Databaseproject: Computerstore

Jeff Gyldenbrand
Username: jegyl16
D.O.B.: 22-11-1984
Supervisor: Jan Baumbach
DM505

6. april 2017

Indhold

1 Specification

This project was to program a system to manage a computer hardware stock. The computer store needed to keep information on all their computer systems and components, by printing a list of everything in stock. Also it should calculate how many computer systems possible to build from current stock.

To manage this, the system needed the following objects to be modeled; Component, given a name, kind and a price. The different kinds had to be one of cpu, ram, graphiccard, mainboard and case. These would make a complete computer system. Furthermore, it was a demand that cpu had the attributes; socket and busspeed. Ram had the attributes of ram-type and busspeed. Mainboard a cpu-socket, ram-type, form-factor and on-board graphics. The idea was that, from these attributes, the system would model only computer systems that would match on sockets, busspeed, ram-type and form-factor.

The system should also be able to print a price list of all the components or systems, and be able to sell them as well by updating the current amount.
And calculate a discount of 2% from each additional sell of computer systems, with a cap of maximum 20% discount.

Finally, the system should print a restock list, which shows which components and how many should be restocked on each Saturday.

To complete this goal, it was necessary to design an E/R model and transfer it to a relational model and implement it in a DBMS (database management system).

2 Design

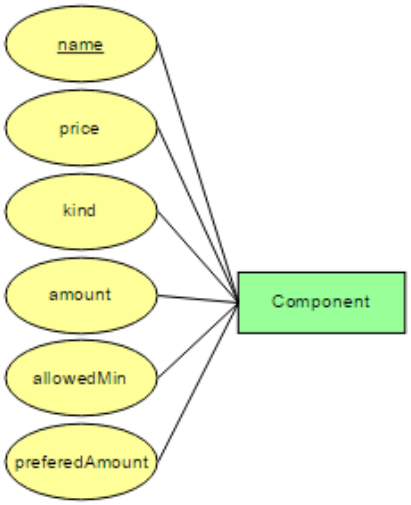
So given the demands from the specification, the first thing we do is to create an entity-relation diagram (E/R). This will give us an overview of the database we need to construct.

We know the different tables should be cpu, graphiccard, mainboard, ram and case. However, we've chosen to include an harddrive and powersupply too. Furthermore, we know that we should somehow model a complete computer system from these tables. Therefore we've modeled the tables computer system and Component to handle this.

To keep track of current stock and a minimum inventory, we need to create to more tables. The reason why we wants to have two separate tables will be explained in the next section.

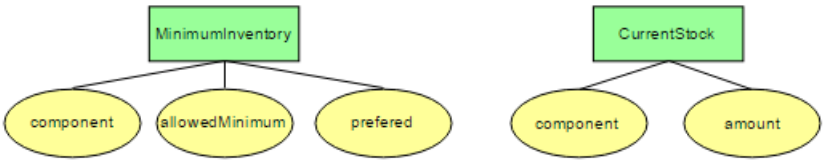
2.1 Design choices and reasons

The first attempt to create a database was actually successfully. From the information in previous sections, we created the tables: "Computer system", "Components", "CPU", "RAM", "GraphicCards", "HDD", "PowerSupply", "CASE_".
The table 'Component' as shown on the right, where given the attributes 'name', 'price', 'kind', 'amount', 'allowedMin' and 'preferedAmount'. However .. Even though it worked, it was later discovered that it broke the conditions for 3NF.



So as recommended in the project-description, we created two new tables: 'MinInventory' and 'CurrentStock' as shown in the picture below. And left the 'Component'-table with only the attributes: 'name', 'price' and 'kind'.

In this project we assume that there wont exist two different components with the same name. Therefor only the name is assigned primary key. If we where to build a larger database with thousands of components for a real company, we would make this differently to uniquely identify the different components.



2.1.1 Database-table contra SQL-function

```
SELECT *
FROM CPU, RAM, Mainboard, CASE_
WHERE (CPU.busspeed = RAM.busspeed AND CPU.socket = Mainboard.CPU_socket AND RAM.RAM_type = Mainboard.RAM_type AND Mainboard.formfactor = CASE_.formfactor)
ORDER BY CASE_
```

Data Output | Explain | Messages | History

name character varying (39)	socket character varyi...	busspeed real	cores integer	chipset character (10)	name character varying (39)	ram_type character varyin...	busspeed real	memorysize integer	name character varying (39)	cpu_socket character varying...	ram_type character va
Intel Core i7-6950X Extreme	LGA2011-v3	3	10	INTEL	HyperX Savage	DDR4	3	65536	MSI X99A RAIDER	LGA2011-v3	DDR4
Intel Core i7-6950X Extreme	LGA2011-v3	3	10	INTEL	Corsair Vengeance LEED	DDR4	3	16384	MSI X99A RAIDER	LGA2011-v3	DDR4
Intel Core i7-6950X Extreme	LGA2011-v3	3	10	INTEL	HyperX Fury	DDR4	3	16384	MSI X99A RAIDER	LGA2011-v3	DDR4
Intel Core i7-6950X Extreme	LGA2011-v3	3	10	INTEL	HyperX Savage	DDR4	3	65536	MSI X99A Gaming Pro Carbon	LGA2011-v3	DDR4
Intel Core i7-6950X Extreme	LGA2011-v3	3	10	INTEL	Corsair Vengeance MAX	DDR4	3	32768	MSI X99A Gaming Pro Carbon	LGA2011-v3	DDR4
Intel Core i7-6950X Extreme	LGA2011-v3	3	10	INTEL	Corsair Vengeance LEED	DDR4	3	16384	MSI X99A Gaming Pro Carbon	LGA2011-v3	DDR4
Intel Core i7-6950X Extreme	LGA2011-v3	3	10	INTEL	HyperX Fury	DDR4	3	16384	MSI X99A Gaming Pro Carbon	LGA2011-v3	DDR4
AMD Ryzen 7 1800X	AM4	3.2	8	AMD	Corsair Vengeance LED	DDR4	3.2	16384	MSI X370 XPower Gaming Titanium	AM4	DDR4
AMD Ryzen 7 1800X	AM4	3.2	8	AMD	Crucial Ballistix Sport	DDR4	3.2	16384	MSI X370 XPower Gaming Titanium	AM4	DDR4
AMD Ryzen 7 1800X	AM4	3.2	8	AMD	Corsair Vengeance LED	DDR4	3.2	16384	Gigabyte Aorus GA-AX370-Gaming 5	AM4	DDR4
AMD Ryzen 7 1800X	AM4	3.2	8	AMD	Crucial Ballistix Sport	DDR4	3.2	16384	Gigabyte Aorus GA-AX370-Gaming 5	AM4	DDR4
Intel Core i5-6600K Skylake	LGA1151	3.5	4	INTEL	Corsair Vengeance LPX	DDR4	3.5	16384	ASUS ROG Strix Z270E Gaming	LGA1151	DDR4
AMD FX-8350 Black Edition	AM3+	4	8	AMD	Kingston DDR3	DDR3	4	16384	ASUS M5A97 R2.0	AM3+	DDR3

