Database-programming project

Andreas Toftegaard
Supervisor: Jan Baumbach & Anders Moeslund
DM505

April 6th 2016

Contents

1 Specification

This assignment called for the design and implementation of a database to be used in a computer-store, managing all products for sale. The database would have means of approach, in the form of a java-application, to be able to view the contents of the database. Together, the application and database should be able to perform several operations defined in the assignment (project.pdf).

2 Design

From the beginning, I adopted the idea of programming functionalities into the application in chronological order, as they were listed in the assignment. The database itself was the obvious starting point, as it was on this that the application would extract data. The database would ensure some constraints on the various products in the system, though I implemented some of these in the java-application and some by constructing the database-data appropriately. On top of this, the application would be placed, containing the operations. During the project, I kept focus on the main capabilities described and as such, ensuring stability was not given priority.

3 Implementation

```
1  CREATE TABLE public.component
2  (
3  modelno integer NOT NULL,
4  kind character(20),
5  price double precision,
6  title character(50),
7  currentstock integer,
8  minimuminventory integer,
9  prefamtafterrestock integer,
10 CONSTRAINT component_pkey PRIMARY KEY (modelno)
```

The above is responsible for the probably most used table in this project; the table containing records and information regarding the various components. Initially I created it without making use of constraints, but as seen below in the second piece of code, I added a foreign key constraint, to ensure any component added (in this example the CPU-table), would be present in the component-list too. In this project I also made use of various SQL-statements, such as the query and update, to interact with the database from within the application.

```
1  CREATE TABLE public.cpu
2  (
3  modelno integer NOT NULL,
4  socket character(20),
5  busspeed integer,
6  CONSTRAINT cpu_pkey PRIMARY KEY (modelno),
7  CONSTRAINT cpu_modelno_fkey FOREIGN KEY (modelno)
8  REFERENCES public.component (modelno) MATCH SIMPLE
9  ON UPDATE NO ACTION ON DELETE NO ACTION
```

The overall java-application is, for the sake of overview, divided and named according to the desired operation it performs. I will shortly go through the functions and what they do:

```
1  public static void HowManyComponents(Connection con) throws SQLException
       {
2  Statement st = con.createStatement();
3  ResultSet rs = st.executeQuery("SELECT * FROM component");
4  while (rs.next()) {
5  System.out.println("
       ------------------------------------------------------------");
6  System.out.println(rs.getString("modelno") + " " + rs.getString("title")
       + " " + rs.getString("currentstock"));
7  }
8  rs.close();
9  st.close();
10 Scanner scanner = new Scanner(System.in);
11 System.out.println("Type anything to return to main menu");
12 if (scanner.hasNext() == true) {
13 StartMenu(con);
14 }
15 }
```

The function HowManyComponents, likely the simplest of them all, performs a query to the database, returning the columns by name 'title','currentstock', and 'modelno'. Thus providing a complete list of all components in stock.

```
1          public static int HowMany(Connection con, int cpu, int ram, int
               Case, int gpu, int mainboard) throws SQLException {
2
3          Statement st = con.createStatement();
4          Statement st1 = con.createStatement();
5          Statement st2 = con.createStatement();
6          Statement st3 = con.createStatement();
7          Statement st4 = con.createStatement();
8
9          int NumberOfCPU = 0;
10         int NumberOfRAM = 0;
11         int NumberOfCASE = 0;
12         int NumberOfMB = 0;
13         int NumberOfGPU = 0;
14
15         ResultSet cpuquery = st.executeQuery("SELECT currentstock FROM
               Component WHERE modelno =" + cpu);
16         while (cpuquery.next()) {
17         NumberOfCPU = cpuquery.getInt("currentstock");
18         }
19         ResultSet ramquery = st1.executeQuery("SELECT currentstock FROM
               Component WHERE modelno =" + ram);
20         while (ramquery.next()) {
21         NumberOfRAM = ramquery.getInt("currentstock");
22         }
```

```
23          ResultSet casequery = st2.executeQuery("SELECT currentstock FROM
                 Component WHERE modelno =" + Case);
24          while (casequery.next()) {
25          NumberOfCASE = casequery.getInt("currentstock");
26          }
27          ResultSet gpuQuery = st3.executeQuery("SELECT currentstock FROM
                Component WHERE modelno =" + gpu);
28          while (gpuQuery.next()) {
29          NumberOfGPU = gpuQuery.getInt("currentstock");
30          }
31          ResultSet MbQuery = st4.executeQuery("SELECT currentstock FROM
                Component WHERE modelno =" + mainboard);
32          while (MbQuery.next()) {
33          NumberOfMB = MbQuery.getInt("currentstock");
34          }
35
36
37          int lowest;
38          int[] numbers = {NumberOfCPU, NumberOfRAM, NumberOfCASE,
                NumberOfMB, NumberOfGPU};
39          lowest = numbers[0];
40          for (int index = 1; index < numbers.length; index++)
41          if (numbers[index] < lowest) {
42          lowest = numbers[index];
43          }
44          return lowest;
45          }
```

