

Database Exam Project

Meraki webshop

By Nikola Wulf-Andersen

1. semester KEA 25-05-18



TABLE OF CONTENTS

Introduction	3	Choice of databases	17
The project	4-11	Execution & commands	18-27
My Account	5	Setting up database in MySQL	18
Shop	6	ER diagram in MySQL	19
Single view	7	CRUD	20
Cart	8	Prepared Statements	22
Dashboard - Admin	9	Transactions	23
Users - Admin	10	Shop filters	24
Products - Admin	11	VIEWS	25
Normalization	12	Stored procedures	26
First Normal Form	12	Triggers	27
Second Normal Form	12	Execution & commands MongoDB	28-31
Third Normal Form	12	Install MongoDB	28
Structure & datatypes	13-14	Starting Mongo	28
Entity Relationship diagram	13	Creating database	28
Datatypes	13	CRUD	29
Primary and foreign keys	14	Reflection	32
ER diagram	15	Conclusion	33
Databases	16	Literature	34
Text files & MongoDB	16		
MySQL	16		
SQLite	16		

INTRODUCTION

This exam report is about databases and the understanding of its structure, datatypes, different types of databases and the creation of the database itself. To illustrate the theory in praxis I created a project that shows two types of databases and how they are managed.

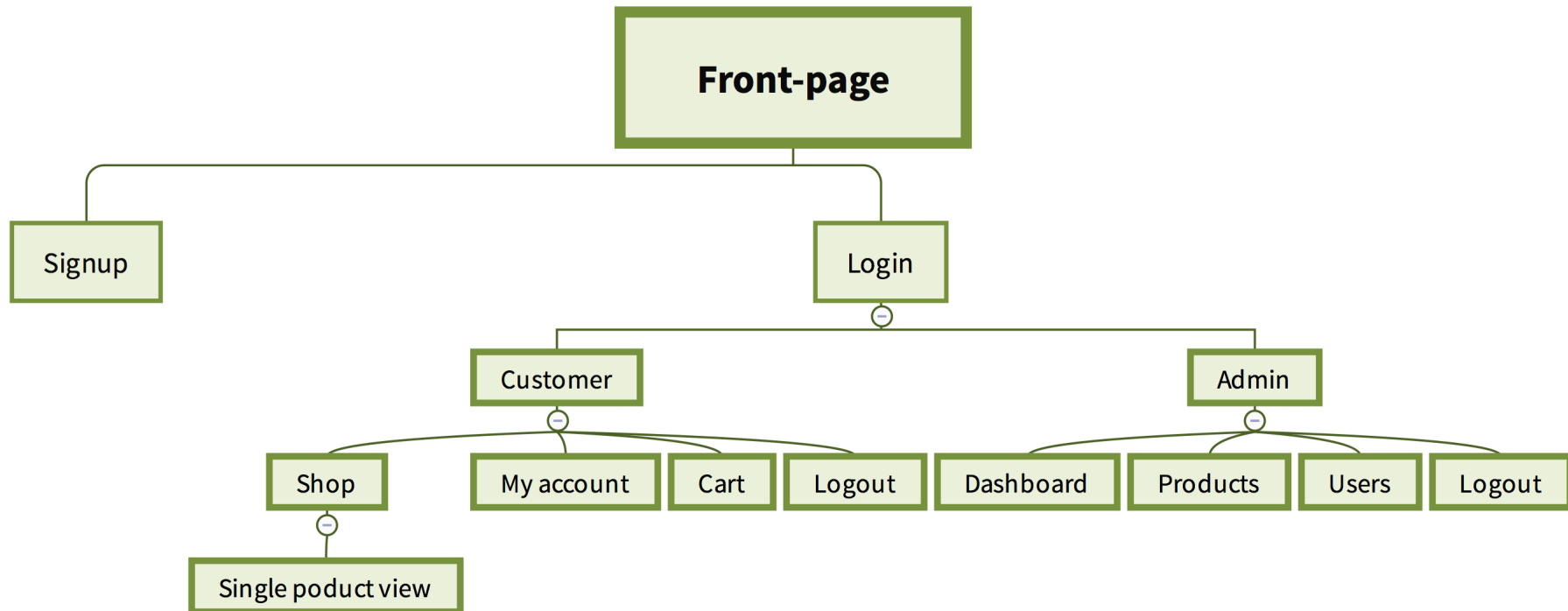
Throughout this report I will explain the theory behind databases such as normalization, entity relationship diagrams, datatypes, different types of databases and the execution of the project itself.

I chose to create a project simulating a web shop which is close to a real-world scenario. It is an important part of the process to actually think about the structure and relationships between the tables as if it were a real project with actual data. Though the project is just a prototype and not a full-blown web shop, it demonstrates the important features and creates a context for the data which is the main purpose of it.

THE PROJECT

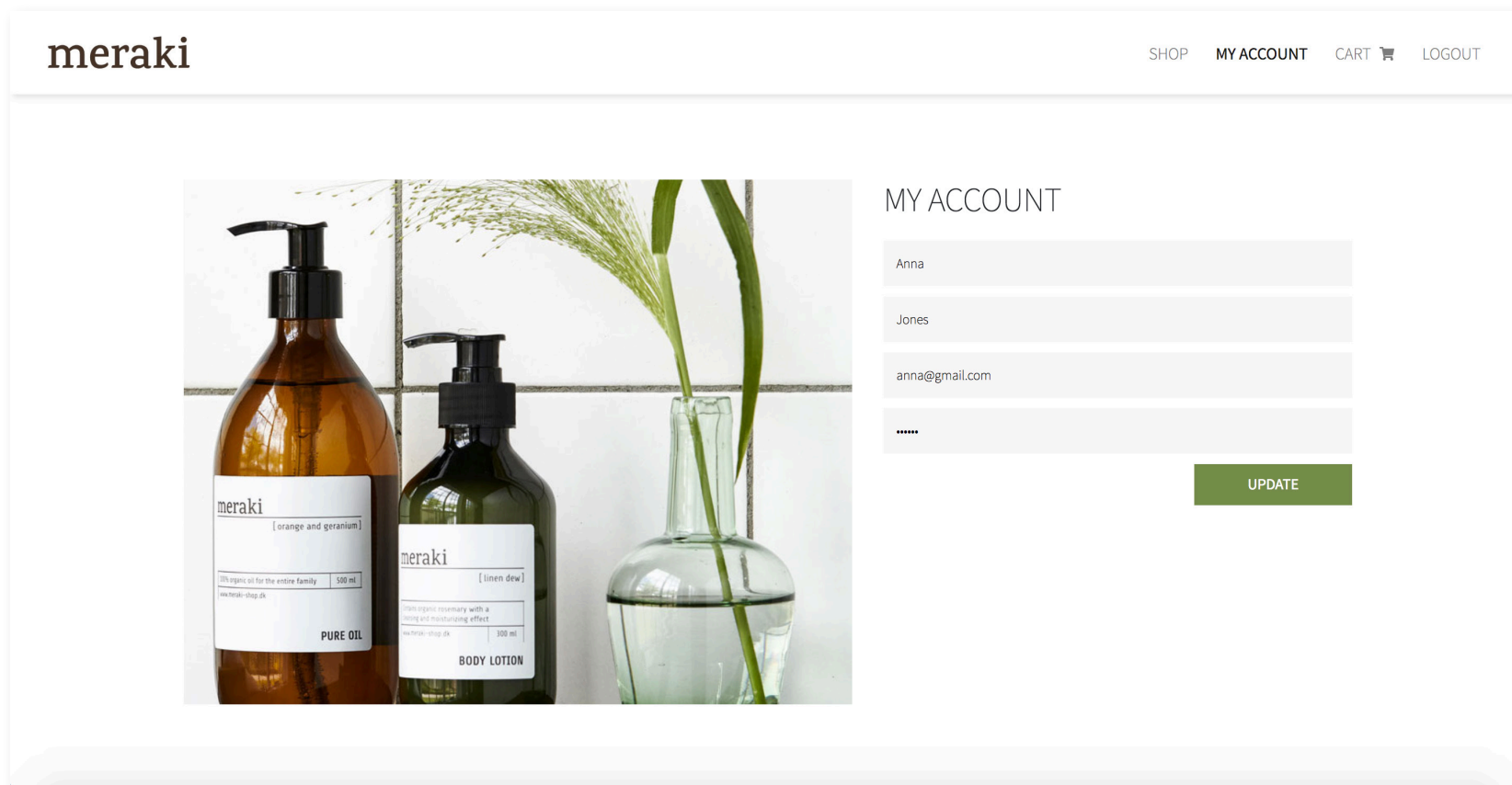
In the beginning of the process I started out creating a basic sitemap to get an overview. The project consists of a front-page where the user can choose to signup or login to their account. If the user doesn't have an account they can create one, and then have the ability to login and start shopping. When logged in they can look through the shop pages, click on items and add them to their cart. They can also edit their profile and logout when they are done shopping.

