

Database-programming project

Andreas Toftegaard
Supervisor: Jan Baumbach & Anders Moeslund
DM505

April 6th 2016

Contents

1	Specification	2
2	Design	2
3	Implementation	2
4	Testing	7
5	User-manual	11
6	Conclusion	11
7	Appendix (source code)	12
7.1	Relational model and arguments for 3nf	12

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 {
3     Statement st = con.createStatement();
4     ResultSet rs = st.executeQuery("SELECT * FROM component");
5     while (rs.next()) {
6         System.out.println("
7             -----");
8         System.out.println(rs.getString("modelno") + " " + rs.getString("title")
9             + " " + rs.getString("currentstock"));
10    }
11    rs.close();
12    st.close();
13    Scanner scanner = new Scanner(System.in);
14    System.out.println("Type anything to return to main menu");
15    if (scanner.hasNext() == true) {
16        StartMenu(con);
17    }
18 }
```

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
2     Case, int gpu, int mainboard) throws SQLException {
3
4     Statement st = con.createStatement();
5     Statement st1 = con.createStatement();
6     Statement st2 = con.createStatement();
7     Statement st3 = con.createStatement();
8     Statement st4 = con.createStatement();
9
10    int NumberOfCPU = 0;
11    int NumberOfRAM = 0;
12    int NumberOfCASE = 0;
13    int NumberOfMB = 0;
14    int NumberOfGPU = 0;
15
16    ResultSet cpuquery = st.executeQuery("SELECT currentstock FROM
17        Component WHERE modelno =" + cpu);
18    while (cpuquery.next()) {
19        NumberOfCPU = cpuquery.getInt("currentstock");
20    }
21    ResultSet ramquery = st1.executeQuery("SELECT currentstock FROM
22        Component WHERE modelno =" + ram);
23    while (ramquery.next()) {
24        NumberOfRAM = ramquery.getInt("currentstock");
25    }
26 }
```

```

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      {
3          Statement st = con.createStatement();
4          ResultSet rs = st.executeQuery("SELECT title, modelno FROM
5              component");
6          while (rs.next()) {
7              System.out.println(rs.getString("title") + rs.getInt("modelno"))
8              ;
9          }
10         Statement st1 = con.createStatement();
11         ResultSet rs1 = st1.executeQuery("SELECT title FROM
12             computersystems");
13         while (rs1.next()) {
14             System.out.println(rs1.getString("title"));
15         }
16         System.out.println("Type 'part' or 'system' followed by modelno
17             or systemname. eg. 'part 1001'");
18         Scanner scanner = new Scanner(System.in);
19         String PartOrSystem = scanner.next();
20         if (PartOrSystem.contentEquals("system")) {
21             String Systemname = "" + scanner.next() + "";
22             if (InStockSystem(con, Systemname)){
23                 FindParts(Systemname, con);}
24             } else {
25                 RemovePart(Integer.parseInt(scanner.next()), con);
26             }
27         System.out.println("Type anything to return to main menu");
28         if (scanner.hasNext() == true) {
29             StartMenu(con);
30         }
31     }
```

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)
2      throws SQLException {
3          Statement st = con.createStatement();
4          st.executeUpdate("UPDATE component SET currentstock =
```

```

4         currentstock-1 WHERE modelno =" + modelno);
5     System.out.println("stock has been updated");
6     }
7
8     private static void FindParts(String Systemname, Connection con)
9         throws SQLException {
10         Statement st = con.createStatement();
11         ResultSet rs = st.executeQuery("SELECT * FROM computersystems
12             WHERE title =" + Systemname);
13         while (rs.next()) {
14             RemovePart(rs.getInt("cpu"), con);
15             RemovePart(rs.getInt("mainboard"), con);
16             RemovePart(rs.getInt("ram"), con);
17             RemovePart(rs.getInt("graphicscard"), con);
18             RemovePart(rs.getInt("cases"), con);
19         }
20         System.out.println("stock has been updated");
21     }

```

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.

