DM505 Exam project

Managing a computer hardware stock

By Mads Petersen

Table of Contents

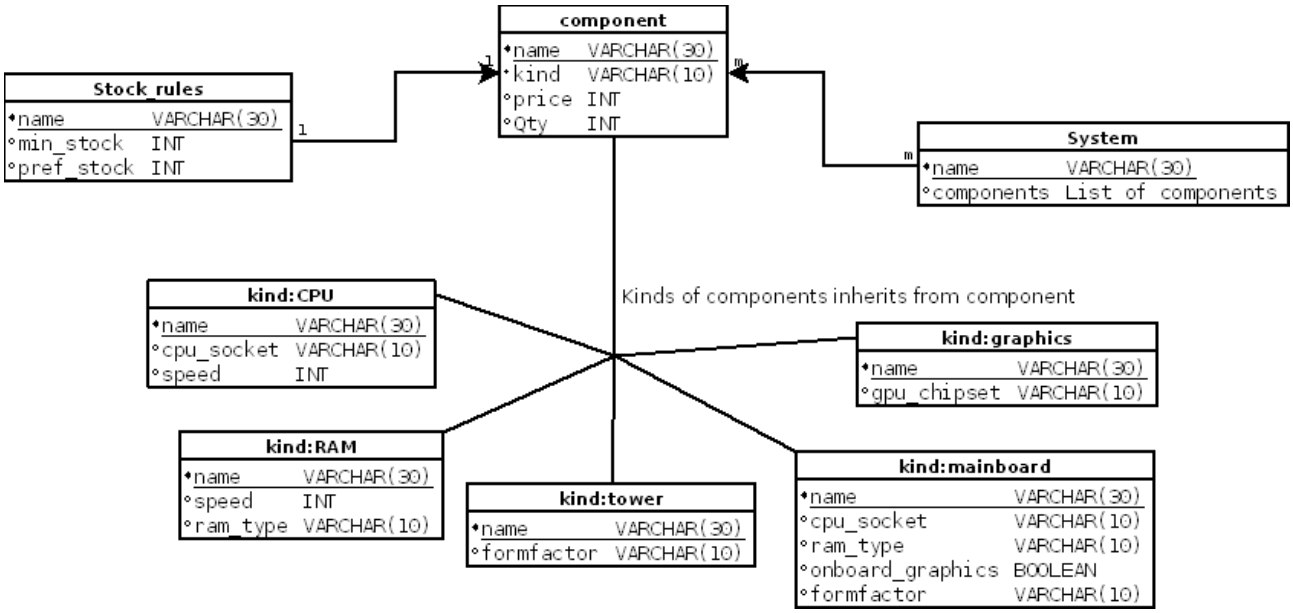
ER model and implementation

ER model

The objects to be modelled are:

- *component: name, kind, price.*
- *kind being on of: CPU, RAM, graphics card, mainboard or case(renamed to tower).*
- *CPU components must have the following attributes: socket, bus speed.*
- *RAM: type, bus speed.*
- *Mainboard: CPU socket, RAM type, onboard graphics?, form factor.*
- *Tower: form factor.*
- *computer system, this models a collection of components into a functioning system and must have a name and a list of components.*
- *Current stock of each component.*
- *Minimum and preffered numbers of component in stock.*

The ER model devoloped to satisfy this:



Keys are underlined.

To model the components and their attributes, one entity named component is used. It has the attributes: *kind, price and qty(short for quantity)* these are the attributes that all components share. Then there are 5 additional entities called: *CPU, RAM, tower, mainboard and graphics.* If a component is of kind CPU, it also has an entry in the CPU entity, if it is of kind RAM, it also has an entry in the RAM entity and so forth. The entries share the attribute *name* as common key. So e.g. If a new CPU is to be entered into the system, its' name, kind(in this case CPU) and the current quantity is first entered into the component entity, then next the specifics of the CPU are entered into the CPU entity, its' socket and speed. If all of these attributes where to be part of the same entity then it would have a lot of empty space. E.g. Whenever a component was entered it would have the attributes of all the other kinds that it does not belong to, but they would be empty.

The bus speed for RAM was dropped in this model since it is not really relevant since it need not match as long as the type of RAM is supported by the mainboard the same is true for CPU bus speed as long as the socket is supported by the mainboard. But if it was still required it could easily

be implemented in the same way as cpu_socket, ramtype and formfactor.

The *system* entity has a *name* attribute, this is the name of the system and it has the list of names of the components that make up the system. It is essentially an alias for a set of components.
The relationship of *system* to *component* is such that 1 system must have at least 4 components*(tower, RAM, CPU and mainboard)* but the *graphics* attribute may be null if the *mainbaord* has onboard graphics. Each *component* may be part of many systems or none.

To implement the preferred and minimum component quantities the relation *stock_rules* is used. If a component particular stock rules(i.e. min. or preferred quantities) then it will have an entry in *stock_rules* containing the name of the component and the minimum and/or preferred quantity.

The relation of *stock_rules* to *component* is such that each component can have a maximum of 1 stock rule tied to it, but might have 0 if no rules exsists for a particular component. Each *stock_rule* MUST have exactly 1 component tied to it, even if the rules for 2 components are exactly alike, they must have seperate entries.

This model is in 3NF since the only FDs are the ones with *name* on the left. There are no other attributes or combination of attributes that define a component than its' name. For *component*, the *kind, qty and price* may be the same for any number of components and not necessarily have the same name(i.e. *kind, qty, price > name* is not true, only *name > kind, qty, price*).
The same is true for *CPU, RAM, tower, mainboard and graphics*

name > cpu_socket, speed NOT *cpu_socket, speed > name.*

name > speed, ram_type NOT *speed, ram_type > name.*

name > formfactor NOT *formfactor > name.*

name > cpu_socket, ram_type, onboard_graphics, formfactor
NOT *cpu_socket, ram_type, onboard_graphics, formfactor > name.*

name > gpu_chipset NOT *gpu_chipset > name.*

Respectively.