To manage the web shop, I created a user profile for myself with the status set to admin. If I login to the web shop I will get an overview of the dashboard, all the products and users in my system. In the administrator site I will have the option to delete, update and create both products and users.



MY ACCOUNT

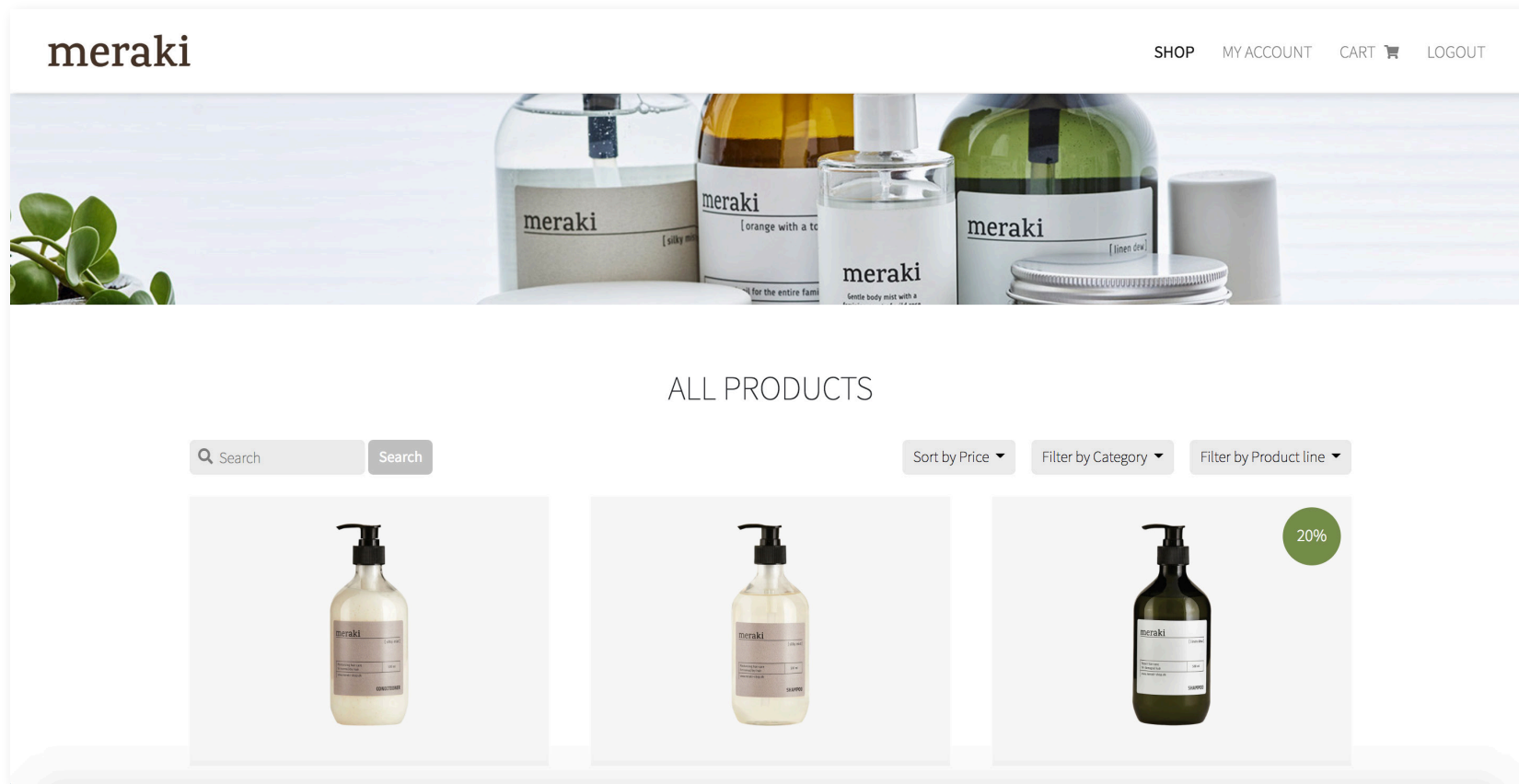
The first thing a user will meet when they login is their account page. On this page is their personal information shown like their name email and password and they have the option to update the information. The users actually have a bit more information than displayed here such as their id and status.



SHOP

When the users are logged in they have access to the shop. The shop displays all the products in the web shop. At the top of the page is a search/filter bar. Here the users have the option to search for a product or filter the page by category or product line. They also have the opportunity to sort the

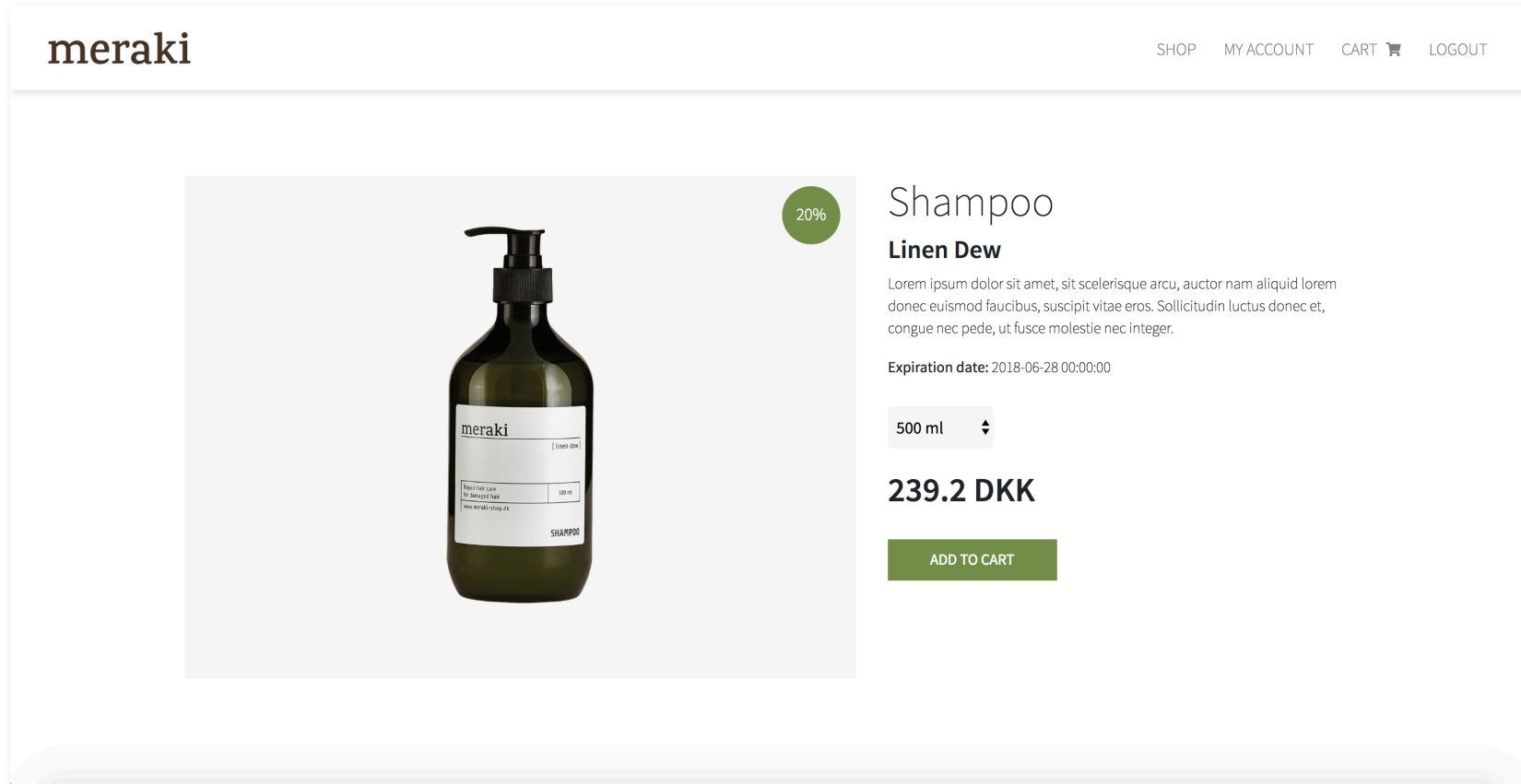
products by price either from high-low or low-high. If a product is on sale a green bobble will appear with the percentage and the reduced price will replace the original one.