```

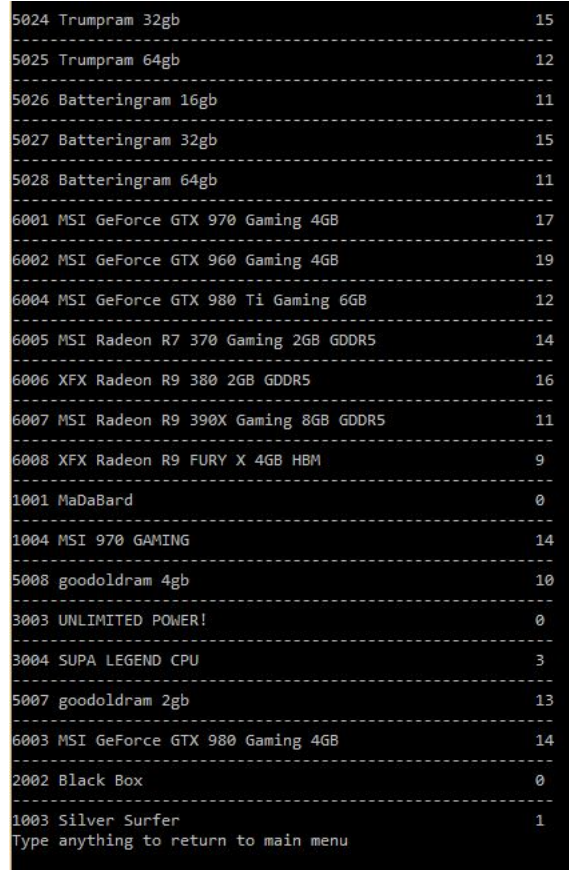
1     private static void Restocklist(Connection con) throws
2         SQLException {
3         Statement st = con.createStatement();
4         ResultSet rs = st.executeQuery("SELECT * FROM component");
5         System.out.println("Negative restock indicates surplus compared
6             to preferred amount");
7         while (rs.next()){
8             System.out.println(rs.getString("title")+(rs.getInt("
9                 prefamtafterrestock")-rs.getInt("currentstock")));
10        }
11        System.out.println("Type anything to return to main menu");
12        Scanner scanner = new Scanner(System.in);
13        if (scanner.hasNext() == true) {
14            StartMenu(con);
15        }
16    }

```

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.

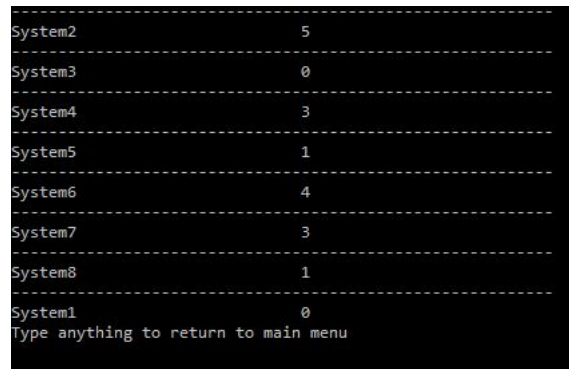
4 Testing

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.



5024 Trumpram 32gb	15
5025 Trumpram 64gb	12
5026 Batteringram 16gb	11
5027 Batteringram 32gb	15
5028 Batteringram 64gb	11
6001 MSI GeForce GTX 970 Gaming 4GB	17
6002 MSI GeForce GTX 960 Gaming 4GB	19
6004 MSI GeForce GTX 980 Ti Gaming 6GB	12
6005 MSI Radeon R7 370 Gaming 2GB GDDR5	14
6006 XFX Radeon R9 380 2GB GDDR5	16
6007 MSI Radeon R9 390X Gaming 8GB GDDR5	11
6008 XFX Radeon R9 FURY X 4GB HBM	9
1001 MaDaBard	0
1004 MSI 970 GAMING	14
5008 goodoldram 4gb	10
3003 UNLIMITED POWER!	0
3004 SUPA LEGEND CPU	3
5007 goodoldram 2gb	13
6003 MSI GeForce GTX 980 Gaming 4GB	14
2002 Black Box	0
1003 Silver Surfer	1
Type anything to return to main menu	

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



A screenshot of a terminal window with a black background and white text. The text is organized into a list of systems, each followed by a number, and separated by dashed horizontal lines. The systems are listed from top to bottom: System2 (5), System3 (0), System4 (3), System5 (1), System6 (4), System7 (3), System8 (1), and System1 (0). At the bottom of the list, there is a prompt that says 'Type anything to return to main menu'.

System2	5
System3	0
System4	3
System5	1
System6	4
System7	3
System8	1
System1	0