It also holds for *stock_rules, name > min_qty, pref_qty* NOT *min_qty, pref_qty > name.*

And while it is most likely the case that in the system entity:

component list > name. There is not actually a reason that to systems with the exact same components could not be sold under different names, therefore it is not necessarily true. But *name > component list* is always true. Therefore, since the left side of all FDs consists of *name* and name is the key of every entity, it must be in third normal form.

Implementation

The implmentation of the ER model into postgreSQL consists of 8 tables:

component, cpu, graphics, mainboard, ram, stock_rules, system and tower.

With the colums and attributes as stated in the ER model. The component list of the system entity is implemented as five seperate columns, one for each of *CPU, RAM, mainboard, tower and graphics.* Each containing the name of the component that fills that role. Each is NOT NULL, except for *graphics* since there might be no graphics card in the system if the mainboard has onboard graphics.

To maintain the specification:
Entries in CPU, RAM, mainboard, tower or graphics must exsist in the component table.

Foreign key constraints exists, in each of the 5 kinds of component tables, referencing name in the

component table. Therefore entries that does not first exist in the component table cannot be inserted into any of those 5.

To maintain the specification:

A system must have at least: a CPU, RAM, mainboard and a tower. And optionally a graphics card depending on whether the mainboard specified has onboard graphics card or not and the cpu socket, ram type and form factor must match.

Triggers on the system table along with functions are used. The triggers used on the system table are as follows:

```
CREATE TRIGGER iscpusocketcompatible
  BEFORE INSERT OR UPDATE
  ON system
  FOR EACH ROW
  EXECUTE PROCEDURE cpucompatible();
```

Identical triggers exists for each of the functions: *formcompatible(), graphicscompatible(), ramcompatible().* The functions triggered looks like:

```
CREATE OR REPLACE FUNCTION cpucompatible()
  RETURNS trigger AS $$

BEGIN
IF (SELECT getcpusocket(NEW.CPU)) <> (SELECT getmainsocket(NEW.mainboard)) THEN
        RAISE EXCEPTION 'CPU and mainboard CPU sockets must match';
ELSE
        return NEW;

END IF;
END;
$$ LANGUAGE plpgsql;
```

getcpusocket(varchar) and *getmainsocket(varchar)* does almost the same thing:

```
CREATE OR REPLACE FUNCTION getcpusocket(newcpu character varying)
  RETURNS character varying AS $$
declare
        mine varchar;
BEGIN
SELECT cpu_socket into mine FROM CPU WHERE CPU.name = newCpu;
return mine;
END;
$$ LANGUAGE plpgsql;
```

getcpusocket(varchar) fetches the cpu socket of the component supplied as parameter. *getmainsocket(varchar)* does mostly the same thing only it fetches the cpu socket of the mainboard. These 2 values are then compared, if they are different i.e. the socket supported by the mainboard and the socket required by the cpu are different, then an exception is raised saying that they need to match. If they are not different, then the insert or update is simply allowed to continue.

formcompatible(), graphicscompatible(), ramcompatible() does almost the same thing, except they compare *formfactor, ram_type* and *graphicscompatible()* checks that, if the graphics value of a row inserted or update in the *system* table is null, and the mainboard does NOT have onboard graphics, then an exception is raised.

Combined, these triggers and functions ensure that, before a row is inserted or updated in the *system* table, it is checked that the components functions together. There are also foreign key constraints on the *system* table that checks that a component supplied as CPU must exist in the *CPU* table and likewise for RAM, tower, mainboard and graphics. And since it does not need to exist if the value is null this works just fine for the graphics column.

In most other tables values(except for keys) are allowed to be NULL since there aren't any specifications stating the opposite. Though in the case of *formfactor, ram_type* and *cpu_socket* the mainboard must also have NULL in these columns if they are to be part of the same system.

Testing

List of all components:

Can be seen by starting the java application in the top part(see manual). Can also be extracted directly from the database with the statement:

SELECT name,qty

FROM component;

List of all computer systems:

Can be seen in the java application in the bottom part(see manual). Can also be extracted directly from the database with the statement:

SELECT system.name, MIN(component.qty)

FROM system, component

WHERE (system.ram = component.name OR system.cpu = component.name OR system.tower = component.name OR system.mainboard = component.name OR system.graphics = component.name)

GROUP BY system.name;

Price list:

Components and prices can be seen in the java application top window, systems and selling prices can be seen in the bottom window. The systems list can be obtained directly from the database with the statement:

*SELECT system.name, ((((CAST(SUM(component.price) * 1.3 AS INTEGER) / 100) + 1) * 100) -1, system.ram, system.cpu, system.mainboard, system.tower, system.graphics, MIN(component.qty)*

FROM system, component

WHERE (component.name IN(system.ram, system.cpu, system.tower, system.mainboard, system.graphics))

GROUP BY system.name;

The components list with:

*SELECT name, kind, price * 1.3*

FROM component

ORDER BY kind;

Price offer:

Can be seen in the java application by selecting the appropriate system in the drop-down box in the bottom and inputting the desired number. Or it could be obtained directly from the database with a function that takes the system to buy and the number of systems as parameters:

```
CREATE OR REPLACE FUNCTION getprice(n integer, sys varchar)
  RETURNS numeric AS $$
declare
        t integer;
BEGIN
SELECT ((((CAST(SUM(component.price) * 1.3 AS INTEGER) / 100) + 1) * 100) -1 into t
FROM system, component
WHERE (component.name IN(system.ram, system.cpu, system.tower, system.mainboard,
system.graphics) AND system.name = sys)
GROUP BY system.name;


IF (n = 1) THEN
        return t;
ELSIF (n > 1 AND n < 11) THEN
        return (t * n * (1-(0.02*(n-1))));
ELSE
        return t * n * 0.8;
END IF;
END;
$$ LANGUAGE plpgsql;
```

Sell component or computer system:

Can be done in the java application by pressing the ”Sell component” button, or the ”sell indicated number of systems” button. Can also be done directly through SQL by updating relevant quantities.