Rather than the showing the function printing the result of the above, which is not very interesting, this function in the application returns the number of systems buildable with the current stock. It firstly takes as arguments the various 'modelno' ints queried from the 'computersystems' table and performs another query on the currentstock of each of the parts, inserting the results into an array. Since any computersystem will only require one piece of each kind, the resulting total of systems buildable will be limited by the lowest stock of any part, thus we can return the lowest of the 5 elements in the array as the answer.

```
1         public static double FinalPrice(double price) {
2         price = price * 1.3;
3         int cast = (int) price;
4         cast = (cast / 100) * 100;
5         double PriceDone = (double) cast;
6         PriceDone = PriceDone + 99.99;
7         return PriceDone;
8         }
9
10        public static double WithBulkdiscount(double Systemprice, int
              NRofSystems) {
11    Systemprice = Systemprice * (1 - (0.00 + (0.02 * (NRofSystems - 1)))
          );
12    double roundOff = Math.round(Systemprice * 100.0) / 100.0;
13    return roundOff;
```

```
14      }
```

In computing a price-offer for a given system, and satisfying the given price-format(rounding up, ending in 99), I employ the 2 functions shown above. FinalPrice will take as argument the price taken directly from the database, and by casting to int and using integer-division, one can erase the last digits and replace them with 99, thus satisfying the requirement.
WithBulkdiscount is used in the price-offer operation, where the user can request a price for a given system and a given amount. A 2% discount is applied for every additional system bought in addition to the first by accordingly multiplying the discount.

```
1       public static void MakeSale(Connection con) throws SQLException
            {
2       Statement st = con.createStatement();
3       ResultSet rs = st.executeQuery("SELECT title, modelno FROM
            component");
4       while (rs.next()) {
5       System.out.println(rs.getString("title") + rs.getInt("modelno"))
            ;
6       }
7       Statement st1 = con.createStatement();
8       ResultSet rs1 = st1.executeQuery("SELECT title FROM
            computersystems");
9       while (rs1.next()) {
10      System.out.println(rs1.getString("title"));
11      }
12      System.out.println("Type 'part' or 'system' followed by modelno
            or systemname. eg. 'part 1001'");
13      Scanner scanner = new Scanner(System.in);
14      String PartOrSystem = scanner.next();
15      if (PartOrSystem.contentEquals("system")) {
16      String Systemname = "'" + scanner.next() + "'";
17      if (InStockSystem(con,Systemname)){
18      FindParts(Systemname, con);}
19      } else {
20      RemovePart(Integer.parseInt(scanner.next()), con);
21      }
22      System.out.println("Type anything to return to main menu");
23      if (scanner.hasNext() == true) {
24      StartMenu(con);
25      }
26      }
```

When entering the page for entering a sale, the function will ask for a type i.e "system" or "part" followed by either the modelno for a part or the name of a system. Depending on if 'part' or 'system' was entered, the function will call FindParts or RemovePart.

```
1       private static void RemovePart(int modelno, Connection con)
            throws SQLException {
2       Statement st = con.createStatement();
3       st.executeUpdate("UPDATE component SET currentstock =
```

```
            currentstock-1 WHERE modelno =" + modelno);
        System.out.println("stock has been updated");
        }

        private static void FindParts(String Systemname, Connection con)
             throws SQLException {
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("SELECT * FROM computersystems
            WHERE title =" + Systemname);
        while (rs.next()) {
        RemovePart(rs.getInt("cpu"), con);
        RemovePart(rs.getInt("mainboard"), con);
        RemovePart(rs.getInt("ram"), con);
        RemovePart(rs.getInt("graphicscard"), con);
        RemovePart(rs.getInt("cases"), con);
        }
        System.out.println("stock has been updated");
        }
```

RemovePart and FindParts are somewhat similar in the sense that FindParts eventually calls RemovePart on all the parts it queries. FindParts is through querying the computersystems table able to call RemovePart on the parts contained in that system. RemovePart then contains an update-statement which reduces the stock by one for the modelno given as argument. Through the InStock and InStockSystem functions the application produces a boolean on whether the sale can in fact be done, by comparing the currentstock with the minimuminventory.

```
        private static void Restocklist(Connection con) throws
            SQLException {
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("SELECT * FROM component");
        System.out.println("Negative restock indicates surplus compared
            to preferred amount");
        while (rs.next()){
        System.out.println(rs.getString("title")+(rs.getInt("
            prefamtafterrestock")-rs.getInt("currentstock")));
        }
        System.out.println("Type anything to return to main menu");
        Scanner scanner = new Scanner(System.in);
        if (scanner.hasNext() == true) {
        StartMenu(con);
        }
        }
```

The final major function, Restocklist, will provide the user with the difference between the preferred stock and currentstock. I retained the possibility for it to print a negative restock, which shows a surplus compared to the preferred.