Type anything to return to main menu

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

mainboard	MSI H81M-P33	1099.99kr.
mainboard	MaDaBard	1099.99kr.
mainboard	MSI 970 GAMING	1199.99kr.
mainboard	Silver Surfer	1199.99kr.
ram	Trumpram 4gb	399.99kr.
ram	Trumpram 8gb	799.99kr.
ram	Trumpram 16gb	1599.99kr.
ram	Trumpram 32gb	2899.99kr.
ram	Trumpram 64gb	5299.99kr.
ram	Batteringram 16gb	1699.99kr.
ram	Batteringram 32gb	3199.99kr.
ram	Batteringram 64gb	6599.99kr.
ram	Crucial Super-Hype 8gb	499.99kr.
ram	Crucial Balls 16gb	799.99kr.
ram	Crucial Balls 8gb	399.99kr.
ram	Crucial Ballistix 16gb	699.99kr.
ram	Crucial Ballistix 8gb	399.99kr.
ram	Kingston KTA 8gb	399.99kr.
ram	Kingston KTA 4gb	299.99kr.
ram	Oldram 4gb	299.99kr.
ram	Oldram 2gb	199.99kr.
ram	goodoldram 4gb	299.99kr.
ram	Kingston Valueram 8gb	399.99kr.
ram	Kingston Valueram 4gb	299.99kr.
ram	goodoldram 2gb	199.99kr.
ram	Crucial Super-Hype 16gb	799.99kr.
ram	Crucial Mega-Hype 16gb	899.99kr.
ram	Crucial Mega-Hype 32gb	1199.99kr.
System2		
mainboard	Silver Surfer	1199.99kr.
Graphics Card	MSI GeForce GTX 960 Gaming 4GB	2499.99kr.
ram	Oldram 4gb	299.99kr.
CPU	ALLAHU CPU	2099.99kr.
Case	CaseCase	299.99kr.
System4		
mainboard	MSI 970 GAMING	1199.99kr.
Graphics Card	MSI GeForce GTX 980 Ti Gaming 6GB	7299.99kr.
ram	goodoldram 4gb	299.99kr.
CPU	SUPA LEGEND CPU	4199.99kr.
Case	Box Deluxe	299.99kr.
System5		
mainboard	SUPER MEGA BYTE MF-1337	2199.99kr.
Graphics Card	MSI Radeon R7 370 Gaming 2GB GDDR5	1699.99kr.
ram	Kingston Valueram 4gb	299.99kr.
CPU	TYRIONS D	2599.99kr.
Case	MoneyMaker	399.99kr.
System6		
mainboard	ASRock H97M PRO4	1299.99kr.
Graphics Card	XFX Radeon R9 380 2GB GDDR5	2299.99kr.
ram	Kingston Valueram 8gb	399.99kr.
CPU	CrapCPU	1299.99kr.
Case	CrapCase	399.99kr.
System7		
mainboard	TRUMP Motherboard	1599.99kr.
Graphics Card	MSI Radeon R9 390X Gaming 8GB GDDR5	4699.99kr.
ram	Kingston KTA 4gb	299.99kr.
CPU	TRUMP PROCESSOR	1599.99kr.
Case	Box of Case	399.99kr.
System8		
mainboard	MSI H81M-P33	1099.99kr.
Graphics Card	XFx Radeon R9 FURY X 4GB HBM	7199.99kr.
ram	Kingston KTA 8gb	399.99kr.
CPU	EXCCCPCPU	5799.99kr.
Case	PCmasterCase	899.99kr.

Figure 3: The price-list.