Its easy to create a simple SQL-query and print all possible systems where the CPUs busspeed is equal to RAMs busspeed, CPUs socket matches the mainboard socket, and mainboard matches the computer-case. This gives us 101 possible systems from current stock. From this we could also calculate how many of each systems is possible to build from current stock. To limit the output in java command-line, we could restraint to only print 10 systems.

However, this is not the design choice we are going with. Instead we create a new table named 'Computersystem' with the attributes of: "name", "CPU", "RAM", "Mainboard", "HDD", "PowerSupply", "CASE_"

From the above SQL-query we pick 9 tubles, this will be our 9 systems the computer store is going to sell. Now we can also give them some appropriate names to be recognized from. The reason to this design choice is control. We choose that the computer store only wants to sell these 9 particular systems, and therefore it is more easy to manage, if we create a table to maintain the systems. And with this simple SQL-query:

```
SELECT *
FROM Computersystem
```

Data Output | Explain | Messages | History

name character varying (39)	gpu character varying (39)	mainboard character varying (39)	cpu character varying (39)	ram character varying (39)	case_ character varying (39)	hdd character varying (39)
Ultimate Gaming 3K	Sapphire Radeon RX 460	ASUS A68HM-PLUS	AMD A10-7870K Black Edition	Crucial Ballistix Tactical	In Win EM040 Mini Tower Sort	Samsung 850 EVO 250GB 2.5"
Megatron 10000	Sapphire Radeon RX 480	ASUS M5A97 R2.0	AMD FX-8350 Black Edition	Kingston DDR3	Fractal Design - Define C Window Black	Seagate Barracuda 3TB 3.5"
The ultimate computer scientist pc	XFX Radeon RX 480	MSI X370 XPower Gaming Titanium	AMD Ryzen 7 1800X	Corsair Vengeance LED	Corsair Carbide Clear 400C Midi Tower	Samsung 850 EVO 250GB 2.5"
Kommodore upgraded	MSI Radeon RX 470	Gigabyte Aorus GA-AX370-Gaming 5	AMD Ryzen 7 1800X	Crucial Ballistix Sport	Corsair Graphite 760T	Samsung 850 EVO 250GB 2.5"
Gaming-10	Gainward GeForce 970	ASUS B150M-PLUS	Intel Core i5-6600 Skylake	HyperX Fury	Cooler Master MasterCase 3 Pro	Seagate Barracuda 3TB 3.5"
Gaming-20	EVGA GeForce GTX 1060	ASUS ROG Strix Z270E Gaming	Intel Core i5-6600K Skylake	Corsair Vengeance LPX	In Win 805 Infinity	Seagate Barracuda 3TB 3.5"
Gaming-30	MSI GeForce GTX 1070	MSI X99A RAIDER	Intel Core i7-6950X Extreme	HyperX Savage	CM Storm Trooper Gaming Big Tower Sort	Seagate Barracuda 1TB 3.5"
Gaming-40	ASUS GeForce GTX 1050	MSI X99A Gaming Pro Carbon	Intel Core i7-6950X Extreme	Corsair Vengeance MAX	Fractal Design - Define C Window Black	Samsung 850 EVO 500GB 2.5"
Gaming-1000 Insane Godlike	Gainward GeForce 970	MSI X99A Godlike Gaming Carbon	Intel Core i7-6950X Extreme	Corsair Vengeance LED	CM Storm Stryker Gaming Big Tower	Samsung 850 EVO 500GB 2.5"

2.2 1NF

As explained in section 2.1 the first attempt was to create one big table. This is normal when starting the normalization. From the project description we know all the attributes we want in our table. We even added some more, to make it more interesting. Now, the table is one big mess, so we need to normalize in 1st normal form:

First thing, we check every cell, and makes sure there is only a single value. Luckily that's the case. But some cells has the value null. This is not good.

Then we check every entries in the column are of the same type. For example, if column 'price' had an value called 'RAM' this would violate the type. Everything checks out fine.

Last thing we do, is to check if our rows are uniquely identified. So we need to set a primary key. attribute 'name'. In our example its easy to see they are all unique. But lets say that column name had two rows with the exact same name. Then there wouldn't be any way to distinguish them. So we need to formalize a second time.

component

name	price	kind	amount	allowedMin	prefered	cores	CPUsocket	CPUbusspeed	CPUchipset	RAMtype	RAMbusspeed	hddSize	GPUchipset	GPUmem
HyperX Fury	1459	RAM	25	5	25	null	null	null	null	DDR4	3.0	null	null	null
Intel Core i5-6600 Skylake	2299	CPU	30	5	30	4	LGA1151	4.2	INTEL	null	null	null	null	null
Corsair Vengeance LED	1337	RAM	25	5	25	null	null	null	null	DDR4	3.2	null	null	null
Seagate Barracuda 1TB	548	HDD	50	5	50	null	null	null	null	null	null	1024	null	null

But first we need to identify all the attributes, which do not depend on the key. For example, the attribute 'cores' dont depend on it. It has several null values, indicating, that we should probably make a new table named 'CPU' containing 'cores'. We can do this for all the attributes, and reasoning toward something more decent. Then we can check if its 2NF.