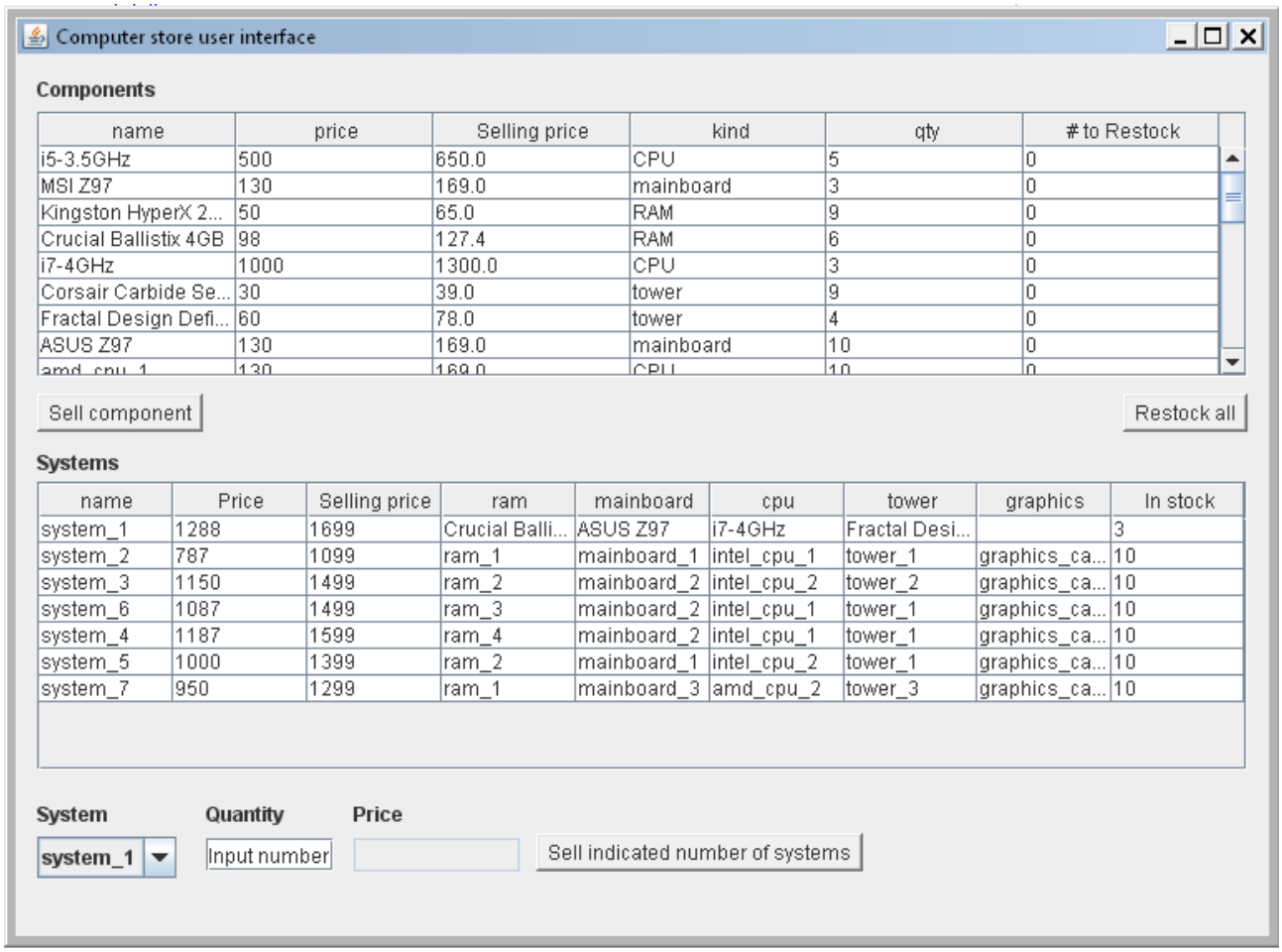
Restocking list:

Can be seen in the java application in the column named ”# to restock” or can be extracted from the database directly with the statement:

```
SELECT component.name, (stock_rules.pref_qty - component.qty)
FROM component, stock_rules
WHERE component.name = stock_rules.name AND (stock_rules.pref_qty - component.qty) > 0
```

Java application user manual

The java application, when started looks like this:



The top area under the heading ”Components” shows all the components in the database along with their selling prices, quantity and the number of components need to restock to preferred given in the stock_rules table. The components shown can be sorted by name or by kind by pressing the column heading for either of those columns. Sorting by any column that contains numbers does not work properly. If the button ”Sell component” is pressed while the name of a component is maked in the ”name” column, the quantity of that component will decrese by 1. If the ”Restock all” button is pressed then the quantity of all components will be returned to their preferred value.

The bottom area under the heading ”Systems” shows all the systems in the databse, along with their selling prices, how many of the given system is in stock at the moment, and the components that make it up. At the very bottom, under the table showing systems, is a box with all the systems in the database, one can be chosen here and a quantity input in the field to the right and the price for that amount of systems is shown in the price field. If the button ”Sell indicated number of systems” is pressed then quantity of the components for that system is updated accordingly.

Notes:

The java application connects to a database located at localhost, port number; 5432, database name:computer store, username:postgres, password: 123.

Appendix

SQL code

Database creation code:

```
CREATE DATABASE "computer store"
  WITH OWNER = postgres
      ENCODING = 'UTF8'
      TABLESPACE = pg_default
      LC_COLLATE = 'Danish_Denmark.1252'
      LC_CTYPE = 'Danish_Denmark.1252'
      CONNECTION LIMIT = -1;

COMMENT ON DATABASE "computer store"
  IS 'Database for use with project';
```

Component table

```
CREATE TABLE component
(
  name character varying(30) NOT NULL,
  price integer,
  kind character varying(10),
  qty integer,
  CONSTRAINT component_pkey PRIMARY KEY (name)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE component
  OWNER TO postgres;
```

