Computer store Database System

Databasedesign- og programmering and DM505

Mark Jervelund
Mjerv15 - D2

IMADA

Instructor
Anders Bjørn Moeslund

April 6, 2016

Contents

Specification

The task in this project is to design and implement a database for a computer store. this should be implemented using a database, in this project postgres was used. at least name, type and price should be implemented and the different parts should have the following attributes:

CPU : socket and bus speed

RAM : Type and bus speed

Motherboard : CPU socket, Ram type, form factor and on-board graphics?

I moved the on-board graphics to the CPU and that it is where it is in modern computers,(since it has the questions mark after it.)

Case: form factor

Computer system: Name and catchy name:

All the parts in the system should have a Current stock, a allowed minimum and a preferred stock after restocking.

The selling price for a part is its price + 30 % and the selling price for a system is the price of all its part + 30 % rounded up to the nearest 99. if a buying is buying multiple systems there is a discount of 2 % per additional system up to a maximum discount of 20 %
The minium specification for the system is that it needs to be able to:

List all parts in system and their current stock

List all systems in the system and how many can be built from the current stock.

Price list, list all parts grouped by their kind, with sell price. as well as all computer systems that could be built from the current stock, including their components and selling price,

Price offer, give a price offer for a system and the quantity.

Sell a component and a computer system by updating the current stock.

Restocking list, including names, and how many of each item is needed for preferred level.

Design

I designed the database by making a ER diagram, It can be seen in the appendix.
The Database is in 3NF because,
Parts is a subset of computers
CPU is a subset of parts where the kind is CPU
Storage is a is a subset of parts where the kind is Storage
Motherboard is a subset of parts where the kind is MB
Ram is a subset of parts where the kind is ram
Graphics is a subset of parts where the kind is GPU
Case is a subset of parts where the kind is CASE
And there is no dependencies in the same schema.

I have the Parts or components table with the following columns:

Model, type, price, stock, refillstock, and producer.

There is constraints on stock, price, and refill stock
there is then a table for each part type, Ram, CPU, graphics, storage, and computercase, which contain all non shared common attributes. in addition there is a systems table that has the pre-built systems, this contains a model, name, cpu, ram, motherboard, storage, computer case, and graphics attributes for each system.
The interface is designed using command line java. which gets all the information from the postgres database.

Implementation

Debase implementation

The code for the Table looks like the following.

```
CREATE TABLE parts (
        model CHAR(30) PRIMARY KEY,
        type Char(5),
        price numeric(11,2) CONSTRAINT positive_price CHECK (price > 0),
        stock integer CONSTRAINT positive_stock CHECK (stock >= 0),
        refillstock integer CONSTRAINT refillstock_check CHECK (refillstock
            >= 0),
        producer CHAR(30)
        );
```

The sceme for parts is implemented with a model ID, type, price, stock, refillstock and producer.
It has constraints so you cant mistakenly enter a price that is negative.
Stock is implemented to minimum stock is implemented as a constraint that is 0, so you cant sell a part that is out of stock. and refill stock cant be below 0. to avoid user input error.

Each of the different type of parts has the own tables as they have non common attributes.. an example of this is the CPU table. The model name in each of the individual part tables references the main parts table with prices, producer, stock, and refillstock. where the individual tables store details, like speed, cores, storage, and other specification.

```
CREATE TABLE CPU (
        model CHAR(30)references parts (model),
        speed DECIMAL(4,2),
        Socket CHAR(10),
        cores integer,
        FSB integer,
        hasgrafics Boolean
        );
```

The data was inserted into the database using the insert function, this can be seen below.

```
INSERT INTO CPU (model, speed, socket, cores,FSB,hasgrafics)
VALUES
('CPU-I7-6700k', 4.2, 'LGA1151','8',160000,true),
```

All of the sql code for building the database can be seen in the file DB.SQL in the included zip folder.