4 Testing

To limit the space used (the pictures take up quite a bit), I will show a picture for each function tested and have comments in the captions.

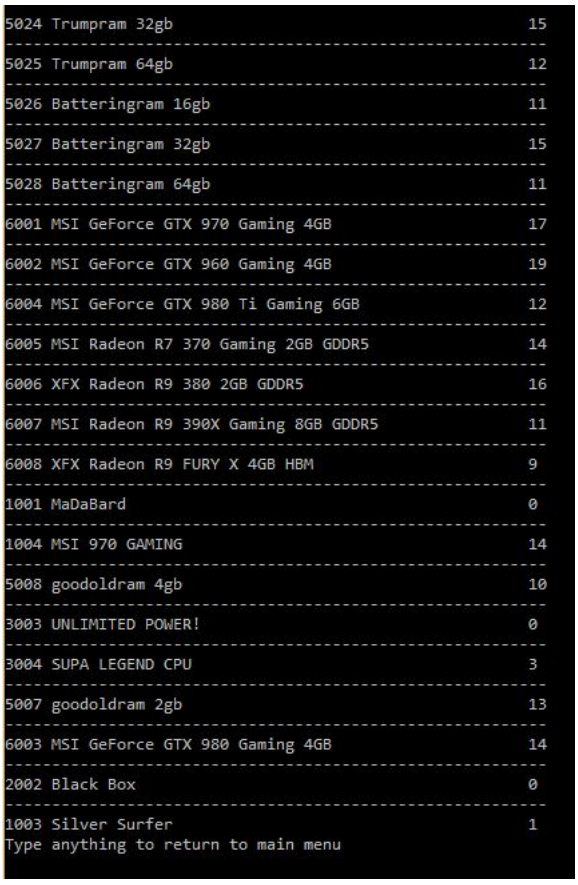

Figure 1: The stock-list. Takes no input and thus no possibility for error, database excluded

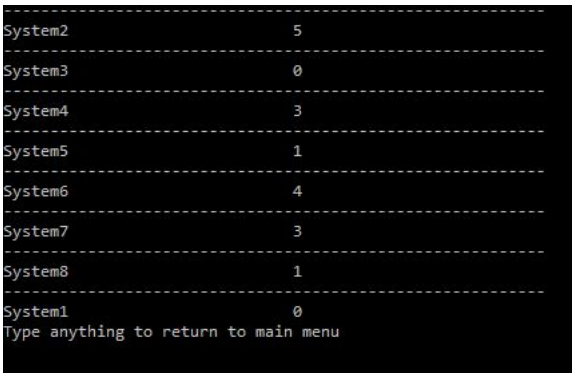

Figure 2: Systems that can be build. Not prone to error.