```

System1
11499.99kr.
11269.99kr. per system by purchase of 2
11039.99kr. per system by purchase of 3
10809.99kr. per system by purchase of 4
10579.99kr. per system by purchase of 5
10349.99kr. per system by purchase of 6
10119.99kr. per system by purchase of 7
9889.99kr. per system by purchase of 8
9659.99kr. per system by purchase of 9
9429.99kr. per system by purchase of 10
9199.99kr. per system by purchase of 11
Type anything to return to main menu

```

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

```

Black Box 2002
Silver Surfer 1003
System2
System3
System4
System5
System6
System7
System8
System1
Type 'part' or 'system' followed by modelno or systemname. eg. 'part 1001'
part 2002
stock has been updated
Type anything to return to main menu

```

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.

MSI GeForce GTX 970 Gaming 4GB	8
MSI GeForce GTX 960 Gaming 4GB	6
MSI GeForce GTX 980 Ti Gaming 6GB	13
MSI Radeon R7 370 Gaming 2GB GDDR5	6
XFX Radeon R9 380 2GB GDDR5	4
MSI Radeon R9 390X Gaming 8GB GDDR5	9
XFX Radeon R9 FURY X 4GB HBM	11
Black Box	6
MaDaBard	20
MSI 970 GAMING	1
goodoldram 4gb	5
UNLIMITED POWER!	5
SUPA LEGEND CPU	12
goodoldram 2gb	2
MSI GeForce GTX 980 Gaming 4GB	11
Silver Surfer	4
Type anything to return to main menu	

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)

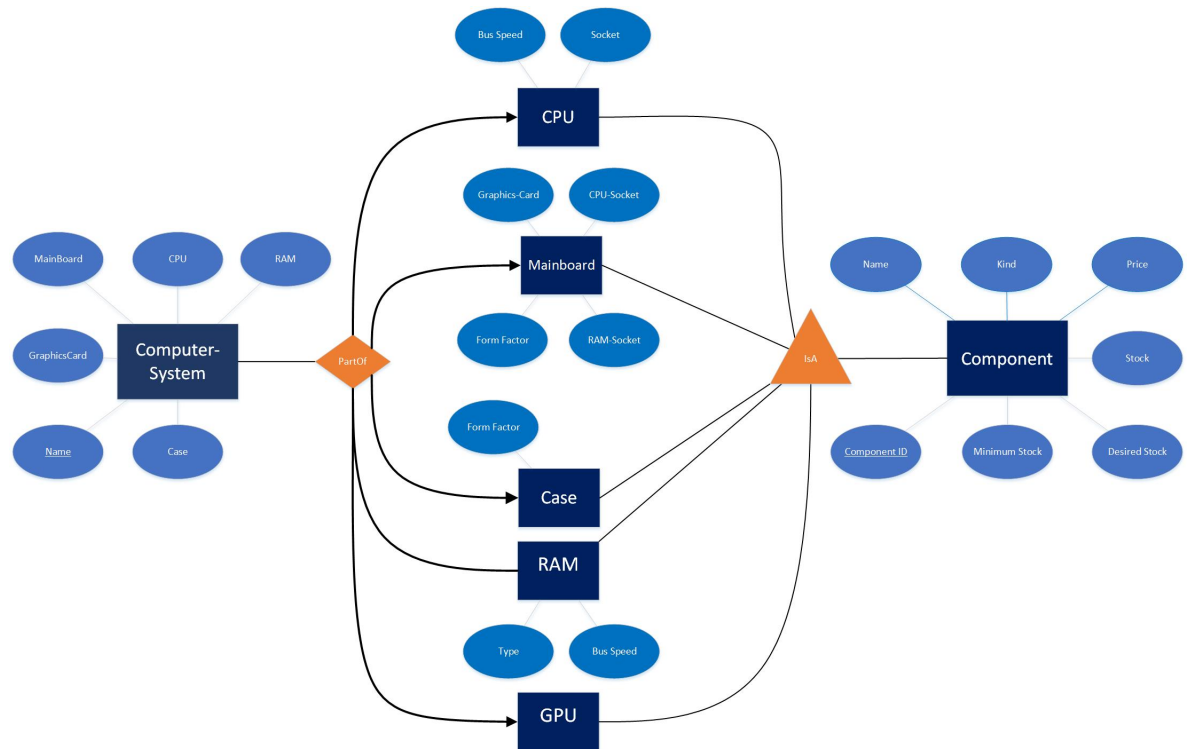


Figure 9: The ER model for the database. Note some names are slightly different from the actual database, as this was created before beginning the programming (eg. component id = modelno).

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
29    // component-table
30    public static void HowManyComponents(Connection con) throws
31    SQLException {
32        Statement st = con.createStatement();
33        ResultSet rs = st.executeQuery("SELECT * FROM component");
34        while (rs.next()) {
35            System.out.println("
36            -----
37            ");
38            System.out.println(rs.getString("modelno") + " " + rs.
39            getString("title") + " " + rs.getString("currentstock"));
40        }
41        rs.close();
42        st.close();
43        Scanner scanner = new Scanner(System.in);
44        System.out.println("Type anything to return to main menu");
45        if (scanner.hasNext() == true) {
46            StartMenu(con);
47        }
48    }
49    // Main menu. Prints options and takes integer as choice
50    public static void StartMenu(Connection con) throws SQLException {
51
52        Scanner scanner = new Scanner(System.in);
53
54        System.out.println("=====");
55        System.out.println("|      MENU SELECTION      |");
56        System.out.println("=====");
57        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
91      // computing amount buildable
92  public static void ShowCompSys(Connection con) throws SQLException {
93      Statement st = con.createStatement();
94      ResultSet rs = st.executeQuery("SELECT * FROM Computersystems");
95
96      while (rs.next()) {
97          System.out.println("
          -----
          ");
98          System.out.println(rs.getString("title") + " " + HowMany(con,
99              rs.getInt("cpu"), rs.getInt("ram"), rs.getInt("cases"),
100              rs.getInt("gpu"), rs.getInt("mainboard")));
101      }
102      rs.close();
103      st.close();
104      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,
109     int gpu, int mainboard) throws SQLException {
110
111     Statement st = con.createStatement();
112     Statement st1 = con.createStatement();
113     Statement st2 = con.createStatement();
114     Statement st3 = con.createStatement();
115     Statement st4 = con.createStatement();
116
117     int NumberOfCPU = 0;
118     int NumberOfRAM = 0;
119     int NumberOfCASE = 0;