Client implementation

The client is implemented in java. here I'll go over the different functions of the database. The user interface is using the command prompt, the input is accepted via scanners, and used using cases. an an example of this is:

```
while(keeprun==true){
        Scanner menu=new Scanner(System.in);
        String menupick=menu.next();

        switch(menupick){
        case"Listall":case"listall":case"la":case"LA":
        DBcalls.Printallparts(con);
        break;
        case"Listallsystems":case"LAS":case"las":case"listallsystems":
        DBcalls.listsystems(con);
        break;
        };
        }
```

This works by having the user input data into the scanner via system.in and then selecting the function in the DB calls class that does what the user requested.

Connection

For connection to the database the client uses the postgres driver and logs in using the information postgres and the password 12. if the connection to the database fails the program prints "Connection failed" which shows the user something went wrong when connecting to the database.

```
{
        String url="jdbc:postgresql://localhost:5432/postgres";
        String user="postgres";
        String password="12";
        Connection con=null;
        try{
        System.out.println("Connecting to database");
        con=DriverManager.getConnection(url,user,password);
        }catch(SQLException ex){
        Logger lgr=Logger.getLogger(DBcalls.class.getName());
        lgr.log(Level.WARNING,ex.getMessage(),ex);
        System.out.println("Connection failed");
        }
        }
```

List all parts

The list all functions selects stock and model from parts and orders them by model, and then prints them to the command promt.

```
public static void Printallparts(Connection con) {
        String query = "SELECT model,stock from parts Order by model;";
        try {
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery(query);
        System.out.println("Model                          | Stock");
        while (rs.next()) {
        System.out.print(rs.getString("model"));
        System.out.println("| " + rs.getString("stock"));
        }
        } catch (SQLException e) {
        e.printStackTrace();
```

```
        }
        }
```

list all parts with price

This is almost the same command as the one above but with this two following lines changed. This does so it also selects the price from parts, and prints it and adds 30 % to the price.

```
String query = "SELECT model, price from parts Order by model;";
System.out.println("    | " + ((rs.getInt("price"))*130/100));
```

List all systems

The list systems list the systems, their sell price, and how many can be built from the stock. it uses 3 functions, the system stock, the system price, and it self to print to needed information to the user.

First the client side program fetches system models, and names then it calls the system price, and system stock function that calculates the price, and how many systems can be built form the stock. when this is done it does this again for the next system while rs.next is true.

```
public static void listsystems(Connection con) {
        //list all systems and their prices.
        String query = "Select model, name from Computer";
        System.out.println("Model                          name
            build cost     price offer stock");
        try {
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery(query);
        while (rs.next()) {
        String model = rs.getString("model");
        System.out.println(model + rs.getString("name") + systemprice(con,
            model) + "      |     " + ((((systemprice(con, model)) * 13 / 10)
             / 100) * 100 + 99)+"    |   "+systemstock(con, model));
        }
        } catch (SQLException e) {
        e.printStackTrace();
        }
        }
```

the list stock function fetches the stock for the part needed for the system with the lowest stock and returns this which is then returned.

```
public static int systemstock(Connection con,String Part_iD) {
        String query = ("Select min(parts.stock) from Parts where model in
            " +
        "((select computer.cpu from computer where model similar to '%" +
            Part_iD + "%'), " +
        "(select computer.ram from computer where model similar to '%" +
            Part_iD + "%') , " +
        "(select computer.storage from computer where model similar to '%"
            + Part_iD + "%'), " +
        "(select computer.motherboard from computer where model similar to
            '%" + Part_iD + "%'), " +
        "(select computer.computercase from computer where model similar to
            '%" + Part_iD + "%'), " +
        "(select computer.graphics from computer where model similar to '%"
             + Part_iD + "%'))");
        int stock = 0;
        try {
```

```
        // System.out.println(query);
12      Statement st = con.createStatement();
        ResultSet rs = st.executeQuery(query);
        rs.next();
15      stock = rs.getInt(1);
        } catch (SQLException e) {
        e.printStackTrace();
18      }
        return stock;

21      }
```