CPU table

```
CREATE TABLE cpu
(
  name character varying(30) NOT NULL,
  cpu_socket character varying(20),
  cpu_speed integer,
  CONSTRAINT cpu_pkey PRIMARY KEY (name),
  CONSTRAINT "nameIsInComponent" FOREIGN KEY (name)
      REFERENCES component (name) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)
WITH (
  OIDS=FALSE
);
ALTER TABLE cpu
  OWNER TO postgres;
```

graphics table

```
CREATE TABLE graphics
```

```
(
  name character varying(30) NOT NULL,
  type character varying(10),
  CONSTRAINT graphics_pkey PRIMARY KEY (name),
  CONSTRAINT "nameIsInComponent" FOREIGN KEY (name)
      REFERENCES component (name) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)
WITH (
  OIDS=FALSE
);
ALTER TABLE graphics
  OWNER TO postgres;
```

mainboard table

```
CREATE TABLE mainboard
(
  name character varying(30) NOT NULL,
  formfactor character varying(20),
  cpu_socket character varying(20),
  ram_type character varying(20),
  onboard_graphics boolean,
  CONSTRAINT mainboard_pkey PRIMARY KEY (name),
  CONSTRAINT "nameIsInComponent" FOREIGN KEY (name)
      REFERENCES component (name) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)
WITH (
  OIDS=FALSE
);
ALTER TABLE mainboard
  OWNER TO postgres;
```

RAM table

```
CREATE TABLE ram
(
  name character varying(30) NOT NULL,
  ram_type character varying(10),
  speed integer,
  CONSTRAINT ram_pkey PRIMARY KEY (name),
  CONSTRAINT "nameIsInComponent" FOREIGN KEY (name)
      REFERENCES component (name) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)
WITH (
  OIDS=FALSE
);
ALTER TABLE ram
  OWNER TO postgres;
```

stock_rules table

```
CREATE TABLE stock_rules
(
  name character varying(30) NOT NULL,
  min_qty integer,
  pref_qty integer,
  CONSTRAINT stock_rules_pkey PRIMARY KEY (name),
  CONSTRAINT "nameIsInComponent" FOREIGN KEY (name)
      REFERENCES component (name) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)
WITH (
  OIDS=FALSE
);
ALTER TABLE stock_rules
  OWNER TO postgres;
```

system table

```
CREATE TABLE system
(
  name character varying(30) NOT NULL,
  ram character varying(30) NOT NULL,
  mainboard character varying(30) NOT NULL,
  cpu character varying(30) NOT NULL,
  tower character varying(30) NOT NULL,
  graphics character varying(30),
  CONSTRAINT system_pkey PRIMARY KEY (name),
  CONSTRAINT "cpuIsInCPU" FOREIGN KEY (cpu)
      REFERENCES cpu (name) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT "graphicsIsInGraphics" FOREIGN KEY (graphics)
      REFERENCES graphics (name) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT "mainboardIsInMainboard" FOREIGN KEY (mainboard)
      REFERENCES mainboard (name) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT "ramIsInRam" FOREIGN KEY (ram)
      REFERENCES ram (name) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT "towerIsInTower" FOREIGN KEY (tower)
      REFERENCES tower (name) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)
WITH (
  OIDS=FALSE
);
ALTER TABLE system
  OWNER TO postgres;

-- Trigger: iscpusocketcompatible on system
```

```
-- DROP TRIGGER iscpusocketcompatible ON system;

CREATE TRIGGER iscpusocketcompatible
  BEFORE INSERT OR UPDATE
  ON system
  FOR EACH ROW
  EXECUTE PROCEDURE cpucompatible();

-- Trigger: isformfactorcompatible on system

-- DROP TRIGGER isformfactorcompatible ON system;

CREATE TRIGGER isformfactorcompatible
  BEFORE INSERT OR UPDATE
  ON system
  FOR EACH ROW
  EXECUTE PROCEDURE formcompatible();

-- Trigger: isgraphicscompatible on system

-- DROP TRIGGER isgraphicscompatible ON system;

CREATE TRIGGER isgraphicscompatible
  BEFORE INSERT OR UPDATE
  ON system
  FOR EACH ROW
  EXECUTE PROCEDURE graphicscompatible();

-- Trigger: isramcompatible on system

-- DROP TRIGGER isramcompatible ON system;

CREATE TRIGGER isramcompatible
  BEFORE INSERT OR UPDATE
  ON system
  FOR EACH ROW
  EXECUTE PROCEDURE ramcompatible();
```

tower table

```
CREATE TABLE tower
(
  name character varying(30) NOT NULL,
  formfactor character varying(10),
  CONSTRAINT tower_pkey PRIMARY KEY (name)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tower
  OWNER TO postgres;
```

Functions

getcpusocket(character varying)

```
-- Function: getcpusocket(character varying)

-- DROP FUNCTION getcpusocket(character varying);

CREATE OR REPLACE FUNCTION getcpusocket(newcpu character varying)
  RETURNS character varying AS
$BODY$
declare
        mine varchar;
BEGIN
SELECT cpu_socket into mine FROM CPU WHERE CPU.name = newCpu;
return mine;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION getcpusocket(character varying)
  OWNER TO postgres;
```

getmainformfactor(character varying)

```
-- Function: getmainformfactor(character varying)

-- DROP FUNCTION getmainformfactor(character varying);

CREATE OR REPLACE FUNCTION getmainformfactor(newmain character varying)
  RETURNS character varying AS
$BODY$
declare
        mine varchar;
BEGIN
SELECT formfactor into mine FROM mainboard WHERE mainboard.name = newMain;
return mine;
END;
 $BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION getmainformfactor(character varying)
  OWNER TO postgres;
```

getmainramtype(character varying)

```
-- Function: getmainramtype(character varying)

-- DROP FUNCTION getmainramtype(character varying);

CREATE OR REPLACE FUNCTION getmainramtype(newmain character varying)
  RETURNS character varying AS
$BODY$
declare
```

```
        mine varchar;
BEGIN
SELECT ram_type into mine FROM mainboard WHERE mainboard.name = newMain;
return mine;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION getmainramtype(character varying)
  OWNER TO postgres;
```

getmainsocket(character varying)

```
-- Function: getmainsocket(character varying)

-- DROP FUNCTION getmainsocket(character varying);

CREATE OR REPLACE FUNCTION getmainsocket(newmain character varying)
  RETURNS character varying AS
$BODY$
declare
        mine varchar;
BEGIN
SELECT cpu_socket into mine FROM mainboard WHERE mainboard.name = newMain;
return mine;
END;
 $BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION getmainsocket(character varying)
  OWNER TO postgres;
```