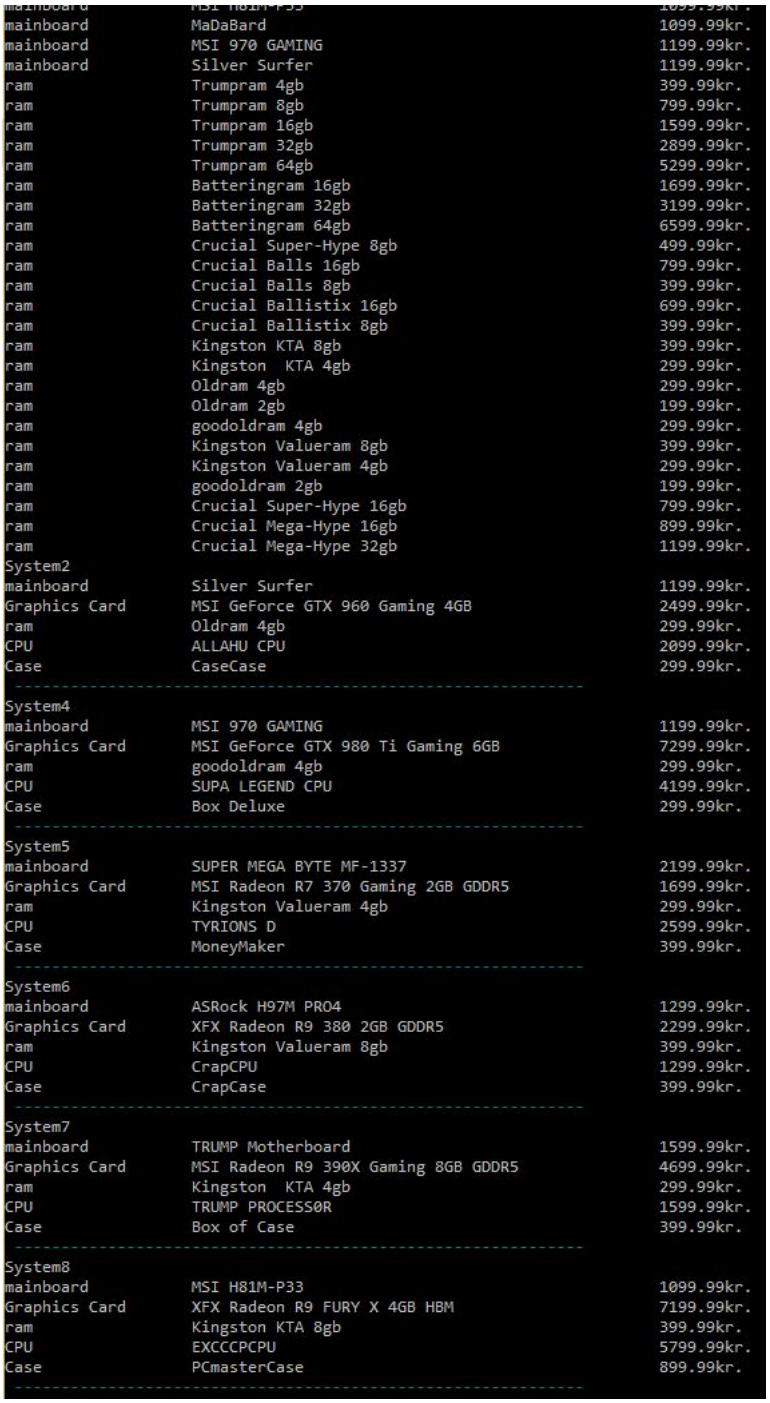

Figure 3: The price-list.

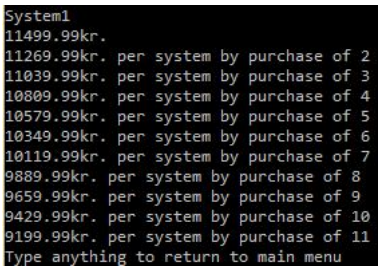

Figure 4: A price-offer for System1. It is possible throw an exception by inputting an incorrect name

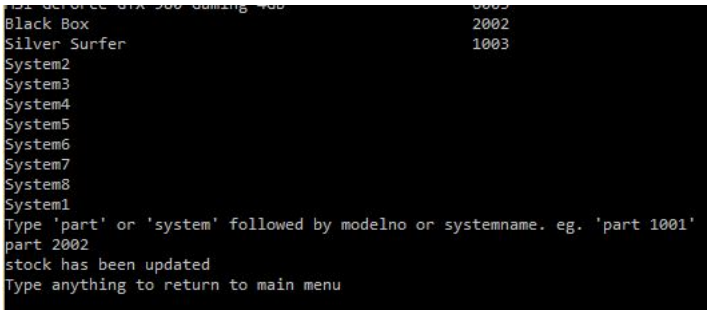

Figure 5: An example of a sale being inputted. Again, it is possible to throw an exception with incorrect input.

title character(50)	currentstock integer
MaDaBard	0
Silver Surfer	25
Silver Surfer	1

Figure 6: The database before a sale is inputted, for reference.

title character(50)	currentstock integer
MaDaBard	0
Silver Surfer	24
Silver Surfer	1

Figure 7: The database after a sale.

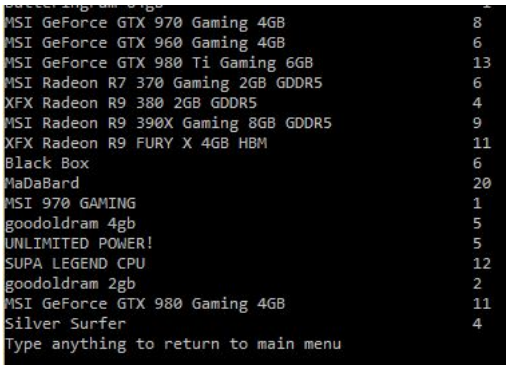

Figure 8: Restocking-list.

5 User-manual

To best give an overview, the application is divided by the operations specified in the assignment. When initiating the program, the main menu will appear and the user can choose an operation by inputting the corresponding integer. Most of the operations will finish without further input, but 'price-offer' and 'execute sale' requires further input, the first by inputting one of the system-names and the second by inputting the kind of sale and name or modelnumber. Both of these are further described by the application, and ought to be fairly easy to navigate.

6 Conclusion

Having finished the project and being able to look in retrospect, overall I am satisfied with the result. Some things, however, could have been better. During the project I preferred to keep error-handling in the java-application, for example the InStock-check operation, and together with this I did not implement actual constraints for creating a computersystem, making sure the parts matched. While these things are not required for the program to function properly, I do feel that in the spirit of this being a database-project, these things could have been implemented.