SINGLE VIEW


If the user clicks on a product a new page will open and show that product only, but with additional data about the product such as a detailed description, size selection and expiration date. The user can select the desired


size and add the product to the cart. When a product is added to the cart it will be saved in the table cart and the time and current price will be logged.



CART

Each product in the cart is displayed with an image, category, size, product line, description, price and quantity. The user can remove an item from the shopping list by clicking on “remove” on a chosen item. In the bottom of the cart the total price and number of items are calculated.


meraki SHOP MY ACCOUNT CART  LOGOUT




Conditioner, 750 ml
Silky Mist
Lorem ipsum dolor sit amet, sit scelerisque arcu, auctor nam aliquid lorem donec euismod faucibus, suscipit vitae eros. Sollicitudin luctus donec et, congue nec pede, ut fusce molestie nec integer.

299 DKK

- 1 +


REMOVE 



Shampoo, 500 ml
Linen Dew
Lorem ipsum dolor sit amet, sit scelerisque arcu, auctor nam aliquid lorem donec euismod faucibus, suscipit vitae eros. Sollicitudin luctus donec et, congue nec pede, ut fusce molestie nec integer.

239.2 DKK

- 1 +

REMOVE 

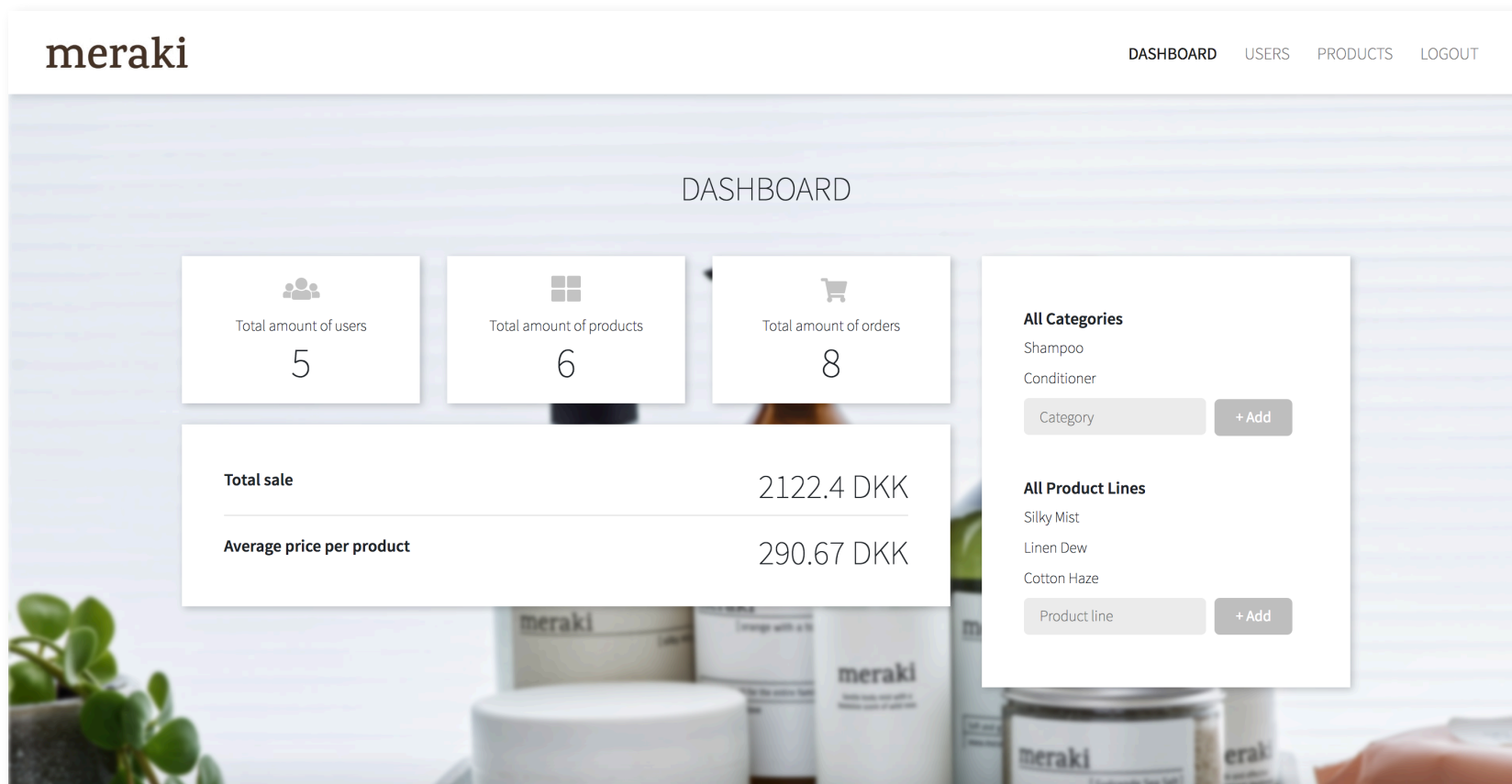
Total price
538.2 DKK
2 itmes

GO TO CHECKOUT

DASHBOARD

When the web shop administrator is logged in, in this case me, I will have access to information via the dashboard as well as all the users and products in the system. The dashboard contains some overall information about

users, products, sales, categories, product lines and so on. I also have the option to create new categories and product lines in the shop.



USERS - ADMIN

In the users page all users are displayed in a table with their name, email and id. As administrator I have authority to delete, update and create users. If the little trash icon is clicked that specific user will be deleted from the system. To edit a user, I can click the little pencil and then a new row be-

neath the user will appear, where the new information can be entered and then updated. To create a new user, I can click on the button "Add user" and a template will appear where the information for a new user can be entered and saved.

First name	Last name	Email	Id		
Nikola	Wulf-Andersen	nikola@wulfandersen.dk	5ae609f928be1		
Christian	Hagen	christianhagen@live.dk	5ae636e6d63d0		
<input type="text" value="Christian"/>	<input type="text" value="Hagen"/>	<input type="text" value="christianhagen@live.dk"/>	<input type="button" value="UPDATE USER"/>		
Anna	Jones	anna@gmail.com	5ae2da4f333d		
Mads	Larsen	mads@larsen.dk	5aef7261ca905		
Laura	Johanson	laura@gmail.com	5af4984f86f4c		




PRODUCTS - ADMIN

The products page contains all the products in the web shop, displayed in a table. Equivalent to the users page the administrator has the authority to delete, update and create products. As the users page the little trash icon is for delete, the pencil is for edit and the button at the top “Add product” is for creating new products.

meraki
DASHBOARD USERS **PRODUCTS** LOGOUT

ALL PRODUCTS

+ ADD PRODUCT

Product image	Category	Productline	Sizes	Description	Expirationdate	Price	
	Conditioner	Silky Mist	500	Lorem ipsum dolor sit amet, sit scelerisque arcu, auctor nam aliquid lorem donec euismod faucibus, suscipit vitae eros. Sollicitudin luctus donec et, congue nec pede, ut fusce molestie nec integer.	2018-07-31 04:00:00	299	🗑️ ✎️
	Shampoo	Silky Mist	500	Lorem ipsum dolor sit amet, sit scelerisque arcu, auctor nam aliquid lorem donec euismod faucibus, suscipit vitae eros. Sollicitudin luctus donec et, congue nec pede, ut fusce molestie nec integer.	2018-06-22 04:00:00	249	🗑️ ✎️
	Shampoo	Linen Dew	500	Lorem ipsum dolor sit amet, sit scelerisque arcu, auctor nam aliquid lorem donec euismod faucibus, suscipit vitae eros. Sollicitudin luctus donec et, congue nec pede, ut fusce molestie nec integer.	2018-06-28 00:00:00	299	🗑️ ✎️

NORMALIZATION

The term normalization is often used within the field of databases. It deals with the process of organizing the data in a database. This concerns creation of tables, setting correct data types and establishing relationships between tables to avoid redundancy and inconsistent dependency. If the database is not normalized and several tables are not updated or deleted correctly due to missing relationships, the database can grow out of hand and become very slow. Also, there is a chance that some tables will get updated and some tables that should be related won't get updated. This can lead to incorrect data which can cause very fatal errors in the system. If the database is normalized these errors can be avoided and the database will run faster.

