*Written by Jonas Lagoni (92412820)*

# Database Project

Software Engineering 2016-2021
Syddansk Universitet - TEK
DM505 – spring 2017
Project Assignment

## Introduction

This report is a reflection on the application and database created for DM505 spring 2017 project assignment. This report contains description on E/R model with the corresponding dependencies and explanations on why it does not violate 3NF and the insight into the application developed to accommodate the database. Attributes, relations and keys are styled the following way: Keys will have <u>underline</u>. Attributes will have *cursive*. Relations will have **bold text**.

## Table of contents

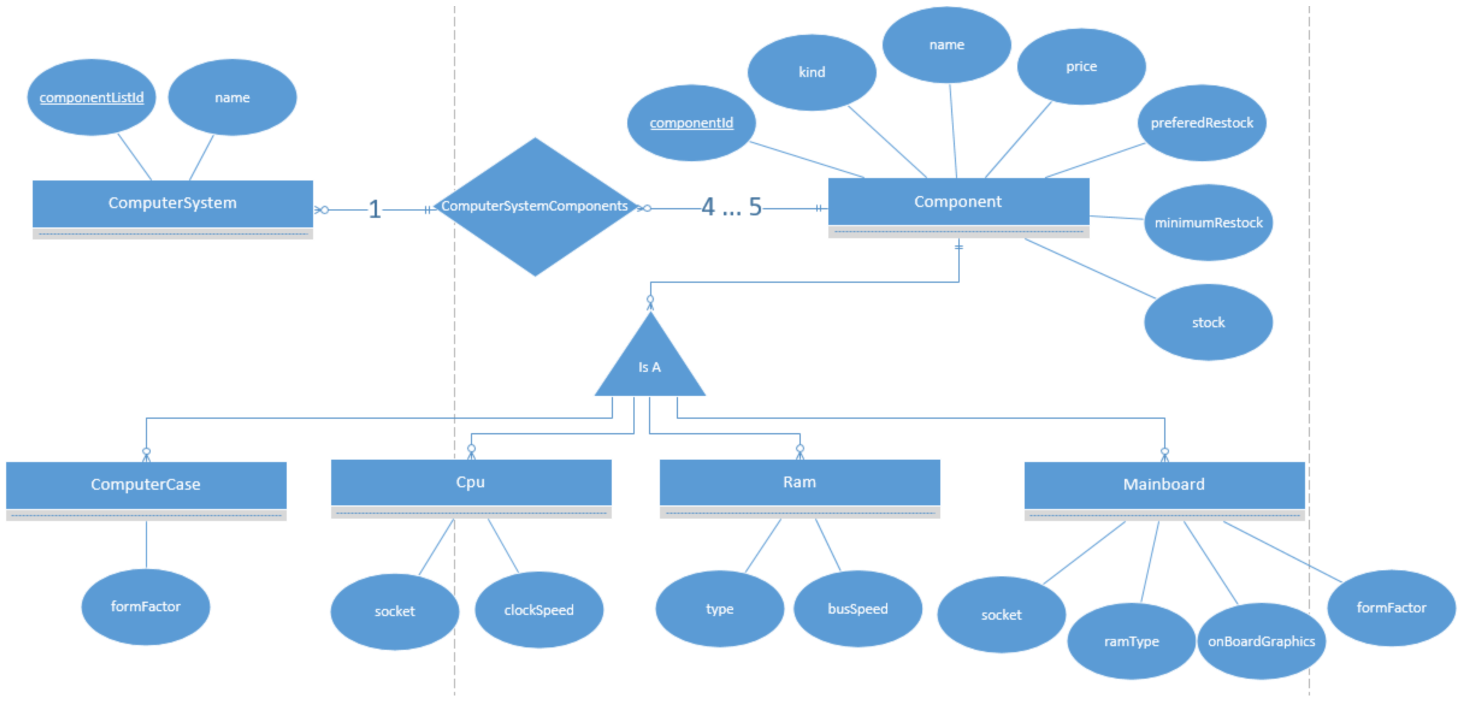
# 1 The database

In the handout, assignment there was giving the following objects, which should be included in the database[1]. A component should contain *name, kind, price* and an inventory option, which means the first relation we find, is **component**. There were two ways the inventory option could be added, onto a relation itself or added directly to the component as attributes. In this solution, the second option is used and therefore the *stock, preferedRestock* and *minimumRestock* is added directly to **component** as attributes. Switching to the other option would be from the performance point of view be better if updating components is essential. *Kind* would represent "cpu", "ram", "gpu" (Graphics card), "mainboard" and "case" accordingly. Each of these kinds have extra information connected to them except the gpu. Therefor each of these will be having their own relation with their corresponding attributes. The **cpu** should contain *socket* and busSpeed but in this case busSpeed will be renamed to *clockspeed* since it is more appropriate since it is a cpu. **Ram** contains *ramType* and *busSpeed,* where *ramType* could be DDR2, DDR3 or even DDR4. **Case** contains formFactor, this could be ATX, M-ATX or even L-ATX. **Mainboard** contains *socket, ramType, onBoardGraphics* and *formFactor*.

There should also be included a system with a name and containing 4-5 component depending on if the **mainboard** have on board graphics. On top of that, the **cpu**, **ram** and **case** should all fit with the correct **mainboard**. The application insures this so no constraints are added to the database regarding this function.

## 1.1 E/R model

The below E/R model represent each functionality described above. This is the first and only E/R model created for database.



*E/R model for the database*

[1] Project assignment: http://www.imada.sdu.dk/~jbaumbac/download/teaching/ss17/DM505/project/project.pdf

## 1.2 Relation model

From the E/R model we determine the following relation models.