7 Appendix (source code)

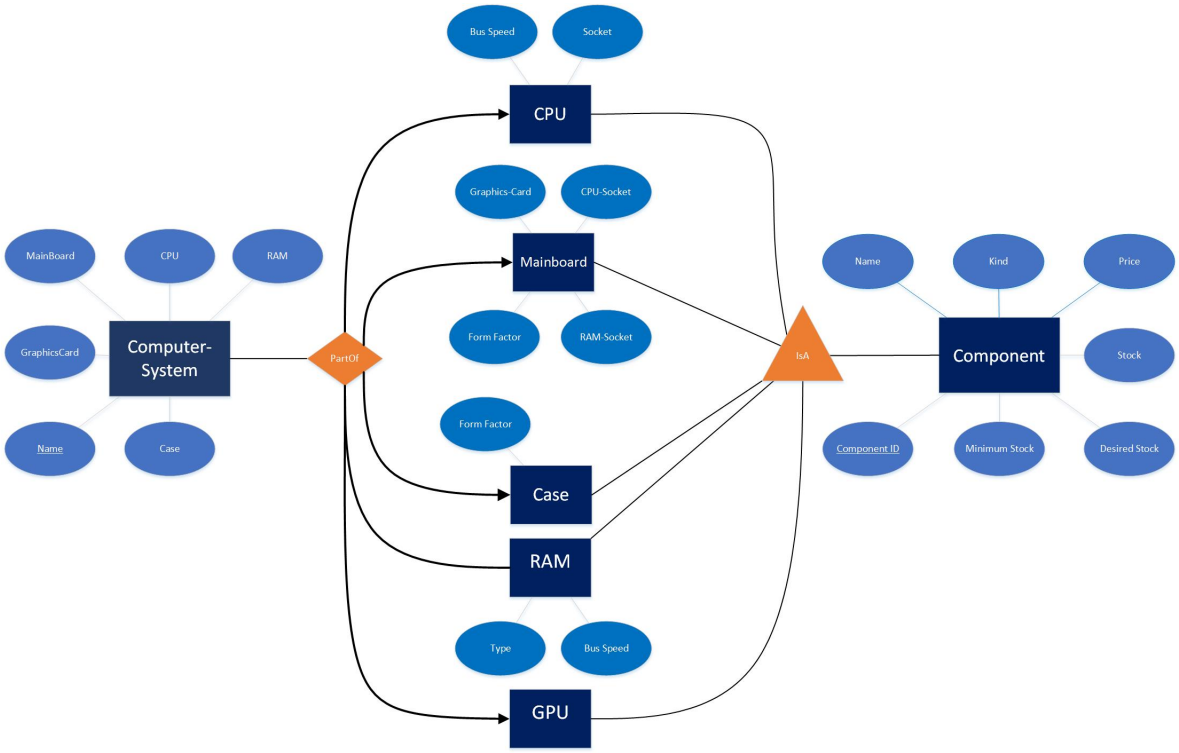

Figure 9: The ER model for the database. Note some names are slightly different from the actual database, as this was created before beginning the programming (eg. component id = modelno).