2.3 2NF

So now we have split this big messy table into multiple smaller ones. All with name as key. This means all the other attributes of the different tables now depends on the key. But we still need to reference all the tables to the component-table. Now we are not violating 2NF because all attributes depends on the key, which references to table component.

component

name	price	kind
HyperX Fury	1459	RAM
Intel Core i5-6600 Skylake	2299	CPU
Corsair Vengeance LED	1337	RAM
Seagate Barracuda 1TB	548	HDD

cpu

name	cores	CPUsocket	CPUbusspeed	CPUchipset
Intel Core i5-6600 Skylake	DDR4	LGA1151	4.2	INTEL

ram

name	RAMtype	RAMbusspeed
Corsair Vengeance LED	DDR4	3.2
HyperX Fury	DDR4	3.0

currentStock

name	amount
HyperX Fury	25
Intel Core i5-6600 Skylake	30
Corsair Vengeance LED	25
Seagate Barracuda 1TB	50

hdd

name	hddSize
Seagate Barracuda 1TB	1024

minInventory

name	allowedMin	prefered
HyperX Fury	5	25
Intel Core i5-6600 Skylake	5	30
Corsair Vengeance LED	5	25
Seagate Barracuda 1TB	5	50

2.4 3NF and final structure

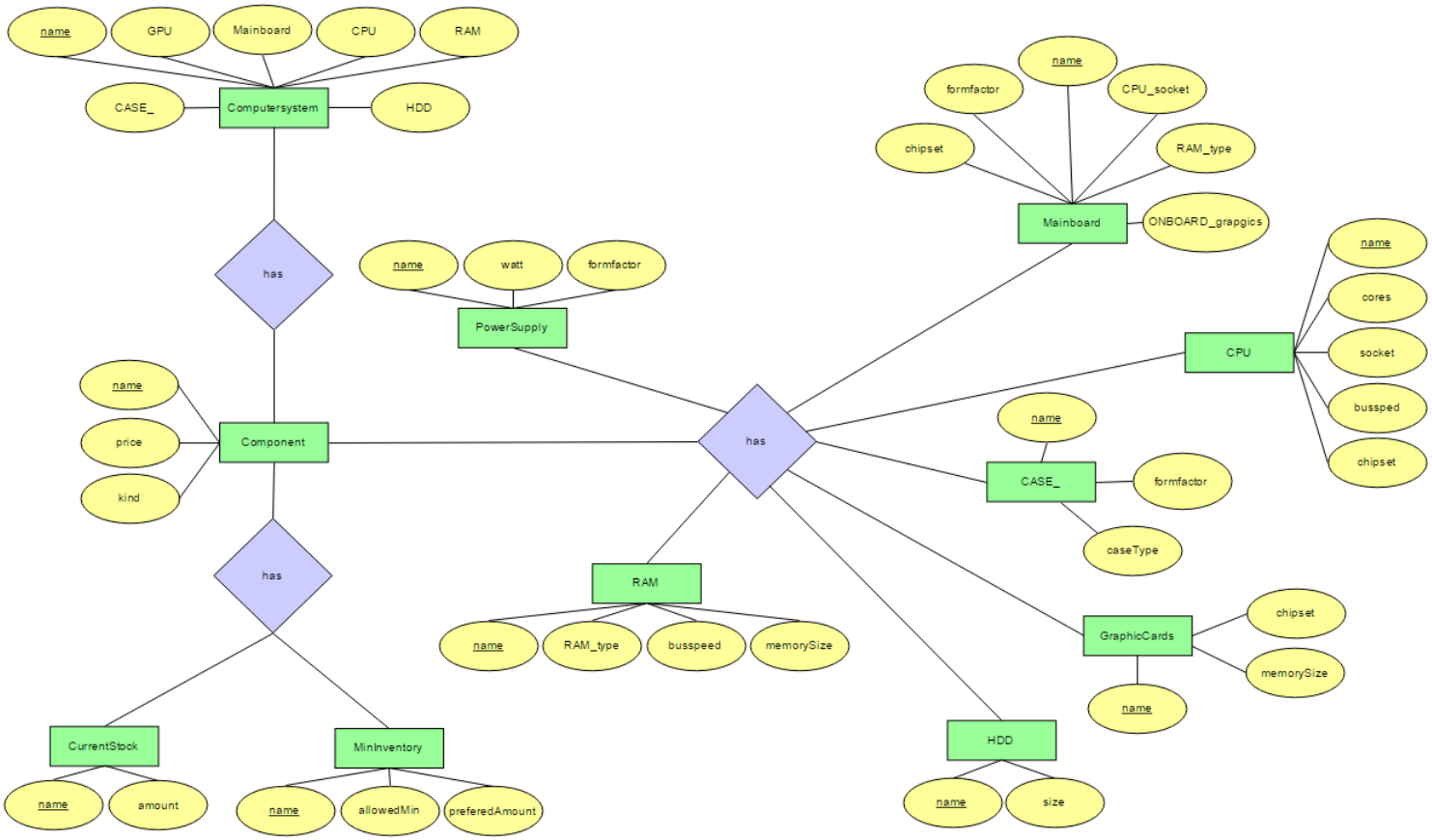
Luckily our tables are all in 3NF too, because all the attributes in the tables can only be determined by the key in their tables. If this was violated, we needed to split them into more tables, until 3NF is obtained. Notice that, the mentioned normalization in this rapport only displays some of the component-types, and the table computer system is not mentioned. The reason to this is to not waste to many pages on explanation of this, when a selected instance would explain the principle behind this database.

Based on the information from section 2.0-2.3, we can now create a relation-schemas:

- **Computersystem**(name, GPU, CPU, RAM, Mainboard, HDD, CASE_)
- **Component**(name, price, kind)
- **MinInventory**(name, allowedMin, preferedAmount)
- **CurrentStock**(name, amount)
- **CPU**(name, cores, socket, busspeed, chipset)
- **GraphicCards**(name, clockspeed, memorySize, chipset)
- **PowerSupply**(name, watt, formfactor)
- **HDD**(name, size)
- **CASE_**(name, formfactor, caseType)
- **Mainboard**(name, CPU_socket, RAM_type, ONBOARD_graphics, formfactor, chipset)

2.5 E/R diagram

Now we can draw our E/R-diagram and see the final structure of the database:



3 Implementation

3.1 Central parts of the SQL-code

This is basically how to get information from the database. Here we extract information on all the components, their name, kind, current amount, allowed minimum and prefered amount. This is done by combining three tables: Component, CurrentStock and MinInventory with the requirement the name of the components are identical.

```
1  public static void All_components(Connection con) throws IOException{
2            try {
3     Statement st = con.createStatement();
4     String query = "SELECT *"
5            + " FROM Component, CurrentStock, MinInventory"
6            + " WHERE (Component.name = CurrentStock.name) "
7            + "AND (Component.name = MinInventory.name)"
8            + " ORDER BY kind";
9
10           ResultSet rs = st.executeQuery(query);
11           System.out.printf("%-39s %3s %22s %18s %10s",
12                  "Component", "Kind", "Current amount",
13                  "Allowed minimum", "Prefered");
14
15           while (rs.next()) {
16                  int amount = rs.getInt("amount");
17                  int allowedMin = rs.getInt("allowedMin");
18                  int prefered = rs.getInt("preferedAmount");
19                  String name = rs.getString("name");
20                  String kind = rs.getString("kind");
21                  System.out.printf("%n %-39s %3s %9d %17d %13d",
22                         name, kind, amount, allowedMin, prefered);
23           }
24           System.out.println("\n\npress [0] to return.\n");
25           IO.returnMenu();
26           IO.input();
27    } catch (SQLException e) {
28    }
29       }
```

See class 'Components.java' for this.

Its the same principle for printing complete computer systems. Here we would simply select everything from the 'Computersystem'-table. However, to calculate how many of each systems could be build from current stock, we need to call a new function that can calculate this. We call it from within the 'ComputerSystems'-method.

```
try {
Statement st = con.createStatement();
String query = "SELECT *"
       + " FROM Computersystem";
ResultSet rs = st.executeQuery(query);

     while (rs.next()) {
         String name = rs.getString("name");
         String gpu = rs.getString("GPU");
         String cpu = rs.getString("CPU");
         String mainboard = rs.getString("Mainboard");
         String ram = rs.getString("RAM");
         String case_ = rs.getString("CASE_");
         String hdd = rs.getString("hdd");
         String line = "|---------------------------";
         System.out.print("| PC: " + name + "\n" + line
                + "\n| Motherboard:  " + mainboard
                + "\n| CPU:          " + cpu
                + "\n| GPU:          " + gpu
                + "\n| RAM:          " + ram
                + "\n| HDD:          " + hdd
                + "\n| CASE:         " + case_);
                System.out.print("\n|\n| From current stock '");
                CalculateNumberOfSystems(con, name, cpu, gpu,
                       ram, hdd, case_, mainboard);
                System.out.print("' system(s) can be build\n\n");
```

Taking the Motherboard, CPU, GPU, RAM, HDD and CASE as argument, and passing to the 'CalculateNumberOfSystems', we can make a new SQL-query for each component in the while-loop.

```
public static void CalculateNumberOfSystems(Connection con, String name,
            String CPU, String GPU, String RAM, String HDD,
            String CASE, String Mainboard) throws IOException {
         try {
           Statement st = con.createStatement();
           String query = "SELECT min(amount)"
                  + " FROM Component, CurrentStock"
                  + " WHERE (Component.name = '" + name + "'"
                  + " OR Component.name = '" + CPU + "'"
                  + " OR Component.name = '" + GPU + "'"
                  + " OR Component.name = '" + Mainboard + "'"
                  + " OR Component.name = '" + RAM + "'"
                  + " OR Component.name = '" + HDD + "'"
                  + " OR Component.name = '" + CASE + "')"
                  + " AND CurrentStock.name = Component.name";
           ResultSet rs = st.executeQuery(query);
```

```
17                  while (rs.next()) {
18                      int count = rs.getInt("min");
19                    System.out.print(count);
```

See class 'ComputerSystems.java' for this.

The above examples are straight forward. And to calculate an price offer on one or more complete computer systems, we just use the same principles. So with a method called 'PriceOffer' retrieving information, we would call a new function, called 'CalculateDiscount', within the while-loop, that calculates the offer:

```
1  while (rs.next()) {
2              // total price for computersystem + 30%
3              double price = rs.getDouble("price");
4              double sellingPrice = price * 1.3;
5
6              // rounded to nearest '99
7              double roundedPrice =
8                     (Math.round(sellingPrice/100) * 100) - 1;
9              if (n == 1) {
10                 System.out.println("\n" + n + "x '" + name + "': with 0%
                       discount:");
11                 System.out.printf("%s%,.2f%s", "Total price: ",
                       roundedPrice, "\n\n");
12                 System.out.println("press [0] to return.\n");
13
14                   IO.returnMenu();
15                   IO.input();
16
17             } else if (n > 1 && n < 11) {
18                 double newPrice =
19                        n * roundedPrice * (1 - ((n-1) * 0.02));
20                 double percentage = (100*(n-1)*0.02);
21
22                 System.out.println("\n" + n + "x '" + name
23                        + "' (with " + (int)percentage
24                        + "% discount):");
25
26                 System.out.printf("%s%,.2f%s", "Total price: ",
27                        newPrice, "\n\n");
```

So whats basically is happening is, we retrieve the price for all components. Then adding 30% to this price. Now we want to round price up to nearest 99 DKK, For example, if the price + 30% equals 2567 DKK, we round it up to 2599 DKK. Then, depending on how many systems the user has asked to see an offer on, we calculate an discount.

One system would give no discount at all. Two systems would give 2% discount, and 2% additional for each systems, up to ten systems. All above ten would give 20% discount.
See class 'PriceOffer.java' for this.

To sell a component we needed to create a prepared statement. Then we just simply update the current amount by subtracting the amount that we wants to sell. In the SQL statement, we set the amount to greatest 0. This means, that if we by accident would type 100 sells on a component, but there is only 30 on stock, we wont get a negative number.

```
PreparedStatement ps = con.prepareStatement("UPDATE CurrentStock "
                  + "SET amount = GREATEST(0, amount - " + antal + ") "
                  + "WHERE CurrentStock.name = '" + choice + "'");
           System.out.println("Sold " + antal + " " + choice + "\n");
           System.out.println("\npress [0] to return.\n");
```

See class 'Sell.java' for this.