There are three levels of normalization: "First Normal Form", "Second Normal Form" and "Third Normal Form". The "First Normal Form" is the lowest level and the "Third Normal Form" is the highest level of normalization.

First Normal Form

- Eliminate repeating groups in individual tables
- Create a separate table for each set of related data
- Identify each set of related data with a primary key

Second Normal Form

- Create separate tables for sets of values that apply to multiple records
- Relate these tables with a foreign key

Third Normal Form

- Eliminate fields that do not depend on the key.

I strived to normalize my database to the "Third Normal Form" which is the highest level. I did that by organizing all my data in an "entity relationship diagram" and dividing the data into the correct tables and look up tables with the appropriate datatypes. When doing my entity relationship diagram, I had three main rules in mind: No repeated data, updates only take place in one place and no empty values. If data were either repeated or there could be an empty value it should always be done as a look up table with a foreign key linked to the primary from the main table.

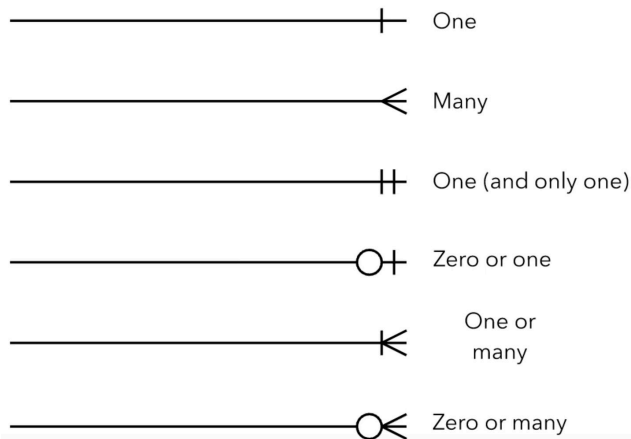
Another thing I did to normalize my database was to index the lookup tables if the data were repeated many times. An example of this, is that every product in my system belongs to a certain "product line". Each product line has several products and if it were to be stored directly in each product, the product line would be repeated many times. Instead I chose to index the product lines in a lookup table and then connect the two tables by a primary and foreign key, so each product would just have the id of the product line and then the name of it could be looked up. This makes the database faster and it will also take up less space.

STRUCTURE & DATATYPES

Entity Relational Diagram

An entity relationship diagram is a diagram that can help organize and determine relationships between tables. When building an entity relationship diagram there are several things to take into account: there is the different tables, the determination of appropriate datatypes and last the cardinality.

The cardinality covers how the tables are connected and what type of relationship they have. There are different types of relationships, it can for example be one-to-one, many-to-one or many-to-many. The cardinality is illustrated by drawing lines with different endings (type of relationship) between the tables. These are the different types of cardinality:



By using the cardinalities properly, it will help you and others to understand the data structure and how each table is related. If it is just a small project like this it might not seem as important but for larger and much more complex databases it is essential to keep the overview.

Datatypes

When setting up the entity relationship diagram and later on the actual database it is fundamental to select the correct datatypes. One of the most important datatypes is “serial”. Each table needs a unique id which is used to identify each row, so when it has to be either updated, deleted or selected you point to the unique id. For that it is ideal to use the datatype “serial”.

“serial” is equivalent to the datatype “bigint(20)” if it is set auto increment, unique and unsigned. Auto increment means that the number will automatically be increased, so none of the ids will be identical. Since none of the ids can be the same it should be set to unique. At last it should be set to the attribute unsigned which implies that the number cannot have any signs therefore it can't be a negative id. Instead of selecting all these things yourself you can just chose the datatype “serial” and then these settings are selected for you.

Some of the most common datatypes which are almost always used in a database are “varchar” and “int”. “varchar” is the datatype used for text and “int” stands for integer and is used for numbers. When using “varchar” you need to define how many characters is allowed to enter. This is written in parentheses after varchar like this “varchar(20)”.

When it comes to the datatype “int” you can also specify how many digits are allowed, but within the datatype “int” you also have the option to choose “tinyint”, “smallint”, “mediumint” and “bigint” which have predefined sizes.

When setting up the database it always helps to be as precise as possible because it will make the database faster and more efficient. The more the database knows about the data the better it will perform.

Another important datatype to remember is “varbinary”. When I set up my database I usually select the collation utf8_general_ci which means that my database will be case insensitive. For most of the data, case is insignificant, but for a password to an account for instance it is essential that it is case sensitive, because the password can contain both upper and lowercase letters. The datatype “varbinary” is therefore ideal for a password because it makes the data case sensitive.

At last I used two more datatypes for my project which is “real” and “timestamp”. “real” is the datatype used for money, so obviously I used that for the prices on my webshop. “timestamp” is a datatype used for dates/time as the name of it implies. I used the timestamp for the orders on my shop to log when the user added the product to their cart.

Primary and Foreign keys

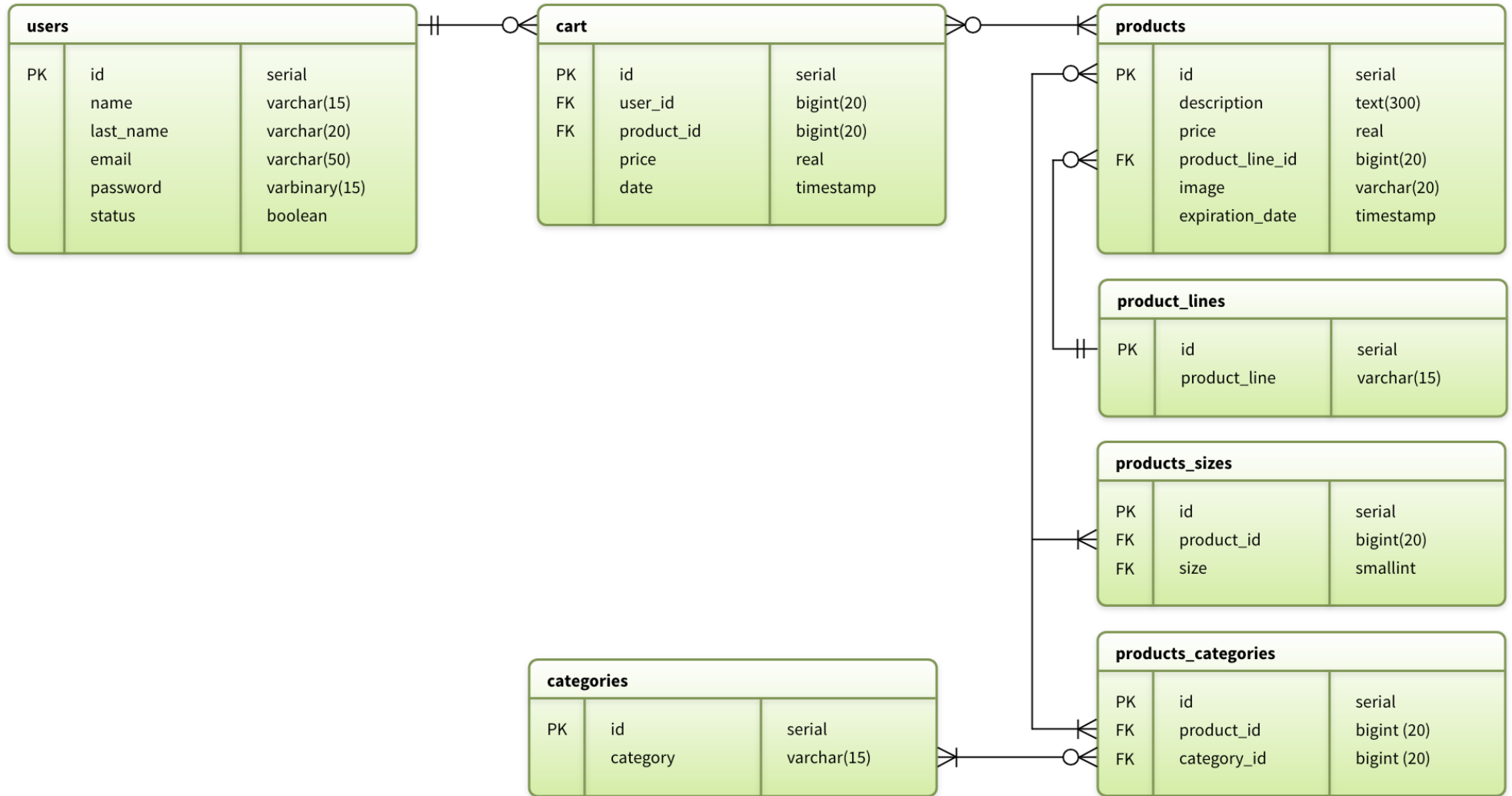
Primary and foreign keys are also an important factor when setting up the entity relationship diagram. A primary key has to be unique and indexed, so therefore it is often the id of the row. A table can only have one primary key, that can be used as a reference to other tables where it is then called a Foreign key.

