Database project

Software Engineering 2016-2021

Syddansk Universitet - TEK

DM505 – spring 2017

Project Assignment

# Introduction

Attributes, relations and keys are styled the following way:

* Keys will have underline.
* Attributes will have *cursive*.
* Relations will have **bold text**.

# Table of contents

[1 The database 4](#_Toc479193619)

[1.1 E/R model 5](#_Toc479193620)

[1.2 Relation model 5](#_Toc479193621)

[1.3 Functional dependencies 5](#_Toc479193622)

[1.3.1 **Component** 5](#_Toc479193623)

[1.3.2 **CPU** 6](#_Toc479193624)

[1.3.3 **Ram** 7](#_Toc479193625)

[1.3.4 **Mainboard** 7](#_Toc479193626)

[1.3.5 **ComputerCase** 8](#_Toc479193627)

[1.3.6 **ComputerSystem** 8](#_Toc479193628)

[1.3.7 **ComputerSystemComponents** 9](#_Toc479193629)

[The application 10](#_Toc479193630)

# 

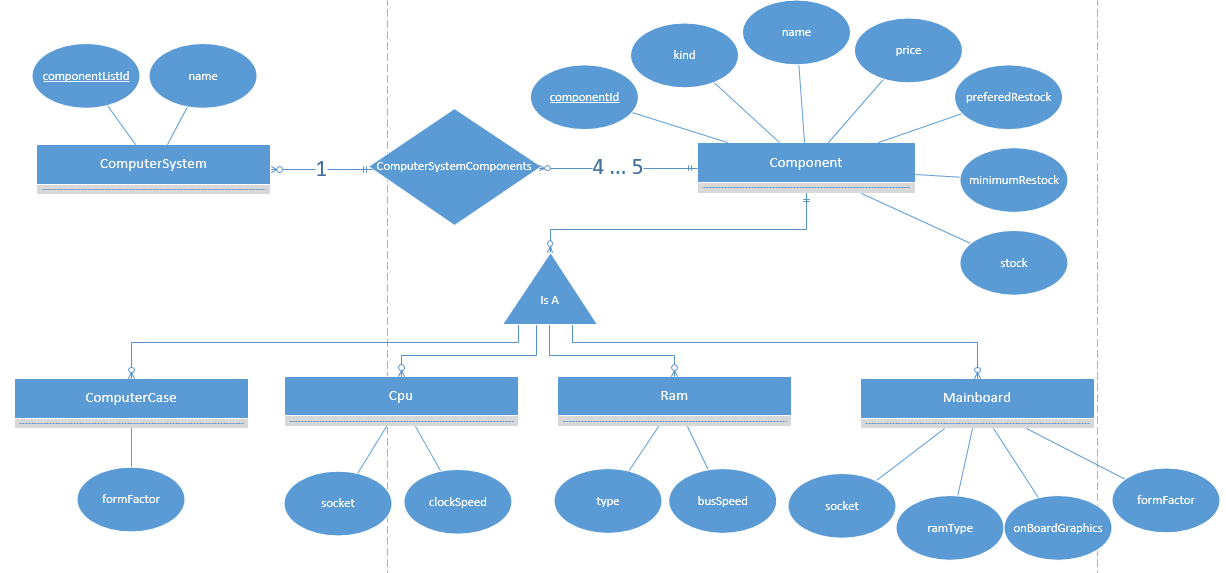
# 

# 1 The database

In the handout, assignment there was giving the following objects, which should be included in the database[[1]](#footnote-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. These checks are insured by the system before created so no constraints are added to the database.

## 1.1 E/R model



*E/R model for the database*

## 1.2 Relation model

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

**Component**(*componentId, name, kind, price, preferedRestock, minimumRestock, stock*)

**Cpu**(*componentId, socket, busSpeed*)

**Ram**(*componentId, type, busSpeed*)

**Mainboard**(*componentId, socket, ramType, onBoardGraphics, formFactor*)

**ComputerCase**(*componentId, formFactor*)

**ComputerSystem**(*componentListId, name*)

**ComputerSystemComponents**(*componentId, componentListId*)

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

*componentId* → *name, kind, price, preferedRestock, minimumRestock, stock*

{*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. *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 *componentId* 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 a 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 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.