To display our restock-list for every Saturday, we simply just retrieve the information on the currentstock and the prefered amount, then subtract them, and returns the value:

```
String query = "SELECT *"
                  + " FROM MinInventory, CurrentStock, Component"
                  + " WHERE (CurrentStock.amount < MinInventory.
                      allowedMin)"
                  + " AND (CurrentStock.name = MinInventory.name)"
                  + " AND (Component.name = CurrentStock.name)";
                  ResultSet rs = st.executeQuery(query);

                  while (rs.next()) {
                         String name = rs.getString("name");
                         String kind = rs.getString("kind");
                         amount = rs.getInt("amount");
                         prefered = rs.getInt("preferedAmount");
                         restockAmount = prefered - amount;
                         System.out.printf("%n %-39s %3s %9d",
                                name, kind, restockAmount);
                  }
```

See class 'Restock.java' for this.

4 Short user manual

To start the program, run the ComputerStore-class. this will print an options-menu with the options 1-6, simply press the number and enter to advance:

1. List of all components:
   Here is listet all components, their kind and current amount.
   To return to main-menu, pres '0' and enter.

2. List of all computer systems:
   Here is all the computer systems listet, and their components.
   Again, pres '0' and enter to return.

3. Price list:
   Here is all components- and computersystems, including their prices listed.
   Pres '0' to return.

4. Price offer:
   Here is all the names of the computersystems listed.
   Type the exact name of the system you want to see an price offer on,
   and hit enter.

   Then type how many systems you would like to see an offer on.
   The system calculates the total price with discount.
   Pres '0' to return.

5. Sell component or system:
   Pres '1' to sell components or '2' for complete computer systems.
   In both cases, type ind the exact name. Hit enter and type how many
   you would like to sell. Pres '0' to return.

6. Restocking list:
   Here is listed all the components needed to be restocked on Saturday.
   Pres '0' to return.

5 Conclusion

This program manages a computer hardware stock, and keeps information on complete computer systems in the database, and all components. The program can print a list of all components and systems, including their prices, current amount, prefered and allowed minimum. It also calculates how many of each computer systems can be build from current stock. For the 9 computer systems, the cpu's busspeed matches ram's busspeed, and cpu's matches the mainboard socket, and the ram-type matches mainboard's ram-type, and finally mainboard matches the case.

The system can also print an offer on complete computer system, with discounts from 0% up to 20%. Furthermore the system can sell single components or complete system by updating the database. Lastly the system can print a

restock-list off all the components needed to be restocked on Saturdays.

So all requirements are met.

6 Appendix

6.1 Sourcecode

ComputerStore class:

```
1  import java.io.IOException;
2  import java.sql.Connection;
3  import java.sql.DriverManager;
4  import java.sql.SQLException;
5
6  public class ComputerStore {
7      public static Connection con;
8      ComputerStore() {
9                  System.out.println("connecting to database ...");
10                 String url = "jdbc:postgresql://localhost:5432/";
11                 String user = "postgres";
12                 String password = "123bum";
13                 ComputerStore.con = null;
14
15                 //CONNECTING
16                 try {
17                        con = DriverManager.getConnection(url, user,
                              password);
18                        System.out.println("connection established!\n\n");
19
20                 } catch (SQLException ex) {
21                        System.out.println
22                        ("connection failed .. restart and try again.");
23                 }
24     }
25        public static void main(String[] args) throws IOException,
              SQLException{
26
27         ComputerSystems pc = new ComputerSystems();
28         IO io = new IO();
29         io.input();
30     }
31 }
```

ComputerSystems class:

```
1
2  import java.io.IOException;
3  import java.sql.Connection;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6  import java.sql.Statement;
7
8  public class ComputerSystems {
9      public static void ComputerSystems(Connection con) throws
           IOException {
10        try {
11            Statement st = con.createStatement();
12            String query = "SELECT *"
13                   + " FROM Computersystem";
14            ResultSet rs = st.executeQuery(query);
15
16                 while (rs.next()) {
17                     String name = rs.getString("name");
18                     String gpu = rs.getString("GPU");
19                     String cpu = rs.getString("CPU");
20                     String mainboard = rs.getString("Mainboard");
21                     String ram = rs.getString("RAM");
22                     String case_ = rs.getString("CASE_");
23                     String hdd = rs.getString("hdd");
24                     String line = "|---------------------------";
25                     System.out.print("| PC: " + name + "\n" + line
26                            + "\n| Motherboard:  " + mainboard
27                            + "\n| CPU:          " + cpu
28                            + "\n| GPU:          " + gpu
29                            + "\n| RAM:          " + ram
30                            + "\n| HDD:          " + hdd
31                            + "\n| CASE:         " + case_);
32                            System.out.print("\n|\n| From current stock '"
                                  );
33                            CalculateNumberOfSystems(con, name, cpu, gpu,
34                                   ram, hdd, case_, mainboard);
35                            System.out.print("' system(s) can be build\n\n
                                  ");
36                 }
37                 System.out.println("press [0] to return.\n");
38                   IO.returnMenu();
39                   IO.input();
40            } catch (SQLException e) {
41            }
42    }
43     public static void PCnames(Connection con) {
44
45        try {
46            Statement st = con.createStatement();
47            String query = "SELECT name "
48                   + "FROM Computersystem ";
49            ResultSet rs = st.executeQuery(query);
```

```
50
51                 while (rs.next()) {
52                     String name = rs.getString("name");
53                     System.out.println(name);
54                 }
55                  System.out.println("\nTo see computer specifications,\n
                        "
56                         + "press [0] and choose"
57                         + " 'List of all computersystems'\n\n");
58           } catch (SQLException e) {
59           }
60     }
61      public static void CalculateNumberOfSystems(Connection con, String
            name,
62             String CPU, String GPU, String RAM, String HDD,
63             String CASE, String Mainboard) throws IOException {
64
65           try {
66             Statement st = con.createStatement();
67             String query = "SELECT min(amount)"
68                    + " FROM Component, CurrentStock"
69                    + " WHERE (Component.name = '" + name + "'"
70                    + " OR Component.name = '" + CPU + "'"
71                    + " OR Component.name = '" + GPU + "'"
72                    + " OR Component.name = '" + Mainboard + "'"
73                    + " OR Component.name = '" + RAM + "'"
74                    + " OR Component.name = '" + HDD + "'"
75                    + " OR Component.name = '" + CASE + "')"
76                    + " AND CurrentStock.name = Component.name";
77             ResultSet rs = st.executeQuery(query);
78
79                   while (rs.next()) {
80                       int count = rs.getInt("min");
81                      System.out.print(count);
82                   }
83             } catch (SQLException e) {
84             }
85      }
86 }
```