7.1 Relational model and arguments for 3nf

Component(modelno,title,price,minimumstock,currentstock,preferredstock,kind)
CPU(modelno,busspeed,socket)
Mainboard(modelno,Graphicscard,CPU-socket,Form-factor,Ram-socket)
Case(modelno,Form-factor)
Ram(modelno,type,busspeed)
GPU(modelno)
Because the primary keys employed, 'modelno' and 'name' are the only ones upon which other data is dependent i.e no transitive dependencies, the data is in third normal. In other words, one can only extract a row from the tables by the modelno or in the case of 'Computer-system', by name. By adding the modelno, a unique identifier for all parts, this is possible. Note I include the source code for the java-application in this pdf. The SQL-piece can be found in the .query file included as well as the database dump.

```
1 import java.sql.*;
2 import java.util.Scanner;
3 import java.util.logging.*;
```

```
4  //java -cp postgresql-9.4-1201.jdbc4.jar:. DBtest
5
6  public class test {
7
8      public static void main(String[] args) throws SQLException {
9          String url = "jdbc:postgresql://localhost:5432/postgres";
10         String user = "postgres";
11         String password = "1assaP22";
12         Connection con = null;
13
14         //CONNECTING
15         System.out.println("connecting to database...");
16         try {
17             con = DriverManager.getConnection(url, user, password);
18
19         } catch (SQLException ex) {
20             Logger lgr = Logger.getLogger(test.class.getName());
21             lgr.log(Level.WARNING, ex.getMessage(), ex);
22
23         }
24         System.out.println("connection established...");
25         // Call main menu
26         StartMenu(con);
27     }
28         // Function prints 'title','modelno', and 'currentstock' from
               component-table
29     public static void HowManyComponents(Connection con) throws
           SQLException {
30         Statement st = con.createStatement();
31         ResultSet rs = st.executeQuery("SELECT * FROM component");
32         while (rs.next()) {
33             System.out.println("
                   ------------------------------------------------------
                   ");
34             System.out.println(rs.getString("modelno") + " " + rs.
                   getString("title") + " " + rs.getString("currentstock"));
35         }
36         rs.close();
37         st.close();
38         Scanner scanner = new Scanner(System.in);
39         System.out.println("Type anything to return to main menu");
40         if (scanner.hasNext() == true) {
41             StartMenu(con);
42         }
43     }
44         // Main menu. Prints options and takes integer as choice
45     public static void StartMenu(Connection con) throws SQLException {
46
47         Scanner scanner = new Scanner(System.in);
48
49         System.out.println("============================");
50         System.out.println("|     MENU SELECTION    |");
51         System.out.println("============================");
52         System.out.println("|     Options:          |");
```

```
53          System.out.println("|     1. Show Stock   |");
54          System.out.println("|     2. Comp-Systems |");
55          System.out.println("|     3. Price-List   |");
56          System.out.println("|     4. Price-Offer  |");
57          System.out.println("|     5. Execute Sale |");
58          System.out.println("|     6. Restocking list|");
59          System.out.println("|     7. Exit         |");
60          System.out.println("============================");
61          System.out.println("What would you like to do?");
62
63          int Response = Integer.parseInt(scanner.nextLine());
64          if (Response == 1) try {
65              HowManyComponents(con);
66          } catch (SQLException e) {
67              e.printStackTrace();
68          }
69          if (Response == 2) try {
70              ShowCompSys(con);
71          } catch (SQLException e) {
72              e.printStackTrace();
73          }
74          if (Response == 3) {
75              PriceList(con);
76          }
77          if (Response == 4) {
78              PriceOfferMenu(con);
79          }
80          if (Response == 5) {
81              MakeSale(con);
82          }
83          if (Response == 6) {
84              Restocklist(con);
85          }
86          if (Response == 7) {
87              Runtime.getRuntime().exit(0);
88          }
89      }
90          // Prints 'title' from computersystem and calls 'howmany' for
                computing amount buildable
91      public static void ShowCompSys(Connection con) throws SQLException {
92          Statement st = con.createStatement();
93          ResultSet rs = st.executeQuery("SELECT * FROM Computersystems");
94
95          while (rs.next()) {
96              System.out.println("
                    ------------------------------------------------------------
                    ");
97              System.out.println(rs.getString("title") + " " + HowMany(con,
                    rs.getInt("cpu"), rs.getInt("ram"), rs.getInt("cases"),
                    rs.getInt("gpu"), rs.getInt("mainboard")));
98          }
99          rs.close();
100         st.close();
101         Scanner scanner = new Scanner(System.in);
```

```
102         System.out.println("Type anything to return to main menu");
103         if (scanner.hasNext() == true) {
104             StartMenu(con);
105         }
106     }
107         // takes modelno of parts and returns smallest
108     public static int HowMany(Connection con, int cpu, int ram, int Case,
            int gpu, int mainboard) throws SQLException {
109
110         Statement st = con.createStatement();
111         Statement st1 = con.createStatement();
112         Statement st2 = con.createStatement();
113         Statement st3 = con.createStatement();
114         Statement st4 = con.createStatement();
115
116         int NumberOfCPU = 0;
117         int NumberOfRAM = 0;
118         int NumberOfCASE = 0;
119         int NumberOfMB = 0;
120         int NumberOfGPU = 0;
121
122         ResultSet cpuquery = st.executeQuery("SELECT currentstock FROM
                Component WHERE modelno =" + cpu);
123         while (cpuquery.next()) {
124             NumberOfCPU = cpuquery.getInt("currentstock");
125         }
126         ResultSet ramquery = st1.executeQuery("SELECT currentstock FROM
                Component WHERE modelno =" + ram);