The system price function fetches all the parts needed for a system, adds them to a list, and uses a for each loop to fetch and sum the price for all the parts needed for the system and returns this.

```
1   private static int systemprice(Connection con, String system) {
        //calculates the price of a system.
3       ArrayList<String> syspartlist = new ArrayList<String>();
        int price = 0;
        try {
6       Statement st = con.createStatement();
        String query = "Select * FROM  computer  WHERE model SIMILAR  TO '%
            " + system + "%';";
        ResultSet rs = st.executeQuery(query);
9       rs.next();
        syspartlist.add(rs.getString("cpu"));
        syspartlist.add(rs.getString("ram"));
12      syspartlist.add(rs.getString("Storage"));
        syspartlist.add(rs.getString("Motherboard"));
        syspartlist.add(rs.getString("computercase"));
15      syspartlist.add(rs.getString("graphics"));
        for (String Part_ID : syspartlist) {
        if (Part_ID != null) {
18      try {
        query = "Select price FROM  parts WHERE model SIMILAR  TO '%" +
            Part_ID + "%';";
        rs = st.executeQuery(query);
21      rs.next();
        int tempvalue = rs.getInt("price");
        price += tempvalue;
24      return price;
        }
```

price offer

The price offer works both for parts and systems, and parts, the parts and handed by the function itself, and the price for systems is handed by the systems price function. for systems it has a multiplier input, which calculates the discount that is up to 20 %.

```
1   public static void Priceoffer(Connection con) {
        //returns price for parts and systems.
3       String Part_ID = null;
        int multiplier = 0;
        Scanner keyboard = new Scanner(System.in);
6       System.out.println("Enter the model ID for the part you wish to get
            a price offer..");
        Part_ID = keyboard.next();
        if (Part_ID.contains("SYS-")) {
9       System.out.println("Enter the multiplier for how many systems you
            would like to buy..");
```

```
        Scanner multiplierinput = new Scanner(System.in);
        multiplier = multiplierinput.nextInt();
12      multiplier -= 1;
        double pricemultiplier = multiplier * 2;
        if (pricemultiplier > 20) {
15      pricemultiplier = 20;
        }
        Double systemprice = (((((systemprice(con, Part_ID)) * 13 / 10) /
            100) * 100 + 99) * (1 + multiplier) * (1 - (pricemultiplier /
            100)));
18      int finalprice = systemprice.intValue();
        System.out.println("Price offer for " + Part_ID + " is " +
            finalprice);
        return;
21      }
        try {
        Statement st = con.createStatement();
24      String query = "Select price FROM parts WHERE model SIMILAR  TO '%"
             + Part_ID + "%';";
        ResultSet rs = st.executeQuery(query);
        rs.next();
27      int price = ((rs.getInt("price") * 13 / 10));
        System.out.println("Price offer " +
        "for " + Part_ID + " is " + price);
30      } catch (SQLException e) {
        e.printStackTrace();
        }
33      }
```

Sell

The sell function is split into 2 functions, the sell function which handles parts, and the sell systems, which handles systems.

The sell function works by updating the stock of the parts to stock - 1, but it first checks if the part exist into the database, and if not it returns an error, or if the input matches multiple parts, like when entering HDD-530 which returns 3 different hard drives.

```
1   public static void Sellitem(Connection con) {
        String Part_ID = null;
3       Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the model ID for the part you wish to
            sell.");
        Part_ID = keyboard.next();
6       if (Part_ID.contains("SYS-")) {
        sellsystem(con, Part_ID);
        return;
9       }
        try {
        Statement st = con.createStatement();
12      ResultSet exsist = (st.executeQuery("SELECT COUNT(*) model From
            parts where model SIMILAR  TO '%" + Part_ID + "%';"));
        exsist.next();
        System.out.println(exsist.getInt(1));
15      if (exsist.getInt(1) == 1) {
        String query = "UPDATE parts SET Stock = Stock-1 WHERE model
            SIMILAR  TO '%" + Part_ID + "%';";
        st.executeUpdate(query);
```

```
18          System.out.println("Sold 1 x " + Part_ID);
            } else if (exsist.getInt(1) > 1) {
            System.out.println("Multiple containing that string found in
                database");
21          } else {
            System.out.println("Part does not exsist in Database");
            }}}
```