**Component**(<u>*componentId*</u>*, name, kind, price, preferedRestock, minimumRestock, stock*)
**Cpu**(<u>*componentId*</u>*, socket, busSpeed*)
**Ram**(<u>*componentId*</u>*, type, busSpeed*)
**Mainboard**(<u>*componentId*</u>*, socket, ramType, onBoardGraphics, formFactor*)
**ComputerCase**(<u>*componentId*</u>*, formFactor*)
**ComputerSystem**(<u>*componentListId*</u>*, name*)
**ComputerSystemComponents**(<u>*componentId*</u>*,* <u>*componentListId*</u>)

The *preferedRestock, minimumRestock* and *stock* attributes in **Component** don't have their own relation because each **Component** need a *preferedRestock, minimumRestock* and a *stock* attribute If the relation **Component** becomes large enough, one would properly create a new relation with the stock options, to make the updating easier for the database

## 1.3 Functional dependencies

Each of the dependencies are split into their corresponding relations including an explanation about why none of the relations violate 3NF and the SQL code used to create these relations.

### 1.3.1 Component

<u>*componentId*</u> → *name, kind, price, preferedRestock, minimumRestock, stock*

{<u>*componentId*</u>} is a superkey and a key because it determine all the other attributes in the relation. This dependency does not violate 3NF because there are only one dependency and the left side of that dependency is a key. *Name* is not a key, because we do not believe that manufacturers know how to name their products correctly. Therefor duplicates may occur.

**Component** is created the following way in postgres:

```
CREATE TABLE Component(
    componentId SERIAL,
    name VARCHAR(100) NOT NULL,
    kind VARCHAR(15) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    preferedRestock INT NOT NULL,
    minimumRestock INT NOT NULL,
    stock INT NOT NULL,
    CHECK (price >= 0),
    CHECK (stock >= 0),
    CHECK (preferedRestock >= 0),
    CHECK (minimumRestock >= 0),
    PRIMARY KEY (componentId)
);
```

*SQL query for creating table* ***Component***

Here <u>*componentId*</u> is an auto increment of type int. Which means it will automatically generate the next int in line so no duplicates can occur. The rest of the attributes are self-explanatory. We also create some constraints for the attributes, `NOT NULL` meaning none of the attributes may be null, and the CHECK insure that neither *price, preferedRestock, minimumRestock, stock* can be under 0 since no such thing should be able to happen. In the end, we add *componentId* as the primary key.

### 1.3.2 **CPU**

*componentId* → socket, *busSpeed*

{*componentId*} is a superkey and a key because it determine all the other attributes in the relation. This dependency does not violate 3NF because there are only 1 dependency and the left side of that dependency is a key.