A foreign key can appear many times and is not unique. A good example of when to use a foreign key is if a user has three phone numbers. According to the rules of normalization you cannot place all three phone numbers in the main table, you have to put them in a lookup table. To connect the main table and the lookup table you use the primary key from them main table as a foreign key in the lookup table. For each phone number you will have the foreign key and phone number. It is important to set the foreign key to be indexed, because it will make the queries run faster.

Indexing is used to make the search for data faster. If a table is indexed by id it will generate a list of all the places where that id appears. When a user is selected the data from all the places where that specific user id appears will be collected much faster, because the index will point to the positions of where the data is located.

ER DIAGRAM

Entity Relationship Diagram



DATABASES

Text files & MongoDB

Text files and document-based databases as MongoDB are very commonly used for storing data. They are non-relational databases which imply that data cannot be linked together. As mentioned both MongoDB and text-files are based on objects and the data is therefore not organized in tables but with JSON.

The advantages of using either text files or document-based databases are that it is much faster than for example MySQL, and it is also very easy to insert data, because the rules of normalization do not apply for these types of databases. There is no real structure for the database and basically you have the ability to insert anything into your object, and you don't have to think about repeated data or null values. Another advantage of using text files or document-based databases is that you don't have to think about datatypes and it is very fast and easy to setup the database.

Text files and document-based databases work very well for smaller amounts of data. The disadvantage of using this type of database is that it does not handle complex data very well. Due to the fact that they are non-relational databases and transactions are not yet supported it will be a challenge to use for larger amounts, and more complex data. In that case it will be necessary to have a more efficient structure and indexing, so it can be more manageable for the people administrating it.

MySQL

MySQL is a relational database management system which is therefore based on tables instead of objects. SQL is a type of programming language that stands for "Structured Query Language". SQL is widely used amongst many database management systems like MySQL, PostgreSQL and Ora-

cle. The MySQL database is managed via the graphical user interface (GUI) "phpMyAdmin" which is where the whole database is set up and can be edited.

There are several advantages of using database management system like MySQL. The first and most important one is that it can handle relational data. In "phpMyAdmin" there is the option to connect tables as you wish based on a Primary and Foreign key. If you choose "Delete on cascade" the relational data will automatically be deleted if the main element is deleted which will simplify the queries and make the database more efficient.

Another advantage is that it is built to handle large amounts of data and structure it in a significantly better way than object-based databases. The rules of normalization do not apply for object-based databases but it does for relational databases. This means that tables should not have null values or repeated data. If the database is normalized it will be much faster, so even though it in general is a bit slower than object-based databases it might be more efficient in the end if there is a lot of data that needs structure and a lot of repeated data.

Sqlite

SQLite is also a relational database management system like MySQL, but there are some differences. In MySQL the data is structured in tables and the tables can be connected with a Primary and Foreign key so that the database can be normalized.

The big difference between MySQL and SQLite is that MySQL strives to implement a shared repository of enterprise data where SQLite strives to provide local data storage for individual applications and devices. SQLite is widely used all over the web and it works especially good for web applications.

CHOICE OF DATABASES

As the project description states we have to choose two types of databases for our project. I chose to use a text file which is object-based and MySQL which is relational. I chose to combine those two databases because some of my data is very simple and some of it is more complex, therefore a text file was suitable for one part and MySQL was more suitable for the other part.

As explained earlier in the report the concept of my project is a web shop with users, products and orders. The users in my system have relatively little data and only one relationship to the orders, that links the products and users together and the products have a lot of relationship to its categories, sizes and product line which makes the structure a bit more complex. Therefore, I chose to store the users in a text file and the products in MySQL. If it weren't for the fact that I could not get MongoDB to work with php on my computer (people with mac had a lot of issues), I would have used that instead of a text file. Since the concept of MongoDB and text files are very similar I chose to use the text files because it was closest equivalent option. Because I was not able to use MongoDB I will explain later in the report how I could have done some queries if I had used it instead of text files.

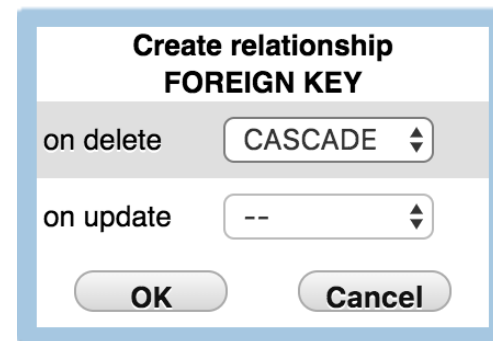
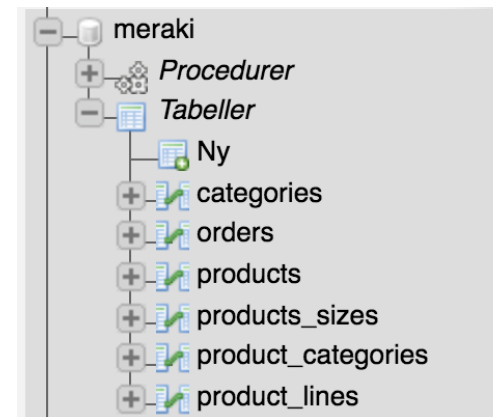
EXECUTION & COMMANDS MYSQL

During this project I have been using a lot of SQL commands to get data out of the database. In the next couple of chapters I will explain how the database was setup and some of the code and commands that I have been using. Due to the fact that I was not able to get MongoDB to work with php only in the terminal, I will explain some of the relevant commands and how I set it up.

Setting up database in MySQL

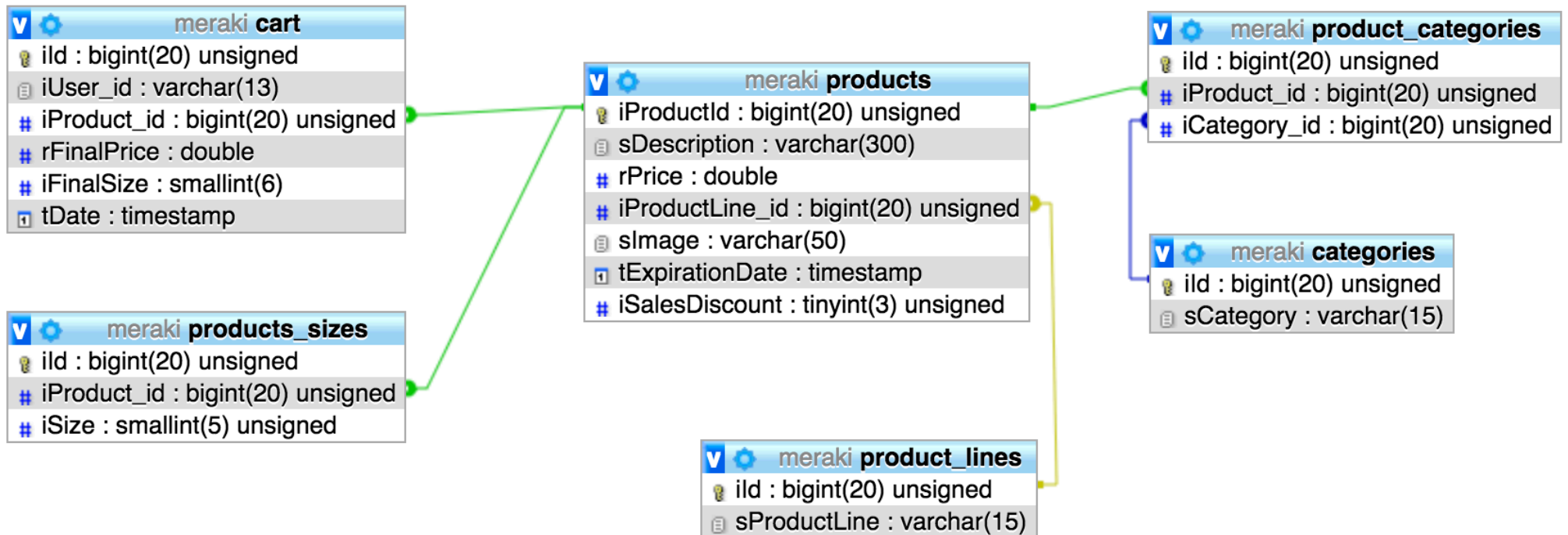
I started out enabling the MySQL database via the program “XAMPP”. Then I went in to “phpmyadmin” and created a new database called “meraki” with the collation utf8_general_ci. Looking at my entity relationship diagram I started creating the different tables and columns with the correct datatypes.