120     int NumberOfMB = 0;
121     int NumberOfGPU = 0;
122
123     ResultSet cpuquery = st.executeQuery("SELECT currentstock FROM
124         Component WHERE modelno =" + cpu);
125     while (cpuquery.next()) {
126         NumberOfCPU = cpuquery.getInt("currentstock");
127     }
128     ResultSet ramquery = st1.executeQuery("SELECT currentstock FROM
129         Component WHERE modelno =" + ram);
130     while (ramquery.next()) {
131         NumberOfRAM = ramquery.getInt("currentstock");
132     }
133     ResultSet casequery = st2.executeQuery("SELECT currentstock FROM
134         Component WHERE modelno =" + Case);
135     while (casequery.next()) {
136         NumberOfCASE = casequery.getInt("currentstock");
137     }
138     ResultSet gpuQuery = st3.executeQuery("SELECT currentstock FROM
139         Component WHERE modelno =" + gpu);
140     while (gpuQuery.next()) {
141         NumberOfGPU = gpuQuery.getInt("currentstock");
142     }
143     ResultSet MbQuery = st4.executeQuery("SELECT currentstock FROM
144         Component WHERE modelno =" + mainboard);
145     while (MbQuery.next()) {
146         NumberOfMB = MbQuery.getInt("currentstock");
147     }
148
149     int lowest;
150     int[] numbers = {NumberOfCPU, NumberOfRAM, NumberOfCASE,
151         NumberOfMB, NumberOfGPU};
152     lowest = numbers[0];
153     for (int index = 1; index < numbers.length; index++)
154         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
159             computersystems");
160         ResultSet PriceListParts = st.executeQuery("SELECT * FROM
161             component ORDER BY kind");
162         while (PriceListParts.next()) {
163             // System.out.printf(rs.getString("title")+rs.getDouble
164             // (""));
165             System.out.println(PriceListParts.getString("kind") +
166                 PriceListParts.getString("title") + FinalPrice(
167                     PriceListParts.getDouble("price")) + "kr.");
168         }
169         while (PriceListSys.next()) {
170             if (HowMany(con, PriceListSys.getInt("cpu"), PriceListSys.
171                 getInt("ram"), PriceListSys.getInt("cases"),
172                 PriceListSys.getInt("gpu"), PriceListSys.getInt("
173                 mainboard")) > 0) {
174                 System.out.print(PriceListSys.getString("title") +
175                     System.lineSeparator()
176                     + FindName(con, PriceListSys.getInt("mainboard"
177                         ))
178                     + FindPrice(con, PriceListSys.getInt("mainboard
179                         ")) + "kr." + System.lineSeparator()
180                     + FindName(con, PriceListSys.getInt("gpu"))
181                     + FindPrice(con, PriceListSys.getInt("gpu")) +
182                     "kr." + System.lineSeparator()
183                     + FindName(con, PriceListSys.getInt("ram"))
184                     + FindPrice(con, PriceListSys.getInt("ram")) +
185                     "kr." + System.lineSeparator()
186                     + FindName(con, PriceListSys.getInt("cpu"))
187                     + FindPrice(con, PriceListSys.getInt("cpu")) +
188                     "kr." + System.lineSeparator()
189                     + FindName(con, PriceListSys.getInt("cases"))
190                     + FindPrice(con, PriceListSys.getInt("cases"))
191                     + "kr." + System.lineSeparator()
192                 );
193                 System.out.println((char) 27 + "[36m
194                     -----
195                     " + (char) 27 + "[0m");
196             }
197         }
198     }
199     Scanner scanner = new Scanner(System.in);
200     System.out.println("Type anything to return to main menu");
201     if (scanner.hasNext() == true) {
202         StartMenu(con);
203     }