**Cpu** is created the following way in postgres:

```
CREATE TABLE Cpu(
   componentId INT,
   socket VARCHAR(30) NOT NULL,
   clockspeed DECIMAL(3,1) NOT NULL,
   CHECK (clockspeed  >= 0),
    FOREIGN KEY (componentId)
    REFERENCES Component (componentId)
    ON DELETE CASCADE
 );
```

*SQL query for creating table **Cpu***

Here each attribute is self explanatory, for *busSpeed* it makes a check to insure that it is always 0 or higher. In the end, we add the foreign key constraint on {*componentId*} to refer to the {*componentId*} key in **Component**. There is also applied an ON DELETE constraint to the foreign key to make sure to delete the tuple in **Cpu** if deleted in **Component**.

### 1.3.3 **Ram**

*componentId* → *type, busSpeed*

{*componentId*} is a superkey and a key because it determine all the other attributes in the relation. This dependency does not violate 3NF because there are only one dependency and the left side of that dependency is a key.

**Ram** is created the following way in postgres:

```
CREATE TABLE Ram(
   componentId INT,
   ramType VARCHAR(20) NOT NULL,
   busspeed REAL NOT NULL,
   CHECK (busspeed >= 0),
   FOREIGN KEY (componentId)
   REFERENCES Component (componentId)
   ON DELETE CASCADE
);
```

*SQL query for creating table **Ram***

Here everything have already been explained in the 1.4.2

### 1.3.4 **Mainboard**

*componentId* → *socket, ramType, onBoardGraphics, formFactor*

{*componentId*} is a superkey and a key because it determine all the other attributes in the relation. This dependency does not violate 3NF because there are only one dependency and the left side of that dependency is a key.

**Mainboard** is created the following way in postgres:

```
CREATE TABLE Mainboard(
   componentId INT,
   socket VARCHAR(30) NOT NULL,
   ramType VARCHAR(20) NOT NULL,
   onBoardGraphics BOOLEAN NOT NULL,
   formFactor VARCHAR(10) NOT NULL,
   FOREIGN KEY (componentId)
   REFERENCES Component (componentId)
   ON DELETE CASCADE
);
```

*SQL query for creating table* ***Mainboard***

In this query everything important have already been explained previously.

### 1.3.5 **ComputerCase**

*componentId* → *formFactor*

{*componentId*} is a superkey and a key because it determine all the other attributes in the relation. This dependency does not violate 3NF because there are only one dependency and the left side of that dependency is a key.

**ComputerCase** is created the following way in postgres:

```
CREATE TABLE ComputerCase(

   componentId INT,
   formFactor VARCHAR(10) NOT NULL,

   FOREIGN KEY (componentId)
   REFERENCES Component (componentId)
   ON DELETE CASCADE
);
```

*SQL query for creating table* ***ComputerCase***

In this above query, everything important have already been explained previously.

### 1.3.6 **ComputerSystem**

*componentListId* → *name*

{*componentListId*} is a superkey and a key because it determine all the other attributes in the relation. This dependency does not violate 3NF because there are only one dependency and the left side of that dependency is a key. *name* is not a key, because I do not believe people can keep making up new names, therefore duplicates may occur.

**ComputerSystem** is created the following way in postgres:

```
CREATE TABLE ComputerSystem(
   componentListId SERIAL,
   name VARCHAR(100) NOT NULL,
   PRIMARY KEY (componentListId)
);
```

*SQL query for creating table **ComputerSystem***

Here *componentListId* is an auto increment of type int. Which means it will automatically generate the next integer in line so no duplicates can occur. In the end, we add *componentListId* as the primary key.

### 1.3.7 ComputerSystemComponents

*componentId*, *componentListId* → *componentId*, *componentListId*

{*componentId*, *componentListId*} is superkey and a key because neither {componentId} nor {componentListId} is a superkey. This dependency does not violate 3NF because there are only one dependencies and the left side of that dependency is keys.