getonboardgraphics(character varying)

```
-- Function: getonboardgraphics(character varying)

-- DROP FUNCTION getonboardgraphics(character varying);

CREATE OR REPLACE FUNCTION getonboardgraphics(newmain character varying)
  RETURNS boolean AS
$BODY$
declare
        mine boolean;
BEGIN
SELECT onboard_graphics into mine FROM mainboard WHERE mainboard.name = newMain;
return mine;
END;
 $BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION getonboardgraphics(character varying)
  OWNER TO postgres;
```

getramtype(character varying)

```
-- Function: getramtype(character varying)

-- DROP FUNCTION getramtype(character varying);

CREATE OR REPLACE FUNCTION getramtype(newram character varying)
  RETURNS character varying AS
$BODY$
declare
        mine varchar;
BEGIN
SELECT ram_type into mine FROM ram WHERE ram.name = newRam;
return mine;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION getramtype(character varying)
  OWNER TO postgres;
```

gettowerformfactor(character varying)

```
-- Function: gettowerformfactor(character varying)

-- DROP FUNCTION gettowerformfactor(character varying);

CREATE OR REPLACE FUNCTION gettowerformfactor(newtower character varying)
  RETURNS character varying AS
$BODY$
declare
        mine varchar;
BEGIN
SELECT formfactor into mine FROM tower WHERE tower.name = newTower;
return mine;
END;
 $BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION gettowerformfactor(character varying)
  OWNER TO postgres;
```

Trigger functions

cpucompatible()

```
-- Function: cpucompatible()

-- DROP FUNCTION cpucompatible();

CREATE OR REPLACE FUNCTION cpucompatible()
  RETURNS trigger AS
$BODY$

BEGIN
IF (SELECT getcpusocket(NEW.CPU)) <> (SELECT getmainsocket(NEW.mainboard)) THEN
        RAISE EXCEPTION 'CPU and mainboard CPU sockets must match';
ELSE
        return NEW;

END IF;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION cpucompatible()
  OWNER TO postgres;
```

formcompatible()

```
-- Function: formcompatible()

-- DROP FUNCTION formcompatible();

CREATE OR REPLACE FUNCTION formcompatible()
  RETURNS trigger AS
$BODY$

BEGIN
IF (SELECT gettowerformfactor(NEW.tower)) <> (SELECT getmainformfactor(NEW.mainboard))
THEN
        RAISE EXCEPTION 'mainboard and tower form factor must match';
ELSE
        return NEW;

END IF;
END;

$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION formcompatible()
```

```
  OWNER TO postgres;
```

graphicscompatible()

```
-- Function: graphicscompatible()

-- DROP FUNCTION graphicscompatible();

CREATE OR REPLACE FUNCTION graphicscompatible()
  RETURNS trigger AS
$BODY$

BEGIN
IF ((SELECT getonboardgraphics(NEW.mainboard)) = false) AND (NEW.graphics IS NULL)
THEN
        RAISE EXCEPTION 'If there is no onboard graphics card one must be specified';
ELSE
        return NEW;

END IF;
END;

$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION graphicscompatible()
  OWNER TO postgres;
```

ramcompatible()

```
-- Function: ramcompatible()

-- DROP FUNCTION ramcompatible();

CREATE OR REPLACE FUNCTION ramcompatible()
  RETURNS trigger AS
$BODY$

BEGIN
IF (SELECT getramtype(NEW.ram)) <> (SELECT getmainramtype(NEW.mainboard)) THEN
        RAISE EXCEPTION 'mainboard and ram types must match';
ELSE
        return NEW;

END IF;
END;

$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

```
ALTER FUNCTION ramcompatible()
  OWNER TO postgres;
```

JAVA code

Computer_store.java

```
package computer_store;


/**
 *
 * @author ShameOnU
 */
public class Computer_store
{
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {

        /* Create and display gui */
        java.awt.EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                SQLHandler handler = new SQLHandler();
                handler.init();
                new GUI(handler).setVisible(true);
            }
        });
    }
}
```

Error.java

```
package computer_store;


/**
```

```
 *
 * @author ShameOnU
 */
public class Error extends javax.swing.JFrame {

    /**
     * Creates new form Error
     * @param msg
     */
    public Error(String msg)
    {
        initComponents();
        setText(msg);
        jTextPane1.setBounds((this.getX()+((this.getWidth()/2)-msg.length()*5)), ((this.getY()/2)+10),
msg.length()*5, 50);

    }

    public void setText(String text)
    {
        jTextPane1.setText(text);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jButton1 = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextPane1 = new javax.swing.JTextPane();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
```

```
        setTitle("Error");
        setAlwaysOnTop(true);
        setBounds(new java.awt.Rectangle(250, 250, 300, 300));
        setModalExclusionType(java.awt.Dialog.ModalExclusionType.APPLICATION_EXCLUDE);
        setResizable(false);

        jButton1.setText("OK");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jTextPane1.setEditable(false);
        jTextPane1.setFocusable(false);
        jScrollPane1.setViewportView(jTextPane1);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(156, 156, 156)
                        .addComponent(jButton1))
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap(195, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(132, Short.MAX_VALUE)
```

```
            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jButton1)
            .addGap(115, 115, 115))
        );

        pack();
    }// </editor-fold>

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextPane jTextPane1;
    // End of variables declaration
}
```

GUI.java

```
package computer_store;

import java.sql.SQLException;
import org.apache.commons.lang3.StringUtils;
import java.text.DecimalFormat;

/**
 *
 * @author ShameOnU
 */
public class GUI extends javax.swing.JFrame
{
    SQLHandler handler;
    DecimalFormat df = new DecimalFormat("#.##");
    /**
```

```
     * Creates new form GUI
     */
    public GUI(SQLHandler handler) {
        this.handler = handler;
        initComponents();
        updateTables();
    }

    private void updateTables()
    {
        fillSysTable(jTable2,handler.getAllSystems());
        fillCompTable(jTable1,handler.getAllComponents());
        jTable1.moveColumn(5, 2);
        jTable2.moveColumn(6, 1);
        jTable2.moveColumn(7, 2);
    }