When all tables were created I went into more>designer and started making the relationships between the tables. Before these relationships could be made, it was important to check that all foreign keys were set to the exact same datatype as the primary key and that all foreign keys were indexed. When creating the relationships I chose the constraint “on delete: cascade” which means that if a product is deleted all the information about that products sizes, category and so on will also be deleted as well. The tables and relationships in MySQL can be seen on the next page.



ER DIAGRAM IN MYSQL

Entity Relationship Diagram



CRUD

As any other database MySQL is based on four basic commands that can create, read, update and delete. These are the most common commands and all of them are used several times throughout my project, some in a very simple way and some in a more complex way.

Create

The first basic command is how to create. To create something in the database the SQL command “insert” is used. The example below is from my project. It show what happens when a user adds a product to their cart. The following values: id, user id, product id, price, size and a timestamp are inserted into the cart table.

The user id and product id is what links the two together. The reason why the price is inserted is because a product can be on sale and therefore the current price needs to be saved. A product can come in several sizes and therefore the size chosen has to be saved as well. At last I save a timetamps as well so I can see when the product was added to the cart.

```
try{
  $stmt = $db->prepare("INSERT INTO cart VALUES (null, :iUserId, :iId, :iPrice, :iSize, CURRENT_TIMESTAMP)");
  $stmt->bindValue(':iUserId', $iUserId);
  $stmt->bindValue(':iId', $iId);
  $stmt->bindValue(':iPrice', $iPrice);
  $stmt->bindValue(':iSize', $iSize);

  $stmt->execute();
  echo 'succes';

}catch( PDOException $ex ){
  echo 'EXCEPTION';
  exit();
}
```

Read

To read from the database the SQL command “select” is used. The example in the next column is when I have to get all the categories for the dashboard, so the administrator will be able to see them also if a new one is added. The little star (*) means “all” and categories is the name of the table.

After the command is executed I use the function fetchAll() to save all the categories in an array. Afterwards a “foreach loop” can be used to print out all the categories. If you are only expecting one element the funtion fetch() can be used instead.

```
try{
  $sRead = $db->prepare( "SELECT * FROM categories" );
  $sRead->execute();
  $aCategories = $sRead->fetchAll();

}catch( PDOException $ex ){
  echo 'EXCEPTION';
  exit();
}
```

The products in my system has a lot of relational data and to retrieve that data the different tables needs to be joined before they can be read from. To join two tables the command INNER JOIN can be used. You also need to specify on what parameter the two tables should be joined, in my case I joined them on the primary and foreign key.

```
// Check if the id of the product is passed via get
if( !isset($_GET['id']) ){
  $iProductId = $_GET['id'];

  // get single product from database
  try{
    $sRead = $db->prepare( "SELECT * FROM products
    INNER JOIN product_categories ON products.iProductId = product_categories.iProduct_id
    INNER JOIN categories ON product_categories.iCategory_id = categories.iId
    INNER JOIN product_lines ON products.iProductLine_id = product_lines.iId
    INNER JOIN products_sizes ON products.iProductId = products_sizes.iProduct_id
    AND products.iProductId = $iProductId" );

    $sRead->execute();
    $aProduct = $sRead->fetch();

  }catch( PDOException $ex ){
    echo 'EXCEPTION';
    exit();
  }
}
```

Update

To update something in the database the SQL command “update” is used. The example below is from when the category of a product is updated. The update consists of three keys: UPDATE, SET and WHERE. After UPDATE you insert the name of the table you want to update. Then you set the new value, in this case the new category id. At last you specify where. In this example I use the product id to specify which product I want to update.

```
try {
    // Update product category
    $stmt = $db->prepare("UPDATE product_categories
                        SET iCategory_id = :iCategory_id
                        WHERE product_categories.iProduct_id = $$sProductId");
    $stmt->bindValue(':iCategory_id', $sCategoryId);
} catch(PDOException $ex){
    echo 'error' . __LINE__;
    $db->rollBack();
}
```

Delete

To delete from the database the SQL command “delete” is used. The example below is also from my project and show when a product has to be deleted. The command is very simple the first key DELETE FROM is followed by the name of the table you want to delete from and the key WHERE specifies which product should be deleted by using the product id.

```
try{
    $stmt = $db->prepare( "DELETE FROM products WHERE products.iProductId = $iId" );
    $stmt->execute();

    header( 'Location: products.php' );

} catch( PDOException $ex ){
    echo 'EXCEPTION';
    exit();
}
```

PREPARED STATEMENTS

An important procedure within MySQL is prepared statements with bound parameters. Prepared statements work as a template for the query with empty placeholders as parameters. These placeholders are later filled in by using the function `bindValue()`. After the values are bound the statement gets executed.

There are several advantages of using prepared statements. The most important one is that it minimize the risk of SQL injections. It will also reduce parsing time and make the database more efficient. On the right side is an example from my project where I insert a new product line into the database using a prepared statement and binding the value.

```
<?php

require_once('database.php');

if( isset($_POST['txtProductLine']) ){

    $sProductLine= $_POST['txtProductLine'];

    try{
        $stmt = $db->prepare( "INSERT INTO product_lines VALUES (null, :sProductLine )" );
        $stmt->bindValue(':sProductLine', $sProductLine);

        $stmt->execute();

        header( 'Location: dashboard.php' );

    }catch( PDOException $ex ){
        echo 'EXCEPTION';
        exit();
    }

} else {
    echo 'error';
}
```


TRANSACTIONS

Transactions are used if two or more relational tables needs to be affected. The reason for using transactions is to avoid invalid or non-synchronized data. If we look at the products in my database they have a lot of relationships to other tables. The products table is related to the product lines, sizes and categories. If a product needs to be updated, the product line, size and category of the product will also be updated and therefore several tables are affected.

Let's say that the update was done without a transaction and during this update something goes wrong when updating the size but everything else goes fine, then the product will be updated except the size. After this update the data for that product will be incorrect because the size was not updated. This can be very fatal for the system and cause a lot of issues, therefore it is vital to use transactions. Transactions can be divided in to three steps:

- **beginTransaction()** will start the transaction and therefore it needs to be in the top.
- **rollback()** is used when something goes wrong in the try/catch. If something goes wrong the rollback will make sure that none of the statements are executed, not even the ones that are successful.
- **commit()** makes sure to save (commit) the data into the database if everything is successfully executed. Therefore, it needs to be in the end after all statements.

To the right is an example from my project. This is after the transaction begun and the main product was inserted. If that statement was executed it goes on and tries to insert the category, and if that is executed it goes on to the next statement that will insert the size. As you can see the size is the last statement and therefore if that statement is executed it will commit or else it will rollback.

```

if($stmt->execute()){

    // Get product id of last insert
    $iLastProductId = $db->lastInsertId();

    try {
        // Insert product category
        $stmt = $db->prepare('INSERT INTO product_categories
            VALUES(null, :iProduct_id, :iCategory_id)');
        $stmt->bindValue(':iProduct_id', $iLastProductId);
        $stmt->bindValue(':iCategory_id', $sCategoryId);

        if($stmt->execute()){

            // in case only one size is passed
            if( empty($_POST['txtSize1']) || empty($_POST['txtSize2']) ) {

                $iSize = !empty($_POST['txtSize1']) ? $_POST['txtSize1'] : $_POST['txtSize2'];
                try {
                    $stmt = $db->prepare('INSERT INTO products_sizes
                        VALUES(null, :iProduct_id, :iSize)');
                    $stmt->bindValue(':iProduct_id', $iLastProductId);
                    $stmt->bindValue(':iSize', $iSize);

                    if($stmt->execute()){
                        $db->commit();
                    }else{
                        $db->rollback();
                    }
                } catch( PDOException $ex){
                    echo 'error'.__LINE__;
                    $db->rollback();
                }
            }
        }
    }
}

```

SHOP FILTERS

In the shop page the users have the ability to filter, sort and search for products. They can sort by price from high-to-low or low-to-high. They can also filter by either product category or product line. To create these options, I pass variables via the method GET with the user's request. If the variable is set I insert the value into an SQL statement.