**ComputerSystemComponents** is created the following way in postgres:

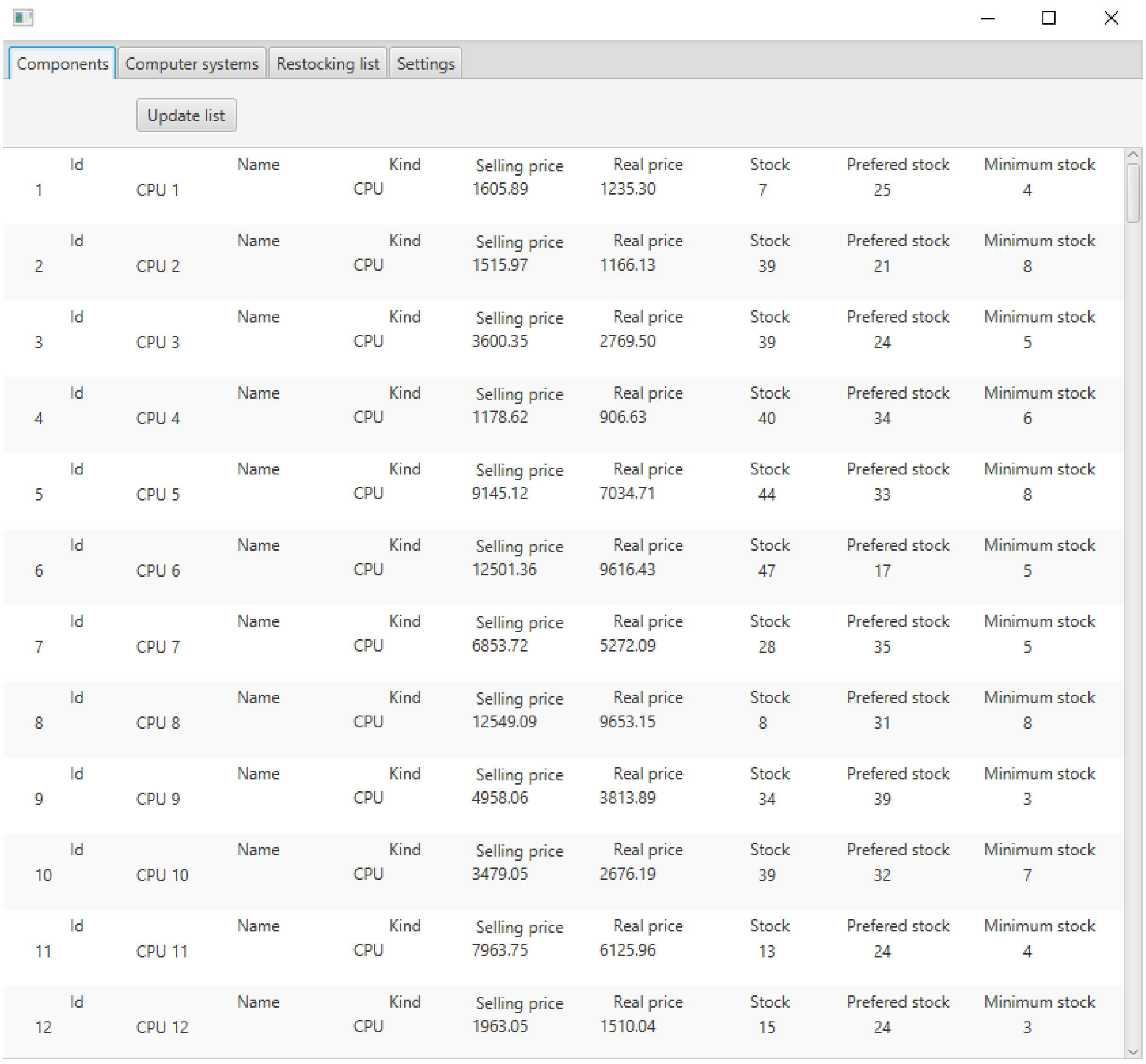
```
CREATE TABLE ComputerSystemComponents(
   componentId INT,
   componentListId INT,
   FOREIGN KEY (componentId)
   REFERENCES Component (componentId)
   ON DELETE CASCADE,
   FOREIGN KEY (componentListId)
   REFERENCES ComputerSystem (componentListId)
   ON DELETE CASCADE
);
```

*SQL query for creating table **ComputerSystemComponents***

In this query, we define the *componentListId* and *componentId* as foreign keys to their corresponding primary keys from their corresponding relations.