127         while (ramquery.next()) {
128             NumberOfRAM = ramquery.getInt("currentstock");
129         }
130         ResultSet casequery = st2.executeQuery("SELECT currentstock FROM
                Component WHERE modelno =" + Case);
131         while (casequery.next()) {
132             NumberOfCASE = casequery.getInt("currentstock");
133         }
134         ResultSet gpuQuery = st3.executeQuery("SELECT currentstock FROM
                Component WHERE modelno =" + gpu);
135         while (gpuQuery.next()) {
136             NumberOfGPU = gpuQuery.getInt("currentstock");
137         }
138         ResultSet MbQuery = st4.executeQuery("SELECT currentstock FROM
                Component WHERE modelno =" + mainboard);
139         while (MbQuery.next()) {
140             NumberOfMB = MbQuery.getInt("currentstock");
141         }
142
143
144         int lowest;
145         int[] numbers = {NumberOfCPU, NumberOfRAM, NumberOfCASE,
                NumberOfMB, NumberOfGPU};
146         lowest = numbers[0];
147         for (int index = 1; index < numbers.length; index++)
148             if (numbers[index] < lowest) {
```

```
149                 lowest = numbers[index];
150             }
151         return lowest;
152     }
153     // Prints 'title' and price of every product, grouped by 'kind'
154     private static void PriceList(Connection con) {
155         try {
156             Statement st = con.createStatement();
157             Statement st2 = con.createStatement();
158             ResultSet PriceListSys = st2.executeQuery("SELECT * FROM
                    computersystems");
159             ResultSet PriceListParts = st.executeQuery("SELECT * FROM
                    component ORDER BY kind");
160             while (PriceListParts.next()) {
161                 //  System.out.printf(rs.getString("title")+rs.getDouble
                        (""));
162                 System.out.println(PriceListParts.getString("kind") +
                        PriceListParts.getString("title") + FinalPrice(
                        PriceListParts.getDouble("price")) + "kr.");
163             }
164             while (PriceListSys.next()) {
165                 if (HowMany(con, PriceListSys.getInt("cpu"), PriceListSys.
                        getInt("ram"), PriceListSys.getInt("cases"),
                        PriceListSys.getInt("gpu"), PriceListSys.getInt("
                        mainboard")) > 0) {
166                     System.out.print(PriceListSys.getString("title") +
                            System.lineSeparator()
167                             + FindName(con, PriceListSys.getInt("mainboard"
                                ))
168                             + FindPrice(con, PriceListSys.getInt("mainboard
                                ")) + "kr." + System.lineSeparator()
169                             + FindName(con, PriceListSys.getInt("gpu"))
170                             + FindPrice(con, PriceListSys.getInt("gpu")) +
                                "kr." + System.lineSeparator()
171                             + FindName(con, PriceListSys.getInt("ram"))
172                             + FindPrice(con, PriceListSys.getInt("ram")) +
                                "kr." + System.lineSeparator()
173                             + FindName(con, PriceListSys.getInt("cpu"))
174                             + FindPrice(con, PriceListSys.getInt("cpu")) +
                                "kr." + System.lineSeparator()
175                             + FindName(con, PriceListSys.getInt("cases"))
176                             + FindPrice(con, PriceListSys.getInt("cases"))
                                    + "kr." + System.lineSeparator()
177                     );
178                     System.out.println((char) 27 + "[36m
                            ----------------------------------------------------------
                            " + (char) 27 + "[0m");
179                 }
180             }
181             Scanner scanner = new Scanner(System.in);
182             System.out.println("Type anything to return to main menu");
183             if (scanner.hasNext() == true) {
184                 StartMenu(con);
185             }
```

```
186         } catch (SQLException e) {
187             e.printStackTrace();
188         }
189     }
190     // takes modelno and returns value of 'price'
191     public static double FindPrice(Connection con, int modelno) throws
            SQLException {
192         double price = 0;
193         Statement st = con.createStatement();
194         ResultSet rs = st.executeQuery("SELECT price FROM component WHERE
                modelno =" + modelno);
195         while (rs.next()) {
196             price = rs.getDouble("price");
197         }
198         return FinalPrice(price);
199     }
200     // takes modelno and returns strings in 'title' and 'kind'
201     public static String FindName(Connection con, int modelno) throws
            SQLException {
202         String name = null;
203         String kind = null;
204         Statement st = con.createStatement();
205         ResultSet rs = st.executeQuery("SELECT * FROM component WHERE
                modelno =" + modelno);
206         while (rs.next()) {
207             name = rs.getString("title");
208             kind = rs.getString("kind");
209         }
210         return kind + name;
211     }
212     // menu for choosing system for offer
213     public static void PriceOfferMenu(Connection con) throws SQLException
            {
214         Scanner scanner = new Scanner(System.in);
215         Statement st = con.createStatement();
216         ResultSet rs = st.executeQuery("SELECT title FROM computersystems
                ");
217         while (rs.next()) {
218             System.out.println(rs.getString("title"));
219         }
220         System.out.println("Type in a system for an offer");
221         String Response = "'" + scanner.next() + "'";
222         System.out.println(PriceOffer(con, Response) + "kr.");
223         for (int i = 2; i < 12; i++) {
224             System.out.println(WithBulkdiscount(PriceOffer(con, Response),
                    i) + "kr. per system by purchase of " + i);
225         }
226         System.out.println("Type anything to return to main menu");
227         if (scanner.hasNext() == true) {
228             StartMenu(con);
229         }
230     }
231     // Takes name of system. Uses FindPrice and FinalPrice to return a