The filters are made as an extra constraint for the query that only selects the products with either a specific product line or category. The sorting is created by using the command ORDER BY and then it is either set to ASC or DESC depending on the user's choice.

At last the search bar is made using LIKE and wildcards. As you can see in the example from my project the user can search within the product line and categories (more can always be added like description for example). The search keyword is saved in a variable and inserted in the statement between two wildcards. By putting wildcards on both sides, the statement will check if either the product line or category contains the search keyword.

```
// check if variable 'productLine' is passed via get
if ( isset($_GET['productLine']) ){
    $productLine = $_GET['productLine'];
    $productLineFilter = "AND products.iProductLine_id = $productLine";
}

// check if variable 'category' is passed via get
if ( isset($_GET['category']) ){
    $category = $_GET['category'];
    $categoryFilter = "AND product_categories.iCategory_id = $category";
}

// check if variable 'orderByPrice' is passed via get
if ( isset($_GET['orderByPrice']) ){
    $order = $_GET['orderByPrice'];
    $orderByPrice = "ORDER BY products.rPrice $order";
}

// check if variable 'search' is passed via get
if ( isset($_GET['search']) ){
    $search = $_GET['search'];
    $searchResult = "AND (product_lines.sProductLine LIKE '%$search%'
                    OR categories.sCategory LIKE '%$search%')";
}
```

VIEWS

A VIEW is a virtual table that points to information from other tables but doesn't store anything in the actual database. VIEWS can contain data from multiple tables using the principle of JOIN. The advantages of using VIEWS are that you can select the exact information that is needed from the tables you wish, hide complicated queries and reuse the information from the VIEW as many times as desired. It is very easy to fetch the information from the VIEW via php you just need to select all from it.

In the example on the right side, I created a VIEW where all the products on sale is displayed. The query is relatively long because I need to JOIN several tables. The screenshot in the bottom displays how easy it is to fetch the information via php.

```
CREATE VIEW products_on_sale AS
SELECT products.iProductId, products.rPrice, products.sImage, products.iSalesDiscount,
product_lines.sProductLine, categories.sCategory
FROM products, product_lines, product_categories, categories
WHERE products.iProductLine_id = product_lines.iId
AND products.iProductId = product_categories.iProduct_id
AND product_categories.iCategory_id = categories.iId
AND products.iSalesDiscount > 0
```

iProductId	rPrice	sImage	iSalesDiscount	sProductLine	sCategory
17	249	5af037cbe60fe.jpg	25	Cotton Haze	Shampoo
61	249	5af9f5e100735.jpg	20	Silky Mist	Shampoo

```
try{
    $sRead = $db->prepare( "SELECT * FROM products_on_sale" );
    $sRead->execute();
    $aProducts = $sRead->fetchAll();

}catch( PDOException $ex ){
    echo 'EXCEPTION';
    exit();
}
```

STORED PROCEDURES

Stored procedures are queries that are saved in the database. The advantage of using stored procedures is that you can save a long query and then just call them as functions in php. If you need a query several places it is more efficient to save it as a stored procedure because you will save space and avoid repeated code.

To create a stored procedure, you can go into phpmyadmin under the tab SQL and write the command. You need to start and end the query with DELIMITER so it is aware that you want to create a function. This is followed by the command CREATE PROCEDURE and then the name of the procedure. At last you type BEGIN and END and insert your query between those. Now the query is created and you can now call it from php by using the command CALL followed by the name of the procedure.

```
DELIMITER $$
CREATE PROCEDURE getAllProducts
BEGIN
    SELECT * FROM products
    INNER JOIN product_categories
    INNER JOIN categories
    INNER JOIN product_lines
    INNER JOIN products_sizes
    WHERE products.iProductId = product_categories.iProduct_id
    AND product_categories.iCategory_id = categories.iId
    AND products.iProductLine_id = product_lines.iId
    AND products.iProductId = products_sizes.iProduct_id
    GROUP BY products.iProductId;
END $$
DELIMITER ;
```

```
// Get all products

try{
    $sRead = $db->prepare( "CALL getAllProducts()" );
    $sRead->execute();
    $aProducts = $sRead->fetchAll();

    }catch( PDOException $ex ){
        echo 'EXCEPTION';
        exit();
    }

?>
```

TRIGGERS

A trigger is a type of stored procedure associated with a specific table. The trigger gets executed on a certain action for example on insert, update or delete. You can use triggers for many things, in my project I chose to make a trigger that capitalizes the first letter in the product description.

If the users of my system were stored in the database I would have chosen to make a trigger on that table instead to for example capitalize first letter in the name and lowercase the whole email. You can never know if a user tries to write capital letters in their email or write their whole name with lower cases, so in this situation it would have made even more sense.

To illustrate the concept, I created one trigger on my products table. You start the trigger with the command CREATE TRIGGER followed by the name of it. Then you decide the trigger time, event and on which table. In my case the trigger happens before insert on the table products for each row inserted. At last you insert your SQL statement between BEGIN and END. It's important to write "new" because it should affect the new value that is about to be inserted.

```
CREATE TRIGGER capitalizeFirstLetter
BEFORE INSERT ON products
FOR EACH ROW
BEGIN
SET new.sDescription = CONCAT(UCASE(LEFT(new.sDescription, 1)), SUBSTRING(new.sDescription, 2));
END;
```

EXECUTION & COMMANDS MONGODB

Install MongoDB

I installed version 3.6.3 of MongoDB. To install it I used a package manager for Mac OS called “Home Brew”. Home Brew has a large software library and therefore it made it possible to install MongoDB via the terminal. After installing MongoDB, I downloaded Compass and installed that as well. Compass is the user interface for the client.

Starting Mongo

Mongo has two main files “Mongod” which is the server and “Mongo” which is the client. First, I needed to start the server. To start the server, I used the command “mongod” in the terminal. The “d” in “mongod” stands for “daemon” which means that the program will keep running in the background. To connect to the client (Compass) I used the command “mongo” in the terminal. To test if the connection was established correctly I used the command “show dbs” which shows all the databases.

```
> show dbs
admin    0.000GB
config  0.000GB
kea      0.000GB
local   0.000GB
>
```

Create database

To create a new database, I opened Compass and clicked on the plus icon in the left sidebar. A window pops up and here the database and collection name can be filled in. The collection name is similar to a table name in MySQL. I created a small test project called “kea” just to show the concept of how MongoDB works.

Create Database

Database Name

Collection Name

Capped Collection ⓘ

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

CANCEL CREATE DATABASE

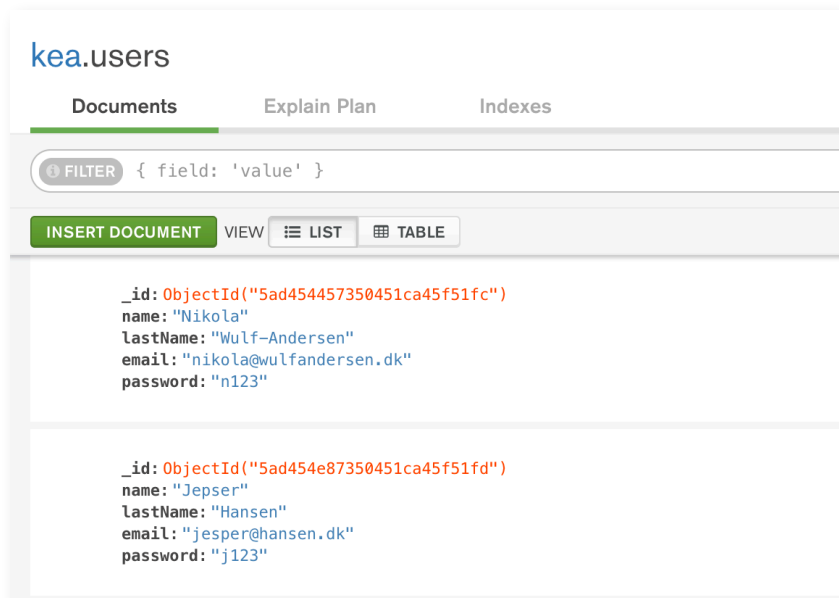
To get into the database that I just created I used the command “use kea”.

```
> use kea
switched to db kea
>
```

CRUD

Create

There are two options to create a new object. You can either create it via compass or via the terminal. To create a new object via compass you have to click “New document” and just insert the object in the window that pops up.