    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jScrollPane1 = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();
        fillTable(jTable1,handler.getAllComponents());
        jScrollPane2 = new javax.swing.JScrollPane();
        jTable2 = new javax.swing.JTable();
        fillSysTable(jTable2,handler.getAllSystems());
        jLabel1 = new javax.swing.JLabel();
```

```
jLabel2 = new javax.swing.JLabel();
jComboBox1 = new javax.swing.JComboBox();
jTextField1 = new javax.swing.JTextField();
jTextField2 = new javax.swing.JTextField();
jLabel4 = new javax.swing.JLabel();
jLabel5 = new javax.swing.JLabel();
jLabel6 = new javax.swing.JLabel();
button3 = new java.awt.Button();
button4 = new java.awt.Button();
button5 = new java.awt.Button();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Computer store user interface");
setBounds(new java.awt.Rectangle(250, 250, 750, 550));
setMinimumSize(new java.awt.Dimension(750, 550));
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent evt) {
        formWindowClosing(evt);
    }
});

jScrollPane1.setMaximumSize(jScrollPane1.getPreferredSize());
jScrollPane1.setPreferredSize(jScrollPane2.getPreferredSize());

jTable1.setAutoCreateRowSorter(true);
jTable1.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {
        {null, null, null, null},
        {null, null, null, null},
        {null, null, null, null},
        {null, null, null, null}
    },
    new String [] {
        "Title 1", "Title 2", "Title 3", "Title 4"
    }
));
jTable1.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_ALL_COLUMNS);
```

```
        jTable1.setCellSelectionEnabled(true);
        jScrollPane1.setViewportView(jTable1);

        jScrollPane2.setMaximumSize(jScrollPane2.getPreferredSize());

        jTable2.setAutoCreateRowSorter(true);
        jTable2.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null}
            },
            new String [] {
                "Title 1", "Title 2", "Title 3", "Title 4"
            }
        ));
        jScrollPane2.setViewportView(jTable2);

        jLabel1.setText("Components");

        jLabel2.setText("Systems");

        try
        {
            jComboBox1.setModel(new
javax.swing.DefaultComboBoxModel(handler.resultSetToArray(handler.getAllSystemNames())));
        }
        catch(Exception e)
        {
            System.out.print(e);
            jComboBox1.setModel(new javax.swing.DefaultComboBoxModel(new String[] {"Error"}));
        }
        jComboBox1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jComboBox1ActionPerformed(evt);
            }
```

```
});

jTextField1.setText("Input number");
jTextField1.addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusGained(java.awt.event.FocusEvent evt) {
        jTextField1FocusGained(evt);
    }
});
jTextField1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1ActionPerformed(evt);
    }
});
jTextField1.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyReleased(java.awt.event.KeyEvent evt) {
        jTextField1KeyReleased(evt);
    }
    public void keyTyped(java.awt.event.KeyEvent evt) {
        jTextField1KeyTyped(evt);
    }
});

jTextField2.setEditable(false);
jTextField2.setAutoscrolls(false);

jLabel4.setText("System");

jLabel5.setText("Quantity");

jLabel6.setText("Price");

button3.setLabel("Sell indicated number of systems");
button3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        button3ActionPerformed(evt);
    }
});
```

```
        button4.setLabel("Sell component");
        button4.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                button4ActionPerformed(evt);
            }
        });

        button5.setLabel("Restock all");
        button5.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                button5ActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 730, Short.MAX_VALUE)
                    .addComponent(jScrollPane2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addGroup(layout.createSequentialGroup()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addComponent(jLabel1)
                            .addGroup(layout.createSequentialGroup()
                                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                                    .addComponent(jComboBox1,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jLabel4))
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATE
D)
```

```
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                        .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel5))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                        .addComponent(jLabel6)
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELAT
ED)
                            .addComponent(button3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))))
                    .addComponent(jLabel2))
                .addGap(0, 0, Short.MAX_VALUE))
            .addGroup(layout.createSequentialGroup()
                .addComponent(button4, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(button5, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addContainerGap())
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jLabel1)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 160,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(button4, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                .addComponent(button5, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel2)
            .addGap(4, 4, 4)
            .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 174,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(18, 18, 18)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
              .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                  .addComponent(jLabel4)
                  .addComponent(jLabel5)
                  .addComponent(jLabel6))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                  .addComponent(jComboBox1, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.B
ASELINE)
                      .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                      .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGap(4, 4, 4))))
              .addGroup(layout.createSequentialGroup()
                .addComponent(button3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(4, 4, 4)))
            .addGap(34, 34, 34))
        );

        pack();
    }// </editor-fold>
```

```
    private void formWindowClosing(java.awt.event.WindowEvent evt) {
        // TODO add your handling code here:
        handler.closeConnection();
    }

    private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jTextField1FocusGained(java.awt.event.FocusEvent evt) {
        jTextField1.setText("");
    }

    private void jTextField1KeyReleased(java.awt.event.KeyEvent evt) {
        if(jTextField1.hasFocus())
        {
            if(StringUtils.isNumeric(jTextField1.getText()))
            {
                int qty = Integer.parseInt(jTextField1.getText());
                if(qty == 1)

jTextField2.setText(""+df.format(handler.getSystemSellPrice((String)jComboBox1.getSelectedItem
()())));
                else if((qty > 1) && (qty <= 11))

jTextField2.setText(""+df.format(handler.getSystemSellPrice((String)jComboBox1.getSelectedItem
())* qty * (1-(0.02*(qty-1)))));
                else

jTextField2.setText(""+df.format(handler.getSystemSellPrice((String)jComboBox1.getSelectedItem
())* qty * 0.8));
            }
            else
                jTextField2.setText("invalid number");
        }
    }

    private void jTextField1KeyTyped(java.awt.event.KeyEvent evt) {
```

```
    }

    private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void button4ActionPerformed(java.awt.event.ActionEvent evt) {
        if(jTable1.isColumnSelected(0))
        {
            try
            {
                if(handler.buyComponents(((String)jTable1.getValueAt(jTable1.getSelectedRow(),
jTable1.getSelectedColumn())), 1))
                    updateTables();
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
        else
        {
            new Error("Please select a component name in the table").setVisible(true);
        }
    }

    private void button3ActionPerformed(java.awt.event.ActionEvent evt) {
        try
        {
            if(handler.buySystems((String)jComboBox1.getSelectedItem(),
Integer.parseInt(jTextField1.getText())))
                updateTables();
            else
                new Error("No enough systems in stock").setVisible(true);
        }
        catch(NumberFormatException | SQLException e)
```