IO class:

```
1
2  import java.io.IOException;
3  import java.sql.SQLException;
4  import java.util.Scanner;
5
6  public class IO extends ComputerStore {
7      private static void menu() {
8             System.out.println("
                   +-----------------------------------------+");
9             System.out.println("|               OPTIONS               |");
10            System.out.println("
                   +-----------------------------------------+");
11            System.out.println("| Type [1] - List of all components   |");
12            System.out.println("| Type [2] - List of all computer systems |
                   ");
13            System.out.println("| Type [3] - Price list               |");
14            System.out.println("| Type [4] - Price offer              |");
15            System.out.println("| Type [5] - Sell component or system |");
16            System.out.println("| Type [6] - Restocking list          |");
17            System.out.println("
                   +-----------------------------------------+");
18     }
19     public static void returnMenu() throws IOException, SQLException {
20         try {
21             Scanner sc = new Scanner(System.in);
22             switch (sc.nextInt()) {
23                 case 0:
24                     input();
25                     break;
26                 default:
27                     System.err.println("try again ..");
28                     returnMenu();
29             }
30         } catch (java.util.InputMismatchException e) {
31             System.out.println("you really need to press [0] .. try again!
                       ");
32             returnMenu();
33         }
34     }
35     public static void input() throws IOException, SQLException {
36         Scanner in = new Scanner(System.in);
37         menu();
38         switch (in.nextInt())
39         {
40             case 1:
41             Components.All_components(con);
42             break;
43
44             case 2:
45             ComputerSystems.ComputerSystems(con);
46             break;
47
```

```
48              case 3:
49              PriceList.pricelist(con);
50              break;
51
52              case 4:
53              PriceOffer.priceOffer(con);
54              break;
55
56              case 5:
57              Sell.sell(con);
58              break;
59
60              case 6:
61              Restock.restock(con);
62              break;
63
64              default:
65              System.err.println ("Not understood. Try again ..");
66              input();
67              break;
68          }
69      }
70  }
```

Components class:

```
1  import java.io.IOException;
2  import java.sql.Connection;
3  import java.sql.ResultSet;
4  import java.sql.SQLException;
5  import java.sql.Statement;
6
7  public class Components {
8          public static void All_components(Connection con) throws
               IOException{
9                  try {
10             Statement st = con.createStatement();
11             String query = "SELECT *"
12                     + " FROM Component, CurrentStock, MinInventory"
13                     + " WHERE (Component.name = CurrentStock.name) "
14                     + "AND (Component.name = MinInventory.name)"
15                     + " ORDER BY kind";
16
17                     ResultSet rs = st.executeQuery(query);
18                     System.out.printf("%-39s %3s %22s %18s %10s",
19                             "Component", "Kind", "Current amount",
20                             "Allowed minimum", "Prefered");
21
22                     while (rs.next()) {
23                             int amount = rs.getInt("amount");
24                             int allowedMin = rs.getInt("allowedMin");
25                             int prefered = rs.getInt("preferedAmount");
26                             String name = rs.getString("name");
27                             String kind = rs.getString("kind");
28                             System.out.printf("%n %-39s %3s %9d %17d %13d",
29                                     name, kind, amount, allowedMin, prefered
                                           );
30                     }
31                     System.out.println("\n\npress [0] to return.\n");
32                     IO.returnMenu();
33                     IO.input();
34             } catch (SQLException e) {
35             }
36         }
37 }
```

Pricelist class:

```
1
2  import java.io.IOException;
3  import java.sql.Connection;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6  import java.sql.Statement;
7
8  public class PriceList {
9      public static void pricelist(Connection con) throws IOException{
10
11         try {
12             Statement st = con.createStatement();
13             String query = "SELECT * FROM Component";
14
15                     ResultSet rs = st.executeQuery(query);
16                     System.out.printf("%-39s %3s %22s",
17                             "Component", "Kind", "Selling price");
18                     while (rs.next()) {
19                             String name = rs.getString("name");
20                             String kind = rs.getString("kind");
21                             double price = rs.getInt("price");
22                             double sellingPrice = price * 1.3;
23                             System.out.printf("%n %-39s %3s %10.2f %s",
24                                     name, kind, sellingPrice, "DKK");
25                     }
26                     System.out.println("\n\n");
27                     pcList(con);
28             } catch (SQLException e) {
29             }
30     }
31     public static void pcList(Connection con) throws IOException {
32          try {
33             Statement st = con.createStatement();
34             String query = "SELECT *"
35                     + " FROM Computersystem";
36             ResultSet rs = st.executeQuery(query);
37
38             while (rs.next()) {
39                     String name = rs.getString("name");
40                     String gpu = rs.getString("GPU");
41                     String cpu = rs.getString("CPU");
42                     String mainboard = rs.getString("Mainboard");
43                     String ram = rs.getString("RAM");
44                     String case_ = rs.getString("CASE_");
45                     String hdd = rs.getString("hdd");
46                     String line = "
                           |----------------------------------------"
47                             + "----------------------------";
48
49                     System.out.printf("%-3s %-56s%s%n","| PC: ",name,"Price
                        :");
50                     System.out.println(line);
```

```
51                  System.out.printf("%-17s %-40s","| Motherboard: ",
52                         mainboard); CalculatePCListPrices(con,
                               mainboard);
53
54                  System.out.printf("%n%-17s %-40s","| CPU: ",
55                         cpu); CalculatePCListPrices(con, cpu);
56
57                  System.out.printf("%n%-17s %-40s","| GPU: ",
58                         gpu); CalculatePCListPrices(con, gpu);
59
60                  System.out.printf("%n%-17s %-40s","| RAM: ",
61                         ram); CalculatePCListPrices(con, ram);
62
63                  System.out.printf("%n%-17s %-40s","| HDD: ",
64                         hdd); CalculatePCListPrices(con, hdd);
65
66                  System.out.printf("%n%-17s %-40s","| CASE: ",
67                         case_); CalculatePCListPrices(con, case_);
68
69                  System.out.print("\n|\n| Total price: ");
70                  CalculatePCListTotal(con, cpu, gpu, mainboard,
71                         ram, hdd, case_);
72                  System.out.print("\n\n");
73                }
74                System.out.println("press [0] to return.\n");
75                  IO.returnMenu();
76                  IO.input();
77          } catch (SQLException e) {
78          }
79    }
80    public static void CalculatePCListPrices(Connection con, String
          component)
81          throws IOException {
82
83         try {
84          Statement st = con.createStatement();
85          String query = "SELECT price"
86                 + " FROM Component"
87                 + " WHERE Component.name = '" + component + "'";
88          ResultSet rs = st.executeQuery(query);
89
90                while (rs.next()) {
91                   double price = rs.getDouble("price");
92                   double sellingPrice = price * 1.3;
93                  System.out.printf("%,.2f%s",sellingPrice," DKK");
94                }
95          } catch (SQLException e) {
96          }
97     }
98    public static void CalculatePCListTotal(Connection con, String CPU,
99          String GPU, String Mainboard, String RAM,
100         String HDD, String CASE) throws IOException {
101
102         try {
```

```
103             Statement st = con.createStatement();
104              String query = "SELECT sum(price) AS price"
105                     + " FROM Component"
106                     + " WHERE Component.name = '" + CPU + "'"
107                     + " OR Component.name = '" + GPU + "'"
108                     + " OR Component.name = '" + Mainboard + "'"
109                     + " OR Component.name = '" + RAM + "'"
110                     + " OR Component.name = '" + HDD + "'"
111                     + " OR Component.name = '" + CASE + "'";
112              ResultSet rs = st.executeQuery(query);
113
114                   while (rs.next()) {
115                       double price = rs.getDouble("price");
116                       double sellingPrice = price * 1.3;
117                      System.out.printf("%,.2f%s",sellingPrice," DKK");
118                   }
119              } catch (SQLException e) {
120              }
121      }
122  }
```