## 2 The application

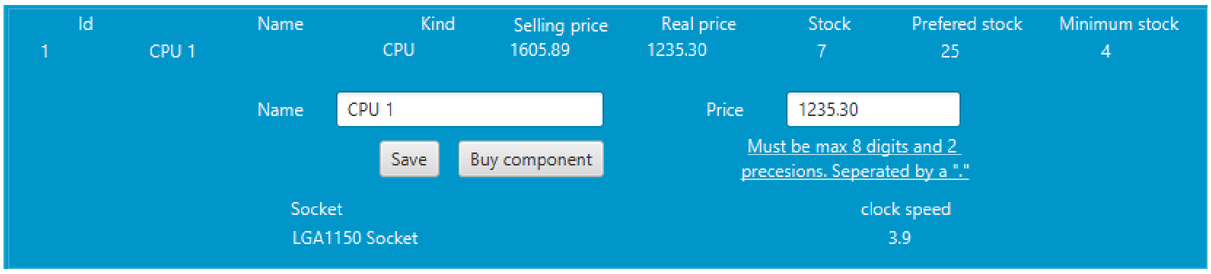
The application is developed in Java, as stated in the project description, and JavaFX platform architecture for controlling the appearance with FXML and CSS. The application is using a single JavaFX thread, which means that the application will wait for the queries to run. This means that when doing many instructions the application will freeze and wait for those instructions to finish. Therefor there is a limit to how much data application can hold. As a stress test with 50.000 components, the application ran out of memory at 8 GB ram. So this application is not optimized to handle large amount of data, so please bare that in mind if using the create data function under settings.



| Id | Name | Kind | Selling price | Real price | Stock | Prefered stock | Minimum stock |
|---|---|---|---|---|---|---|---|
| 1 | CPU 1 | CPU | 1605.89 | 1235.30 | 7 | 25 | 4 |
| 2 | CPU 2 | CPU | 1515.97 | 1166.13 | 39 | 21 | 8 |
| 3 | CPU 3 | CPU | 3600.35 | 2769.50 | 39 | 24 | 5 |
| 4 | CPU 4 | CPU | 1178.62 | 906.63 | 40 | 34 | 6 |
| 5 | CPU 5 | CPU | 9145.12 | 7034.71 | 44 | 33 | 8 |
| 6 | CPU 6 | CPU | 12501.36 | 9616.43 | 47 | 17 | 5 |
| 7 | CPU 7 | CPU | 6853.72 | 5272.09 | 28 | 35 | 5 |
| 8 | CPU 8 | CPU | 12549.09 | 9653.15 | 8 | 31 | 8 |
| 9 | CPU 9 | CPU | 4958.06 | 3813.89 | 34 | 39 | 3 |
| 10 | CPU 10 | CPU | 3479.05 | 2676.19 | 39 | 32 | 7 |
| 11 | CPU 11 | CPU | 7963.75 | 6125.96 | 13 | 24 | 4 |
| 12 | CPU 12 | CPU | 1963.05 | 1510.04 | 15 | 24 | 3 |

*Picture 1. - Displaying the application with the main component tab open.*

As picture 1. shows, each component is displayed with all the information relevant for a component. If the component is clicked on, the advanced options and extra information is shown as shown in picture 2. It is then possible to buy the component and edit the name or price. Bear in mind that the only price accepted are at the displayed format 8 numbers and two decimal precision.



| Id | Name | Kind | Selling price | Real price | Stock | Prefered stock | Minimum stock |
|---|---|---|---|---|---|---|---|
| 1 | CPU 1 | CPU | 1605.89 | 1235.30 | 7 | 25 | 4 |