```
        {
            System.out.println(e);
        }
    }

    private void button5ActionPerformed(java.awt.event.ActionEvent evt) {
        if(handler.restockAll())
            updateTables();
    }

    // Variables declaration - do not modify
    private java.awt.Button button3;
    private java.awt.Button button4;
    private java.awt.Button button5;
    private javax.swing.JComboBox jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTable jTable1;
    private javax.swing.JTable jTable2;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    // End of variables declaration
    /**
     * Takes a JTable and a ResultSet as parameters and populates the JTable with the supplied
ResultSet
     * @param table
     * @param rs
     */
    private void fillTable(javax.swing.JTable table, java.sql.ResultSet rs)
    {
        try
        {
```

```
        //To remove previously added rows
        while(table.getRowCount() > 0)
        {
            ((javax.swing.table.DefaultTableModel) table.getModel()).removeRow(0);
        }
        int columns = rs.getMetaData().getColumnCount();
        Object[] ids = new Object[columns];
        while(rs.next())
        {
            Object[] row = new Object[columns];
            for (int i = 1; i <= columns; i++)
            {
                row[i - 1] = rs.getObject(i);
            }
            ((javax.swing.table.DefaultTableModel) table.getModel()).insertRow(rs.getRow()-1,row);
        }
        for (int i = 1; i <= columns; i++)
            {
                ids[i-1] = rs.getMetaData().getColumnName(i);
            }
        ((javax.swing.table.DefaultTableModel) table.getModel()).setColumnIdentifiers(ids);
        rs.close();
    }
    catch(Exception e)
    {
        System.out.print(e);
    }
}

private void fillCompTable(javax.swing.JTable table, java.sql.ResultSet rs)
{
    try
    {

        //To remove previously added rows
        while(table.getRowCount() > 0)
```

```
        {
           ((javax.swing.table.DefaultTableModel) table.getModel()).removeRow(0);
        }
        int columns = rs.getMetaData().getColumnCount();
        int rows = 0;
        java.util.ArrayList<String> helper = new java.util.ArrayList();
        //Adding column titles
        Object[] ids = new Object[columns];
        for (int i = 1; i <= columns; i++)
           {
              ids[i-1] = rs.getMetaData().getColumnName(i);
           }
        ((javax.swing.table.DefaultTableModel) table.getModel()).setColumnIdentifiers(ids);
        //Adding rows from ResultSet to table model.
        while(rs.next())
        {
           helper.add(rs.getString("name"));
           rows++;
           Object[] row = new Object[columns];
           for (int i = 1; i <= columns; i++)
           {
              row[i - 1] = rs.getObject(i);
           }
           ((javax.swing.table.DefaultTableModel) table.getModel()).insertRow(rs.getRow()-1,row);
        }

        //Adding new column with restock numbers
        Object[] restock = new Object[rows];
        for(int i = 0 ; i < restock.length ; i++)
        {
           restock[i] = handler.getCompRestock(helper.get(i));
        }
        ((javax.swing.table.DefaultTableModel) table.getModel()).addColumn("# to Restock",
restock);
        //Adding selling prices
        Object[] sellPrice = new Object[rows];
        for(int i = 0 ; i < sellPrice.length ; i++)
```

```
        {
            sellPrice[i] = ((int)jTable1.getValueAt(i, 1))*1.3;
        }
        ((javax.swing.table.DefaultTableModel) table.getModel()).addColumn("Selling price",
sellPrice);

        rs.close();
      }
      catch(Exception e)
      {
        System.out.print(e);
      }
   }

   private void fillSysTable(javax.swing.JTable table, java.sql.ResultSet rs)
   {
      try
      {

        //To remove previously added rows
        while(table.getRowCount() > 0)
        {
          ((javax.swing.table.DefaultTableModel) table.getModel()).removeRow(0);
        }
        int columns = rs.getMetaData().getColumnCount();
        int rows = 0;
        //Adding column headers
        Object[] ids = new Object[columns];
        for (int i = 1; i <= columns; i++)
          {
            ids[i-1] = rs.getMetaData().getColumnName(i);
          }
        ((javax.swing.table.DefaultTableModel) table.getModel()).setColumnIdentifiers(ids);
        java.util.ArrayList<String> systems = new java.util.ArrayList();
        //Adding rows from ResultSet
        while(rs.next())
        {
```

```
            systems.add(rs.getString(1));
            Object[] row = new Object[columns];
            for (int i = 1; i <= columns; i++)
            {
               row[i - 1] = rs.getObject(i);
            }
            ((javax.swing.table.DefaultTableModel) table.getModel()).insertRow(rs.getRow()-1,row);
            rows++;
         }
         //Adding new column with prices
         Object[] prices = new Object[rows];
         for(int i = 0 ; i < prices.length ; i++)
         {
            prices[i] = handler.getSystemPrice(systems.get(i));
         }
         ((javax.swing.table.DefaultTableModel) table.getModel()).addColumn("Price", prices);
         //Adding new column with selling prices
         Object[] sellPrices = new Object[rows];
         for(int i = 0 ; i < sellPrices.length ; i++)
         {
            sellPrices[i] = handler.getSystemSellPrice(systems.get(i));
         }
         ((javax.swing.table.DefaultTableModel) table.getModel()).addColumn("Selling price",
sellPrices);
         //Adding new column with current stock
         Object[] inStock = new Object[rows];
         for(int i = 0 ; i < prices.length ; i++)
         {
            inStock[i] = handler.systemsInStock(systems.get(i));
         }
         ((javax.swing.table.DefaultTableModel) table.getModel()).addColumn("In stock", inStock);

         rs.close();
      }
      catch(Exception e)
      {
         System.out.print(e);
```