PriceOffer class:

```
1
2  import java.io.IOException;
3  import java.sql.Connection;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6  import java.sql.Statement;
7  import java.util.Scanner;
8
9  public class PriceOffer {
10     public static void priceOffer
11         (Connection con) throws IOException, SQLException {
12         int amount;
13         String choice;
14         System.out.println("Available systems for sale:"
15                 + "\n-----------------------");
16         ComputerSystems.PCnames(con);
17
18         Scanner sc = new Scanner(System.in);
19         System.out.println("Type which computersystem, to see offers:");
20         System.out.println("( ... OBS: its case-sensitive!)");
21         choice = sc.nextLine();
22         System.out.println("\nHow many would you like?: ");
23         amount = sc.nextInt();
24
25          try {
26             Statement st = con.createStatement();
27             String query = "SELECT *"
28                     + " FROM Computersystem"
29                     + " WHERE Computersystem.name ='" + choice + "';";
30             ResultSet rs = st.executeQuery(query);
31
32                   while (rs.next()) {
33                       String gpu = rs.getString("GPU");
34                       String cpu = rs.getString("CPU");
35                       String mainboard = rs.getString("Mainboard");
36                       String ram = rs.getString("RAM");
37                       String case_ = rs.getString("CASE_");
38                       String hdd = rs.getString("hdd");
39                       calculateDiscount(con, amount, choice, cpu,
40                               gpu, mainboard, ram, hdd, case_, amount);
41                   }
42             } catch (SQLException e) {
43             }
44     }
45         public static void calculateDiscount(Connection con, int n,
46                 String name, String CPU, String GPU, String Mainboard,
47                 String RAM, String HDD, String CASE, int amount)
48                 throws IOException {
49           try {
50            Statement st = con.createStatement();
51             String query = "SELECT sum(price) AS price"
52                     + " FROM Component"
```

```
53                  + " WHERE Component.name = '" + CPU + "'"
54                  + " OR Component.name = '" + GPU + "'"
55                  + " OR Component.name = '" + Mainboard + "'"
56                  + " OR Component.name = '" + RAM + "'"
57                  + " OR Component.name = '" + HDD + "'"
58                  + " OR Component.name = '" + CASE + "'";
59              ResultSet rs = st.executeQuery(query);
60                    while (rs.next()) {
61                       // total price for computersystem + 30%
62                       double price = rs.getDouble("price");
63                       double sellingPrice = price * 1.3;
64                       // rounded to nearest '99
65                       double roundedPrice =
66                              (Math.round(sellingPrice/100) * 100) - 1;
67                       if (n == 1) {
68                           System.out.println("\n" + n + "x '"
69                                  + name + "': with 0% discount:");
70                           System.out.printf("%s%,.2f%s", "Total price: ",
71                                  roundedPrice, "\n\n");
72                           System.out.println("press [0] to return.\n");
73                             IO.returnMenu();
74                             IO.input();
75                       } else if (n > 1 && n < 11) {
76                           double newPrice =
77                                  n * roundedPrice * (1 - ((n-1) * 0.02));
78                           double percentage = (100*(n-1)*0.02);
79
80                           System.out.println("\n" + n + "x '" + name
81                                  + "' (with " + (int)percentage
82                                  + "% discount):");
83
84                           System.out.printf("%s%,.2f%s", "Total price: ",
85                                  newPrice, "\n\n");
86
87                           System.out.println("press [0] to return.\n");
88                             IO.returnMenu();
89                             IO.input();
90                       } else {
91
92                           double newnewPrice = n * roundedPrice * 0.8;
93                            System.out.println("\n" + n + "x '" + name
94                                   + "' with 20% discount:");
95
96                            System.out.printf("%s%,.2f",
97                                   "Total price: ", newnewPrice);
98
99                            System.out.println("\n\npress [0] to return.\n")
                                  ;
100                           IO.returnMenu();
101                           IO.input();
102                      }
103                   }
104             } catch (SQLException e) {
105             }
```

```
106         }
107 }
```