The sell system selects * from system where the model is similar to the requested part. these parts are then added to a list, which is used in a for each in list, it then calls the update parts set stock = stock −1where model similar to part_id from the list.

```
1   private static void sellsystem(Connection con, String system) {
            ArrayList<String> syspartlist = new ArrayList<String>();
3           int price = 0;
            try {
            Statement st = con.createStatement();
6           String query = "Select * FROM  computer  WHERE model SIMILAR  TO '%
                " + system + "%';";
            ResultSet rs = st.executeQuery(query);
            rs.next();
9           syspartlist.add(rs.getString("cpu"));
            syspartlist.add(rs.getString("ram"));
            syspartlist.add(rs.getString("Storage"));
12          syspartlist.add(rs.getString("Motherboard"));
            syspartlist.add(rs.getString("computercase"));
            syspartlist.add(rs.getString("graphics"));
15          for (String Part_ID : syspartlist) {
            if (Part_ID != null) {
            try{
18          query="UPDATE parts SET Stock = Stock-1 WHERE model SIMILAR  TO '%"
                +Part_ID+"%';";
            System.out.println("sold "+Part_ID);
            st.executeUpdate(query);
21          }}}}}
```

Restocking list

The restocking list works by selecting model, stock and refill stock from parts, and prints them if stock is lower than refill value.

```
1   public static void Restockinglist(Connection con) {
            //Prints a list of things to restock.
3           String query = "SELECT model, stock, refillstock from parts Order by
                model";
            try {
            Statement st = con.createStatement();
6           ResultSet rs = st.executeQuery(query);
            System.out.println("Model                          | In Stock | 
                preferred level | to restock");
            while (rs.next()) {
9           String model = rs.getString("model");
            int stock = rs.getInt("stock");
            int restock = rs.getInt("refillstock");
12          if (stock < restock){
            System.out.println(model+" | "+stock+"       | "+restock+"
                       | "+(restock-stock));
            }}}}
```

Restock

The restock function works almost the same as the restockinglist. this just adds the update statement to a list, and executes the update via a for each loop when it has checked all the parts.

```
1
        Restocklist.add("UPDATE parts SET stock = refillstock WHERE model
            SIMILAR  TO '%" + model + "%';");
3
        for (String updatequery : Restocklist) {
        st.executeUpdate(updatequery);
6       }
```

Custom system

The Custom system is implemented by the user first selecting a CPU, then Motherboard, Ram, storage, case, and then lastly is asked if they want graphics if the CPU the user choose doesn't have a included graphic chip. This is the code for the cpu, it prints all cpus in the system, and asks the users which one the user wants to use.

```
1   //choose the cpu
System.out.println("You can use the following CPU for this sysyem");
3       query="Select model from cpu;";
        ResultSet rscpu=st.executeQuery(query);
        int columns=rscpu.getMetaData().getColumnCount();
6       while(rscpu.next()){
        System.out.print(rscpu.getString(columns));
        }
9       System.out.println();
        Scanner keyboard=new Scanner(System.in);
        System.out.println("Enter the CPU that you wish to use.");
12      Part_CPU=keyboard.next();
        syspartlist.add(Part_CPU);
```

The middles stages include, motherboard, ram, case, storage which sql code that looks like the following.
Its from the 4 middle parts where the system fetches Motherboards, depending on what cpu was picked.
From ram where ramtype from motherboard and fsb from cpu matches.
any part from storage since they all match,
and a case where the formfactor matches that of the motherboard.

```
1   Select model from motherboard where motherboard.socket = (Select socket
        from cpu where model SIMILAR TO '%"+Part_CPU+"%');
    Select model from ram where (ram.ramtype = (Select ramtype from motherboard
         where model SIMILAR TO '%"+Part_MB+"%')AND ram.fsb = (Select fsb from
        cpu where model Similar to '%"+Part_CPU+"%'));
3   Select model from storage;
    Select model from computercase where formfactor =(SELECT formfactor FROM
        motherboard where model similar to '%"+Part_MB+"%');
```