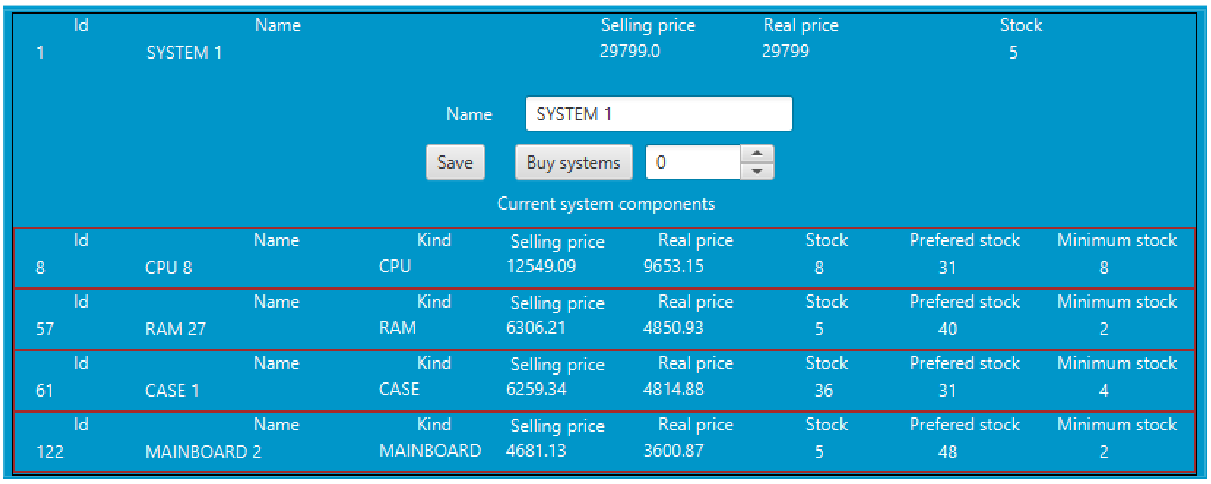
Name: CPU 1

Price: 1235.30

| Socket | clock speed |
|---|---|
| LGA1150 Socket | 3.9 |

*Picture 2. - Displaying* the advanced options and extra information for a component

The selling price is, as stated in the project description, increased by 30 % from the original price shown in “Real price”. The component can also never have a lower stock than zero and if the stock is lower than the minimum stock, then the component is shown in the restocking list.



| Id | Name | Selling price | Real price | Stock |
|---|---|---|---|---|
| 1 | SYSTEM 1 | 29799.0 | 29799 | 5 |

Name: SYSTEM 1

| Id | Name | Kind | Selling price | Real price | Stock | Prefered stock | Minimum stock |
|---|---|---|---|---|---|---|---|
| 8 | CPU 8 | CPU | 12549.09 | 9653.15 | 8 | 31 | 8 |
| 57 | RAM 27 | RAM | 6306.21 | 4850.93 | 5 | 40 | 2 |
| 61 | CASE 1 | CASE | 6259.34 | 4814.88 | 36 | 31 | 4 |
| 122 | MAINBOARD 2 | MAINBOARD | 4681.13 | 3600.87 | 5 | 48 | 2 |

*Picture 3. - Displaying* a single system with its components

When displaying the system each of the regular component functions is still available by clicking on the desired component. On top of that, it is possible to buy multiple systems and even change the system name. When buying multiple systems the following formula are used when you want to buy more than 10 systems:

*round((((((int)Math.round(totalPrice*1.3))+99) / 100)*100-1)*0.8)*numberOfSystems, 2)*

Moreover, the following formula for below 10 systems:

*round((((((int)Math.round(totalPrice*1.3))+99) / 100)*100-1)*(1.0-((numberOfSystems*2.0)/100.0)))*numberOfSystems, 2)*

This insures that the system price is the total real price of each component with the extra 30 % and then reduced by 2% for each system. In the end, the price is rounded down to two decimals.

## 2.1 Database transactions

The application uses the JDBC driver for connecting to the postgres database. The following SQL queries are used to retrieve information from the database. The below query joins the relations **cpu**, **ram**, **mainboard** and **computercase** so the view retrieved contains all the necessary informations about the components so no further queries are required. The query also orders the view by the component id to be ascending. Which means the first result will be the one with the lowest id. To make sure null values are not used the *kind* attribute is used.

```
SELECT
component.componentId,
component.name,
component.kind,
component.price,
component.preferedrestock,
component.minimumrestock,
component.stock,
cpu.socket,
cpu.clockspeed,
ram.ramtype,
ram.busspeed,
mainboard.socket as mainboardSocket,
mainboard.ramtype as mainboardRamType,
mainboard.onboardgraphics,
mainboard.formfactor as mainboardFormFactor,
computercase.formfactor
FROM component
LEFT JOIN cpu ON cpu.componentId = component.componentId
LEFT JOIN ram ON ram.componentId = component.componentId
LEFT JOIN mainboard ON mainboard.componentId = component.componentId
LEFT JOIN computercase ON computercase.componentId =
component.componentId
ORDER BY component.componentId ASC;
```