Restock class:

```
1
2  import java.io.IOException;
3  import java.sql.Connection;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6  import java.sql.Statement;
7
8  public class Restock {
9      public static void restock(Connection con) throws IOException {
10         int amount;
11         int prefered;
12         int restockAmount;
13
14         System.out.println("Components to restock on saturday\n");
15         System.out.printf("%-39s %3s %22s",
16                 "Component", "Kind", "restock amount");
17         try {
18             Statement st = con.createStatement();
19             String query = "SELECT *"
20                    + " FROM MinInventory, CurrentStock, Component"
21                    + " WHERE (CurrentStock.amount < MinInventory.
                          allowedMin)"
22                    + " AND (CurrentStock.name = MinInventory.name)"
23                    + " AND (Component.name = CurrentStock.name)";
24                    ResultSet rs = st.executeQuery(query);
25
26                    while (rs.next()) {
27                           String name = rs.getString("name");
28                           String kind = rs.getString("kind");
29                           amount = rs.getInt("amount");
30                           prefered = rs.getInt("preferedAmount");
31                           restockAmount = prefered - amount;
32                           System.out.printf("%n %-39s %3s %9d",
33                                  name, kind, restockAmount);
34                    }
35             System.out.println("\n\npress [0] to return.\n");
36             IO.returnMenu();
37             IO.input();
38              } catch (SQLException e) {
39             }
40     }
41     public static void restockUpdate(Connection con, int restockAmount) {
42     }
43 }
```

Sell class:

```
1
2  import java.io.IOException;
3  import java.sql.Connection;
4  import java.sql.PreparedStatement;
5  import java.sql.ResultSet;
6  import java.sql.SQLException;
7  import java.sql.Statement;
8  import java.util.Scanner;
9
10 public class Sell {
11        public static void sell(Connection con) throws IOException {
12
13        int choice;
14        Scanner sc = new Scanner(System.in);
15        System.out.println("Do you wish to sell component(s)"
16                + " or pc-system(s)?\n");
17        System.out.println("press [1] for component");
18        System.out.println("press [2] for pc-system(s)\n");
19        choice = sc.nextInt();
20
21        switch (choice) {
22            case 1:
23                sellComponent(con);
24                break;
25            case 2:
26               sellSystem(con);
27                break;
28            default:
29                System.out.println("bye!");
30                break;
31        }
32    }
33     private static void sellComponent(Connection con) throws IOException
           {
34
35        String choice;
36        int antal;
37
38        System.out.println("Amount:\tComponent:");
39        try {
40            Statement st = con.createStatement();
41            String query = "SELECT *"
42                    + " FROM Component, CurrentStock"
43                    + " WHERE Component.name = CurrentStock.name";
44                    ResultSet rs = st.executeQuery(query);
45
46                    while (rs.next()) {
47                            String name = rs.getString("name");
48                            int amount = rs.getInt("amount");
49                            System.out.println(amount + "\t" + name);
50                    }
51                    Scanner sc = new Scanner(System.in);
```

```
52          System.out.println("\n\nWhat component do you want to sell?");
53          System.out.println("... (write component-name"
54                  + " - OBS: its case-sensitive!)");
55
56          choice = sc.nextLine();
57          System.out.println("How many do you wish to sell?");
58          antal = sc.nextInt();
59          SellComponentExecute(con, choice, antal);
60
61              } catch (SQLException e) {
62              }
63      }
64       private static void SellComponentExecute
65          (Connection con, String choice, int antal) throws IOException {
66
67            try {
68              PreparedStatement ps = con.prepareStatement("UPDATE
                    CurrentStock "
69                     + "SET amount = GREATEST(0, amount - " + antal + ") "
70                     + "WHERE CurrentStock.name = '" + choice + "'");
71              System.out.println("Sold " + antal + " " + choice + "\n");
72              System.out.println("\npress [0] to return.\n");
73
74              ps.executeUpdate();
75              ps.close();
76
77              IO.returnMenu();
78              IO.input();
79
80          } catch(SQLException se) {
81          }
82       }
83          public static void sellSystemExecute(Connection con, String
                choice,
84                 String gpu, String cpu, String mainboard, String ram,
85                 String case_, String hdd, int antal) throws IOException {
86
87              try {
88              PreparedStatement ps = con.prepareStatement("UPDATE
                    CurrentStock "
89                     + "SET amount = GREATEST(0, amount - " + antal + ")"
90                     + " WHERE CurrentStock.name = '" + gpu + "'"
91                     + " OR CurrentStock.name = '" + cpu + "'"
92                     + " OR CurrentStock.name = '" + mainboard + "'"
93                     + " OR CurrentStock.name = '" + ram + "'"
94                     + " OR CurrentStock.name = '" + case_ + "'"
95                     + " OR CurrentStock.name = '" + hdd + "'"
96                     + " OR CurrentStock.name = '" + choice + "'");
97              System.out.println("Sold " + antal + " " + choice + "\n");
98              System.out.println("\npress [0] to return.\n");
99
100             ps.executeUpdate();
101             ps.close();
102
```

```
103             IO.returnMenu();
104             IO.input();
105                       } catch(SQLException se) {
106         }
107         }
108 
109         public static void sellSystem(Connection con) throws IOException
                {
110 
111         String choice;
112         int antal;
113         System.out.println("PC-systems:\n
                ----------------------------------");
114         ComputerSystems.PCnames(con);
115         Scanner sc = new Scanner(System.in);
116         System.out.println("What PC-system do you want to sell?");
117         System.out.println("... (write system-name"
118                + " - OBS: its case-sensitive!)");
119 
120         choice = sc.nextLine();
121         System.out.println("How many do you wish to sell?");
122         antal = sc.nextInt();
123 
124         try {
125 
126             Statement st = con.createStatement();
127             String query = "SELECT *"
128                    + " FROM Computersystem"
129                    + " WHERE computersystem.name ='"
130                    + choice + "'";
131                    ResultSet rs = st.executeQuery(query);
132 
133                    while (rs.next()) {
134                           String gpu = rs.getString("gpu");
135                           String cpu = rs.getString("cpu");
136                           String mainboard = rs.getString("mainboard");
137                           String ram = rs.getString("ram");
138                           String case_ = rs.getString("case_");
139                           String hdd = rs.getString("hdd");
140                           sellSystemExecute(con, choice, gpu, cpu,
                                  mainboard,
141                                        ram, case_, hdd, antal);
142                    }
143             } catch (SQLException e) {
144             }
145         }
146 }
```