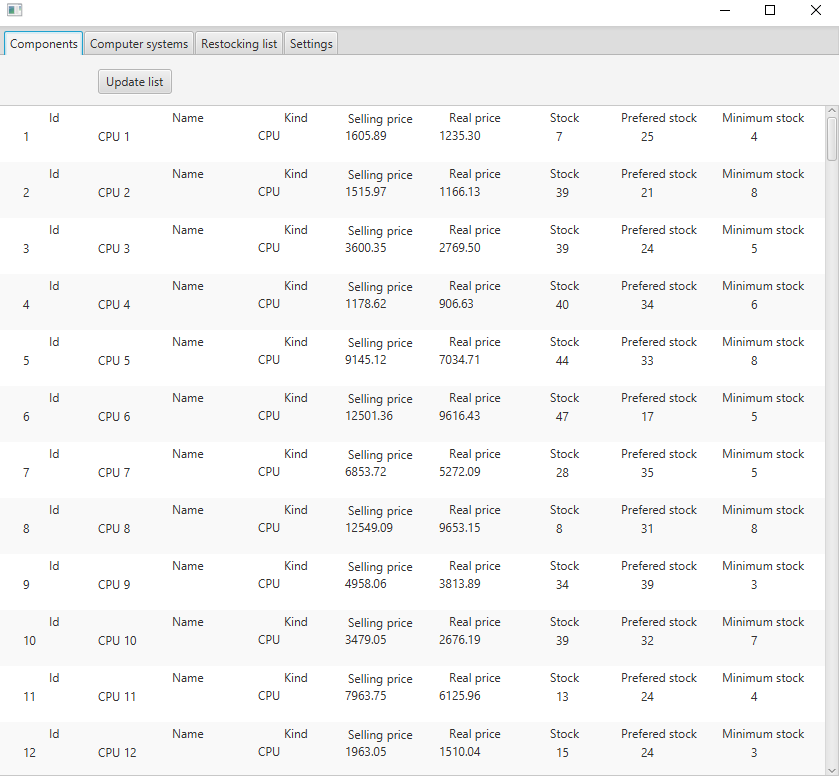
**ComputerSystemComponents** is created the following way in postgres:

|  |
| --- |
| 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***

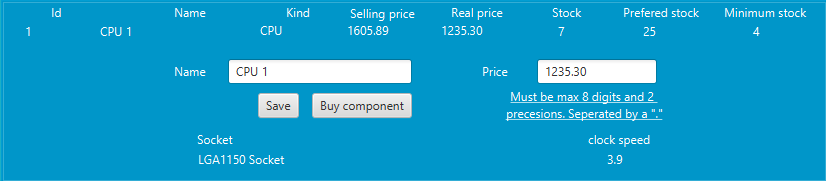
In this query, we define the *componentListId* and *componentId* as foreign keys to their corresponding primary keys from their corresponding relations.

# 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. 

*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. Bare in mind that the only price accepted are at the displayed format 8 numbers and 2 decimal precision.



*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 % compared to the original price shown in “Real price”. The component can also never have a lower stock than zero. If the components stock is lower than the minimum stock, then the component is also shown in the restocking list.

## 2.1 The important parts

The application

## 2.2 User manual

This user manual cover 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

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

* Scroll through every component.
* Force an update by pressing the “Update list” button.
* Click on a component to view more information.
  + 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:

* Scroll through every system.
* Force an update by pressing the “Update list” button.
* 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.
    - 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:

* Scroll through every component.
* Force an update by pressing the “Update list” button.
* Click on “Restock all components” to restock every component in the list.
* Click on a component to view more information.
  + 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

1. Project assignment: <http://www.imada.sdu.dk/~jbaumbac/download/teaching/ss17/DM505/project/project.pdf> [↑](#footnote-ref-1)