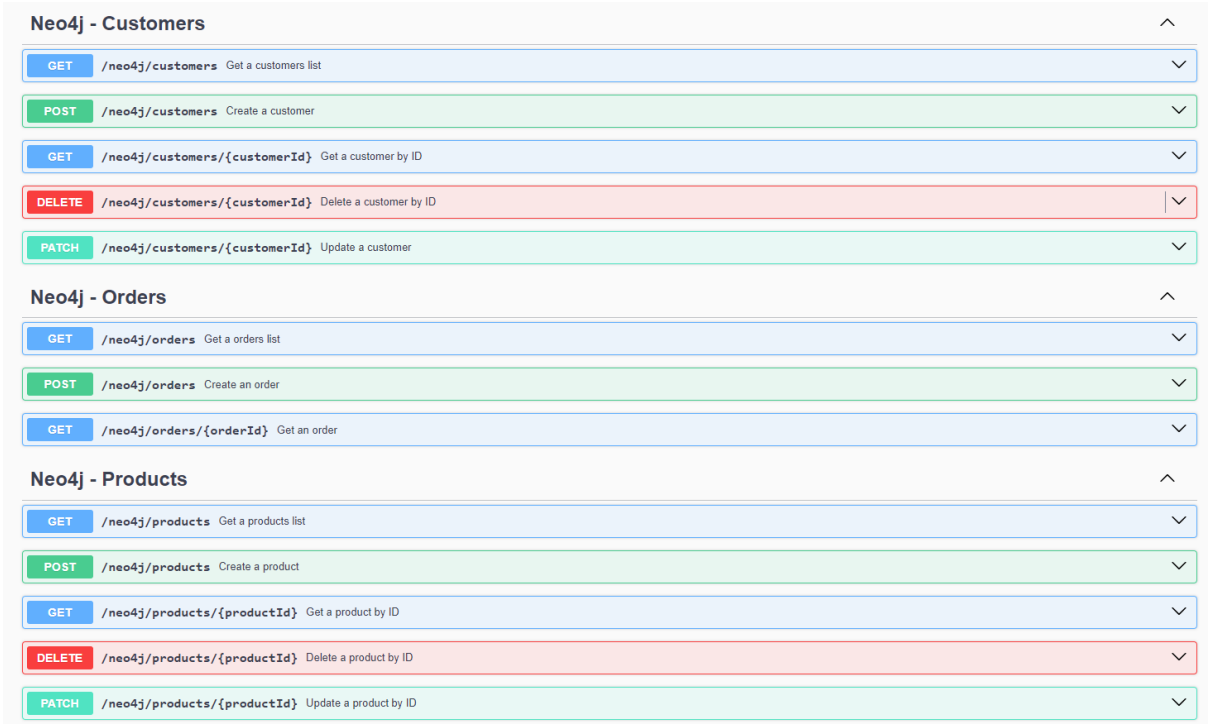
5.2. Database design

When designing a graph database I used a physical model of my MySQL database and tried to denormalize it and alter many to many relationships into edges between nodes. Using this technique I could get rid of some of the relationships between entities and create a much simpler model, that represented the same data. This not only simplified the whole database but also allowed me to create nodes, which are much more optimized for querying complex relationships. The final database models can be found below:



5.3. CRUD application for the graph database

As mentioned in the introduction, I have created an API solution around my Neo4j system. To give an overview of all functionality, that is possible with the system I created the following swagger documentation:



6. Conclusions

Working with 3 different databases during a single project allowed me to see their similarities and differences. When it comes to similarities, I have to say, that all databases allowed me to replicate the business logic for my eshop project without any big showstopper. I have never found myself in a situation, where any of these databases would make it impossible for me to implement a certain feature or support some business use case. Furthermore, using ORM created an abstraction for me, and from time to time it felt like I was only working with one type of database. I would like to also point out that all databases support transactions and that made it easy for me to create transactional actions, which could easily reverted in case of error. However, there were some differences, and each database was shining in a specific scenario.

When it comes to MySQL, I would like to say that MySQL was of all other databases the most mature when it came to preserving data integrity. It stands out as it allowed me to create stored procedures, functions, triggers, and events directly on the database, which could be called from my code. Despite the other two databases also supporting some sort of triggers and functions, I have to say that MySQL does not have an opponent when it comes to complex transactions. I also have to say, then from all the databases, sequalize ORM felt like the most solid one with rich and robust documentation, which has been used by many developers.

When it comes to MongoDB, I would like to say that MongoDB was the database, that took me the least time to set up thanks to it being a very relaxed model structure. One of the best features that MongoDB provides is the versioning of the models, which allowed me to perform any later changes to my models without resulting in a lot of breaking changes. The Mongoose npm package felt very solid when interacting with MongoDB. Lastly, I would like to say, that the embedding feature of MongoDB allowed me to create the most concise database model, which was also extremely easy to understand and was well structured for a huge load requests.

When it comes to Neo4j, I would like to say that the database worked perfectly, but I had a hard time setting it up with the ORM for it. This was the main issue as there are not many people using these types of databases, and hence lots of problems I have encountered had to be solved to me and I had a hard time getting help online. The documentation also lacked a lot of important details. But when it comes to the visualization of the data, I liked the visualization of the data the most when using Neo4j from all databases. Unfortunately, I felt like I could not use the true power of graph traversal in Neo4j, as my application did not require a complex relationship to traverse such as a product recommendation or similar.

Overall I think I succeeded in completing the requirements for my eshop with all 3 types of databases and got a deeper insight on which database to use in which use case based on its strengths and weaknesses.

7. References

During the process of building the project and writing the report, I needed to obtain some information online. Because of that, below I am involving references, that have been used:

- https://www.techtarget.com/searchoracle/definition/MySQL
- https://www.techtarget.com/searchdatamanagement/definition/MongoDB
- https://en.wikipedia.org/wiki/Neo4j
- https://en.wikipedia.org/wiki/Express.js
- https://en.wikipedia.org/wiki/JavaScript
- https://sequelize.org/
- https://mongoosejs.com/docs/
- https://www.npmjs.com/package/neode
- https://www.postman.com/product/what-is-postman/
- https://www.oracle.com/database/what-is-a-relational-database/
- https://en.wikipedia.org/wiki/Database_normalization
- https://en.wikipedia.org/wiki/Database_index
- https://intelligent-ds.com/blog/what-is-referential-integrity#:~:text=Referential%20integrity%20is%20a%20constraint,help%20you%20maintain%20data%20quality.
- https://www.javatpoint.com/mysql-stored-function
- https://dev.mysql.com/doc/refman/8.0/en/create-view.html
- https://www.mysqltutorial.org/mysql-triggers/
- https://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/
- https://www.codecademy.com/article/introduction-to-nosql
- https://neo4j.com/developer/graph-database/