To insert the user via the terminal this command can be used:

```
db.users.insert({"name":"Nikola", "lastName":"Wulf-Andersen"});
```

- **db** stands for the database and since “kea” previously was selected as the database it will be inserted into that
- **users** is the name of the collection that I want to target
- **insert** is the build in function used to create a new object in this case a user. Inside the “insert” function the JSON object is written

It is also possible to use either *insertOne()* or *insertMany()* to create new objects. *insertMany()* takes an array as the argument and looks like this:

```

try {
  db.products.insertMany( [
    { item: "card", qty: 15 },
    { item: "envelope", qty: 20 },
    { item: "stamps" , qty: 30 }
  ] );
} catch (e) {
  print (e);
}
    
```


Read

To read from the database several commands can be used. You have the option to select all users in the database with this command:

```
db.users.find();
```

There is also the option to use either `db.users.findOne({})` or `db.users.findMany({})`. If you are expecting only one object to be returned it is more efficient to use `db.users.findOne({})`. This will stop the search as soon as there is a match and therefore make the search process much faster, similar to when you use “break” when looping through JSON objects in a text file or `LIMIT: 1` in MySQL.

It is also possible to make the search for objects more specific like setting parameters. This command will only return the users that has “31584084” as their phone number:

```
db.users.find({"phones": " 31584084"});
```

And this command will return all users that has either “31584084” or “34667123” as their phone number:

```
db.users.find({"phones": {$all: [" 31584084", " 34667123"]}});
```

Update

To update a document it is the built in function `update()` and the operator `$set` that is used. The following command will update just one document:

```
db.users.updateOne({"name": "Nikola"}, $set: {"name": "Niko"});
```

As well as in MySQL you use “set” to determine the new value for the update. This will only affect one document (object) and the key or keys you choose to target within that document. If you want to update two documents or more at the same time this command can be used:

```
db.users.updateMany({"name": "Nikola"}, $set: {"name": "Niko"});
```

Delete

When it comes to deleting it is divided into two parts: if you want to delete the whole document or just one or some of the keys in the document. To delete the whole document this command can be used:

```
db.users.delete({"name":"Nikola"});
```

This query will delete all document with where the value of the key “name” is set to “Nikola”. To delete only one document this command will be more suitable:

```
db.users.deleteOne({"name":"Nikola"});
```

When it comes to deleting one or more keys in a document it is actually the build in function update() and the operator \$unset that is used. This is also one of the similarities between MongoDB and text files. When operating with text files it is the method unset() that is used to delete a key from an object. To delete the key “name” from all document in MongoDB this command can be used:

```
db.users.update({}, $unset:{"name":true},{multi:true})
```

It is essential that it is set to “multi:true” so it updates all documents, otherwise it will just update the first one. To delete a key only from one specific document the following command can be used:

```
db.users.updateOne({"name":"Nikola"}, $unset:{"name":true})
```

REFLECTION

This project has been a huge learning process for me. I started out knowing almost nothing about databases and now I believe I have gained insight and a broad understanding of the subject. I still have some considerations and things that I might have done differently in a future project.

My ER Diagram states that one product can have many categories but looking at the actual project all the products only have one category. Initially when I created the ER Diagram I thought that each product would have multiple categories, but when I set up the web shop and inserted data, I realised that it wasn't necessary.

The reason I chose not to change it was because I kept on thinking about it, and for now one category is sufficient, but if this were a real web shop the number of products would keep on increasing and then in the future there would probably be a need for parent- and subcategories to organize the products better.

As I learned during my research you should always try to create a database that does not only handle the data you have now but also what will come in the future so you don't have to alter the database too many times. This is an important principal that I will take with me for next time I have to develop a database.

A thing that I would have liked to have worked even more with, if I had had more time, was to make my project even more realistic. For now, the user has to login to start shopping. In a real web shop the user would probably have the choice to login or shop without an account and then the cart items would be saved in a session instead of in the actual database.

In the user's cart is a button "Proceed to checkout" but when the button is clicked nothing happens because I didn't implement a payment solution (wasn't really relevant for databases). If this part of the project had been

implemented there would also be a need for some extra tables like "orders" and "orders_items" so the purchase could be finalised.

Another thing is how the users are saved. Instead of saving them in separate databases I would have saved the whole database in MySQL. There are a lot of small things in this project that I would have done differently if it were an actual customer project. But since the important part of this project was to demonstrate how to manage databases and different commands, not everything in this project will seem as the optimal solution, because that was not my first priority.

CONCLUSION

In general, I am satisfied with my project and I have reached the goals I set out for it in the beginning. I have learned how to manage two different types of databases, normalize it and execute a variety of different commands. Furthermore, I have learned how to manage the GUI for MySQL "phpmyadmin" and how to create, triggers, stored procedures, views and many other things.

During the process of creating this database project I have acquired a lot of new knowledge, so when I look back on the execution of it now, I know that some things could have been done in an even more efficient way. Some small parts have already been corrected as I got a better understanding of the subject and some I learned from for future projects. By combining a strong theoretical part with a practical project, I feel like I have accomplished the best result possible.

LITERATURE

The MongoDB 3.6 Manual — MongoDB Manual 3.6. 2018. The MongoDB 3.6 Manual — MongoDB Manual 3.6. [ONLINE] Available at: <https://docs.mongodb.com/manual/>. [Accessed 25 April 2018].

Install MongoDB Community Edition on macOS — MongoDB Manual 3.6. 2018. Install MongoDB Community Edition on macOS — MongoDB Manual 3.6. [ONLINE] Available at: <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/#install-mongodb-community-edition-with-homebrew>. [Accessed 25 April 2018].

YouTube. 2018. Entity Relationship Diagram (ERD) Tutorial - Part 1 - YouTube. [ONLINE] Available at: <https://www.youtube.com/watch?v=QpdhBUYk7Kk>. [Accessed 30 April 2018].

Essential SQL. 2018. Database Normalization Explained in Simple English - Essential SQL . [ONLINE] Available at: <https://www.essentialsql.com/get-ready-to-learn-sql-database-normalization-explained-in-simple-english/>. [Accessed 7 May 2018].

Microsoft. 2018. Microsoft Support. [ONLINE] Available at: <https://support.microsoft.com/en-gb/help/283878/description-of-the-database-normalization-basics>. [Accessed 7 May 2018].

PHP: MySQL Database. 2018. PHP: MySQL Database. [ONLINE] Available at: https://www.w3schools.com/php/php_mysql_intro.asp. [Accessed 8 May 2018].

Appropriate Uses For SQLite. 2018. Appropriate Uses For SQLite. [ONLINE] Available at: <https://www.sqlite.org/whentouse.html>. [Accessed 9 May 2018].

MySQL - Wikipedia. 2018. MySQL - Wikipedia. [ONLINE] Available at: <https://en.wikipedia.org/wiki/MySQL>. [Accessed 9 May 2018].

Entity Relationship Diagram - Everything You Need to Know About ER Diagrams . 2018. Entity Relationship Diagram - Everything You Need to Know About ER Diagrams . [ONLINE] Available at: <https://www.smartdraw.com/entity-relationship-diagram/>. [Accessed 10 May 2018].

Difference between Primary Key and Foreign Key. 2018. Difference between Primary Key and Foreign Key. [ONLINE] Available at: <https://www.dotnettricks.com/learn/sqlserver/difference-between-primary-key-and-foreign-key>. [Accessed 11 May 2018].

PHP Prepared Statements. 2018. PHP Prepared Statements. [ONLINE] Available at: https://www.w3schools.com/php/php_mysql_prepared_statements.asp. [Accessed 12 May 2018].

MySQL Tutorial. 2018. Create Trigger in MySQL. [ONLINE] Available at: <http://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx>. [Accessed 17 May 2018].

SiteGround Knowledge Base. 2018. What are MySQL triggers and how to use them?. [ONLINE] Available at: <https://www.siteground.com/kb/mysql-triggers-use/>. [Accessed 17 May 2018].

Essential SQL. 2018. What is a Relational Database View? - Essential SQL . [ONLINE] Available at: <https://www.essentialsql.com/what-is-a-relational-database-view/>. [Accessed 18 May 2018].