*SQL query used in the application for retrieving all the components.*

The below query retrieves all the components related to a certain system, in this case the system with the id equals 1.

```
SELECT
component.componentId,
component.name,
component.kind,
component.price,
component.preferedrestock,
component.minimumrestock,
component.stock,
cpu.socket,
cpu.clockspeed,
ram.ramtype,
ram.busspeed,
mainboard.socket as mainboardSocket,
mainboard.ramtype as mainboardRamType,
```

```sql
mainboard.onboardgraphics,
mainboard.formfactor as mainboardFormFactor,
computercase.formfactor
FROM computersystemcomponents
LEFT JOIN component ON component.componentId =
computersystemcomponents.componentId
LEFT JOIN cpu ON cpu.componentId = component.componentId
LEFT JOIN ram ON ram.componentId = component.componentId
LEFT JOIN mainboard ON mainboard.componentId = component.componentId
LEFT JOIN computercase ON computercase.componentId =
component.componentId
WHERE computersystemcomponents.componentListId = 1
ORDER BY component.componentId ASC;
```

*SQL query used in the application for retrieving all the components for the system, and in this case with the id 1.*

For the restocking list, the bellowed query is used to find each component who have a lower *stock* then their corresponding *minimumrestock* value.

```sql
SELECT
component.componentId,
component.name,
component.kind,
component.price,
component.preferedrestock,
component.minimumrestock,
component.stock,
cpu.socket,
cpu.clockspeed,
ram.ramtype,
ram.busspeed,
mainboard.socket as mainboardSocket,
mainboard.ramtype as mainboardRamType,
mainboard.onboardgraphics,
mainboard.formfactor as mainboardFormFactor,
computercase.formfactor
FROM component
LEFT JOIN cpu ON cpu.componentId = component.componentId
LEFT JOIN ram ON ram.componentId = component.componentId
LEFT JOIN mainboard ON mainboard.componentId = component.componentId
LEFT JOIN computercase ON computercase.componentId =
component.componentId
WHERE component.stock < component.minimumrestock
ORDER BY component.componentId ASC;
```

*SQL query used in the application for retrieving all the components, which need a restock. Meaning their stock is lower than their minimum stock.*

## 2.2 User manual

Since the procedure used to test this project is somewhat vague, some extra files have been added under the folder "Extra". The file DatabaseNetbeansProject.jar is the executable version of the application which can be run directly without doing anything other than running it. The Netbeans project is also added for the sole purpose of making it easier to open and review the code. The "Simple sql for relation and value creation.sql" file contains the pure SQL, ready to be added to an existing database, the connection details can then be edited in the application itself to accommodate the correct details.

The rest of this user manual covers each aspect of the application. It will be displayed as a list of options where each option may lead to more functionality.

### 2.2.1 The component list

The component list is used to view every component in the database. These are the functionalities:

- View every component in the database
- Force an update by pressing the "Update list" button, which updates the list.
- Click on a component to view more information and to access the following functionalities:
    - Buy a component
    - Save component information
        - Make sure price is corresponding to the correct format
          Example: 12345678,90

### 2.2.2 The computer system list

Is used to view every computer system in the database. These are the functionalities:

- View every system in the database
- Force an update by pressing the "Update list" button, which updates the list.
- Click on a system to view more information.
    - Change the system name
    - Buy 0 or more systems
        - The selling price will update with the correct price for all the systems
    - Click on a component to view more information and to access the following functionalities:
        - Buy a component
        - Save component information
            - Make sure price is corresponding to the correct format
              Example: 12345678,90
            - When the price is changed for the component the system price will also be updated.

### 2.2.3 The restocking list

Is used to view every component listed for restocking, meaning the stock value is lower than the minimumInventory. These are the functionalities:

- View every component needed for a restock.
- Force an update by pressing the "Update list" button, which updates the list.
- Click on "Restock all components" to restock every component in the list.
- Click on a component to view more information and to access the following functionalities:

  - Buy a component
  - Save component information
    - Make sure price is corresponding to the correct format
      Example: 12345678,90

### 2.2.3 The settings tab

Here it is possible to change the database settings and reset the database with some new values. Be aware that creating over 1k components and or systems might give some performance issues.

## Conclusion

The project was overall a solid assignment, but is missing difficulty for people who have some experience in databases. I would have found it more interesting to be giving an existing (bad) E/R model, which then needed to be rewritten. The E/R model is practically already given from the objects.