            total price.
```

```
232     public static double PriceOffer(Connection con, String system) throws
            SQLException {
233         Statement st = con.createStatement();
234         ResultSet rs = st.executeQuery("SELECT * FROM computersystems
                WHERE title =" + system);
235         double p = 0;
236         while (rs.next()) {
237             p = (FinalPrice(FindPrice(con, rs.getInt("cpu")) + FindPrice(
                    con, rs.getInt("mainboard")) + FindPrice(con, rs.getInt("
                    ram")) + FindPrice(con, rs.getInt("gpu")) + FindPrice(con
                    , rs.getInt("cases")))
238             );
239         }
240         return p;
241     }
242     // casting and integer division to return a rounded up price ending
            in 99.99
243     public static double FinalPrice(double price) {
244         price = price * 1.3;
245         int cast = (int) price;
246         cast = (cast / 100) * 100;
247         double PriceDone = (double) cast;
248         PriceDone = PriceDone + 99.99;
249         return PriceDone;
250     }
251     // Takes systemprice and number of systems, returns the price with
            applied discount of 2%
252     public static double WithBulkdiscount(double Systemprice, int
            NRofSystems) {
253         Systemprice = Systemprice * (1 - (0.00 + (0.02 * (NRofSystems -
                1))));
254         double roundOff = Math.round(Systemprice * 100.0) / 100.0;
255         return roundOff;
256     }
257     // Menu for choosing part or system to buy
258     public static void MakeSale(Connection con) throws SQLException {
259         Statement st = con.createStatement();
260         ResultSet rs = st.executeQuery("SELECT title, modelno FROM
                component");
261         while (rs.next()) {
262             System.out.println(rs.getString("title") + rs.getInt("modelno"
                    ));
263         }
264         Statement st1 = con.createStatement();
265         ResultSet rs1 = st1.executeQuery("SELECT title FROM
                computersystems");
266         while (rs1.next()) {
267             System.out.println(rs1.getString("title"));
268         }
269         System.out.println("Type 'part' or 'system' followed by modelno
                or systemname. eg. 'part 1001'");
270         Scanner scanner = new Scanner(System.in);
271         String PartOrSystem = scanner.next();
272         if (PartOrSystem.contentEquals("system")) {
```

```
273             String Systemname = "'" + scanner.next() + "'";
274             if (InStockSystem(con,Systemname)){
275                 FindParts(Systemname, con);}
276         } else {
277             RemovePart(Integer.parseInt(scanner.next()), con);
278         }
279         System.out.println("Type anything to return to main menu");
280         if (scanner.hasNext() == true) {
281             StartMenu(con);
282         }
283     }
284     // Removes 1 of given part with UPDATE statement
285     private static void RemovePart(int modelno, Connection con) throws
            SQLException {
286         Statement st = con.createStatement();
287         st.executeUpdate("UPDATE component SET currentstock =
                currentstock-1 WHERE modelno =" + modelno);
288         System.out.println("stock has been updated.");
289     }
290     // Takes systemname and calls RemovePart on all parts
291     private static void FindParts(String Systemname, Connection con)
            throws SQLException {
292         Statement st = con.createStatement();
293         ResultSet rs = st.executeQuery("SELECT * FROM computersystems
                WHERE title =" + Systemname);
294         while (rs.next()) {
295             RemovePart(rs.getInt("cpu"), con);
296             RemovePart(rs.getInt("mainboard"), con);
297             RemovePart(rs.getInt("ram"), con);
298             RemovePart(rs.getInt("graphicscard"), con);
299             RemovePart(rs.getInt("cases"), con);
300         }
301         System.out.println("stock has been updated.");
302     }
303     // Prints the difference between currentstock and preferredstock for
            all parts
304     private static void Restocklist(Connection con) throws SQLException {
305         Statement st = con.createStatement();
306         ResultSet rs = st.executeQuery("SELECT * FROM component");
307         System.out.println("Negative restock indicates surplus compared
                to preferred amount");
308         while (rs.next()){
309             System.out.println(rs.getString("title")+(rs.getInt("
                    prefamtafterrestock")-rs.getInt("currentstock")));
310         }
311         System.out.println("Type anything to return to main menu");
312         Scanner scanner = new Scanner(System.in);
313         if (scanner.hasNext() == true) {
314             StartMenu(con);
315         }
316     }
317     // Boolean to check if a sale would reduce stock below
            minimuminventory
318     private static boolean InStockPart (int modelno, Connection con)
```

```
        throws SQLException {
319         Statement st = con.createStatement();
320         ResultSet rs = st.executeQuery("SELECT * FROM component WHERE
                modelno=" +modelno);
321         while (rs.next()) {
322             if ((rs.getInt("currentstock") <= rs.getInt("minimuminventory"
                    ))) {
323                 return false;
324             }
325         }
326         return true;
327     }
328     // Same as above only on all parts in a given system
329     private static boolean InStockSystem (Connection con,String
            Systemname) throws SQLException {
330         Statement st = con.createStatement();
331         ResultSet rs = st.executeQuery("SELECT * FROM computersystems
                WHERE title =" + Systemname);
332         while (rs.next()){
333             System.out.println("Action would violate minimuminventory");
334             if (InStockPart(rs.getInt("cpu"),con)==false) {return false;}
335             if (InStockPart(rs.getInt("mainboard"),con)==false) {return
                    false;}
336             if (InStockPart(rs.getInt("ram"),con)==false) {return false;}
337             if (InStockPart(rs.getInt("graphicscard"),con)==false) {return
                     false;}
338             if (InStockPart(rs.getInt("cases"),con)==false) {return false
                    ;}
339         }
340         return true;
341     }
342 }
```