Before the graphics card can be selected the program checks if the system has on-board graphics, if it has, it asks the user if the user wants to include a graphics card anyway. this is done using a scanner and 3 cases, yes, no, and repick if the users picked a invalid choice.

```
1   if(rshasgrafics.getBoolean("hasgrafics")){
        System.out.println("The system you are designing have onboard
            grafics , do you want to install a one anyway? y/n");
3       while(keeprun==true){
        Scanner menu=new Scanner(System.in);
        String menupick=menu.next();
6       // compares input from system in to the cases that repesent a case
            in use.
```

```
            switch(menupick){
            case"y":case"yes":
 9          hasgrafics=true;keeprun=false;break;
            case"n":case"no":
            hasgrafics=false;keeprun=false;break;
12          default:
            System.out.println("Invalid choice, please pick again.");
            break;}}
15          }else{
            hasgrafics=true;
            }
```

When the user has chosen rather to have a graphics card or not the programs asks the user to enter a model name for the graphics card the users decides to use. when this is done the system compiles a price for the system using a list of parts made when picking the prices, and calculates the price for the parts using a for each loop.

```
 1  int price=0;
          for(String Part_ID:syspartlist){
 3            if(Part_ID!=null){
              try{
                  query="Select price FROM  parts WHERE model SIMILAR  TO '%"
                      +Part_ID+"%';";
 6                System.out.println("getting price for "+Part_ID);
                  rs=st.executeQuery(query);
                  rs.next();
 9                int tempvalue=rs.getInt("price");
                  price+=tempvalue;
              }catch(SQLException e){e.printStackTrace();System.out.println("
                  there was a problem with a Part");
12        }}}
          System.out.println(((price*13/10)/100)*100+99);
```

Testing

I have tested all the functions of the program and the results can be seen here.

List all parts

list all parts with price returns the information like it should, i have included a small part of it here.
Model | Price
CASE-mini | 520
CASE-supreme | 1040
CPU-E5-1320-v3 | 4618
CPU-E5-2999-v3 | 48643
CPU-FX-4300 | 783

List all parts with stock

list all parts with stock returns (i have only include a small part)
Model | Stock
CASE-mini | 20
CASE-supreme | 19
CPU-E5-1320-v3 | 11
CPU-E5-2999-v3 | 12
CPU-FX-4300 | 12

CPU-FX-6300 | 12
CPU-FX-8350 | 12

List all systems

List systems returns, (i have only include a small part)
Model name build cost price offer stock
SYS-1 Blzing Firestorm 18618 | 24299 | 7
SYS-2 Starstruck 5699 | 7499 | 7

Price offer

i entered SYS-8 and a multiplier of 5 discount of 8 % which returned 59795
To check that the output is correct i calculated what i should b
(12999*5)*0.92 = 59795.4
From that i can conclude that the function returns the correct price using the formulas i designed.

Sell

I tested this by first checking stock of CASE-mini which was 20, i then sold 1. and checked stock again and it was 19 therefor i can confirm that selling of a single part works. i then tested this with a system.
I did this by checking what parts are needed and writing down the stock. then i sold the system i checked the parts for, and then checked if the stock of these parts fell by one when selling a system.

Restocking list

I checked that the restocking list worked by printing the restocking list after selling some parts, and noting down how many i sold, and confirmed that it printed the correct amount.

Restock

I checked by first running the restocking list since i confirmed it worked, then i ran the restocking function, and the last i ran the restocking list function to confirm that it didn't print anything.

Custom system

I tested the custom system using CPU-I7-6700k as the cpu, MB-ASUS-Z170K as the motherboard, RAM-Kingston-DDR4-16gb as ram, HDD-530-256 as storage, and CASE-supreme as the case. and by calculating the prices in hand i can confirm that this system works as it should.

User Manuel

You run the Client by either running the run.bat file (on windows in the included zip file) or using the command java -cp postgresql-9.4-1201.jdbc4.jar; Client, and the command java -cp postgresql-9.4-1201.jdbc4.jar:. Client on linux.
The commands and what you can do is listed when you run the problem, it should just be noted that some of the functions are very picky about their input and its case sensitive.

Conclusion

From the testing i have done the program meets the demands that have been met for the project, but it has some problems, like that the inputs are case sensitive, and that some of the input in the custom system, isn't as well implemented as it could have been.

Appendix

ER-Diagram