```



```

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)

```

```

319         throws SQLException {
320             Statement st = con.createStatement();
321             ResultSet rs = st.executeQuery("SELECT * FROM component WHERE
322                 modelno=" + modelno);
323             while (rs.next()) {
324                 if ((rs.getInt("currentstock") <= rs.getInt("minimuminventory"
325                     ))) {
326                     return false;
327                 }
328             }
329             return true;
330         }
331         // Same as above only on all parts in a given system
332         private static boolean InStockSystem (Connection con,String
333             Systemname) throws SQLException {
334             Statement st = con.createStatement();
335             ResultSet rs = st.executeQuery("SELECT * FROM computersystems
336                 WHERE title =" + Systemname);
337             while (rs.next()){
338                 System.out.println("Action would violate minimuminventory");
339                 if (InStockPart(rs.getInt("cpu"),con)==false) {return false;}
340                 if (InStockPart(rs.getInt("mainboard"),con)==false) {return
341                     false;}
342                 if (InStockPart(rs.getInt("ram"),con)==false) {return false;}
343                 if (InStockPart(rs.getInt("graphicscard"),con)==false) {return
344                     false;}
345                 if (InStockPart(rs.getInt("cases"),con)==false) {return false
346                     ;}
347             }
348             return true;
349         }
350     }
351 }

```