```
        }
    }
}
```

SQLHandler.java

```
package computer_store;

import java.sql.*;
import java.util.ArrayList;
import java.util.HashMap;

/**
 *
 * @author ShameOnU
 */
public class SQLHandler
{
    Connection c = null;
    HashMap<String, Integer> prices = new HashMap();
    HashMap<String, Integer> sellPrices = new HashMap();

    public void init()
    {
        try
        {
            if(connect())
                System.out.println("Connected to database");
            else
                System.out.println("Not connected");
            buildPricesDict();
        }
        catch(Exception e)
        {
            e.printStackTrace();
            System.err.println(e.getClass().getName()+": "+e.getMessage());
        }
```

```
    }

    public Boolean connect()
    {
      try
      {
        Class.forName("org.postgresql.Driver");
        c = DriverManager.getConnection("jdbc:postgresql://localhost:5432/computer store",
"postgres", "123");
        return true;
      }
      catch (Exception e)
      {
        e.printStackTrace();
        System.err.println(e.getClass().getName()+": "+e.getMessage());
        return false;
      }

    }

    public ResultSet executeSQL(String statement)
    {
       try
       {
          return c.createStatement().executeQuery(statement);
       }
       catch(Exception e)
       {
          e.printStackTrace();
          System.err.println(e.getClass().getName()+": "+e.getMessage());
          return null;
       }
    }

    public Boolean updateSQL(String statement)
    {
       try
```

```
    {
      c.createStatement().executeUpdate(statement);
      return true;
    }
    catch(Exception e)
    {
       e.printStackTrace();
       System.err.println(e.getClass().getName()+": "+e.getMessage());
       return false;
    }
  }

  public void closeConnection()
  {
    try
    {
       if(c != null)
       c.close();
    }
    catch(Exception e)
    {

    }

  }

  /*
  Takes a result set containing only 1 column and returns a String array.
  */
  public String[] resultSetToArray(ResultSet rs) throws SQLException
  {
    ArrayList<String> helper = new ArrayList<>();
    ResultSetMetaData meta = rs.getMetaData();
    if(meta.getColumnCount() > 1)
    {
       System.out.print("Too many columns");
       return null;
```

```
    }
    else
    {
        int i;
        for(i = 0 ; rs.next() ; i++)
        {
            helper.add(rs.getString(1));
        }

        String[] result = new String[i];

        for(int ni = 0 ; !helper.isEmpty() ; ni++)
        {
            result[ni] = helper.get(0);
            helper.remove(0);
        }
        rs.close();
        return result;
    }

}

public int getComponentPrice(String component) throws SQLException
{
    String statement =
          "SELECT price "
         +"FROM component "
         +"WHERE name = '"+component+"';";
    //System.out.println(statement);
    ResultSet rs = executeSQL(statement);
    if(rs.next())
    {
        int result = rs.getInt(1);
        rs.close();
        return  result;
    }
    else
```

```
    {
        rs.close();
        return 0;
    }

}

private int getComponentQty(String component) throws SQLException
{
    String statement =
          "SELECT qty "
         +"FROM component "
         +"WHERE name = '"+component+"';";
    //System.out.println(statement);
    ResultSet rs = executeSQL(statement);
    if(rs.next())
    {
        int result = rs.getInt(1);
        rs.close();
        return  result;
    }
    else
    {
        rs.close();
        return 0;
    }

}

public ResultSet getAllComponents()
{
    String statement =
    "SELECT * "
  + "FROM component;";
    return executeSQL(statement);
}
```

```
public ResultSet getAllComponentNames()
{
    String statement =
    "SELECT name "
  + "FROM component;";
    return executeSQL(statement);
}

public ResultSet getAllSystemNames()
{
    String statement =
    "SELECT name "
  + "FROM system;";
    return executeSQL(statement);
}

public ResultSet getAllSystems()
{
    String statement =
    "SELECT * "
  + "FROM system;";
    return executeSQL(statement);
}

private ResultSet getSystem(String system)
{
    String statement =
    "SELECT * "
  + "FROM system "
  + "WHERE name = '"+system+"';";
    return executeSQL(statement);
}

public int getSystemPrice(String system)
{
    return prices.get(system);
}
```

```
public int getSystemSellPrice(String system)
{
   return sellPrices.get(system);
}

private void buildPricesDict() throws SQLException
{
   ResultSet systems = getAllSystems();
   ResultSetMetaData meta = systems.getMetaData();
   while(systems.next())
   {
      String name = null;
      int price = 0;
      for(int i = 1 ; i <= meta.getColumnCount() ; i++)
      {
         if(i == 1)
         {
            name = systems.getString(i);
         }
         else
         {
            String comp = systems.getString(i);
            System.out.println(comp);
            if(comp != null)
            price = price + getComponentPrice(comp);
         }
      }
      prices.put(name, price);
      price = (int) (price * 1.3);
      if(price < 100)
            sellPrices.put(name, 99);
         else
         {
            price = price / 100;
            price = price + 1;
            price = price * 100;
```

```
                price = price - 1;
                sellPrices.put(name, price);
            }
        }
        systems.close();
    }

    public int systemsInStock(String system) throws SQLException
    {
        ResultSet rs = getSystem(system);
        if(rs.next())
        {
            ArrayList<Integer> list = new ArrayList();
            int columns = rs.getMetaData().getColumnCount();
            for(int i = 2 ; i < columns ; i++)
            {
                String comp = rs.getString(i);
                if(!comp.equals(""))
                {
                    list.add(getComponentQty(comp));
                }
            }
            return java.util.Collections.min(list);
        }
        else
            return 0;
    }

    public Boolean buyComponents(String component, int qty) throws SQLException
    {
        int currentQty = getComponentQty(component);
        if( currentQty >= qty)
        {
            String statement =
                    "UPDATE component "
                   +"SET qty = "+(currentQty - qty)+" "
                   +"WHERE name ='"+component+"';";
```

```
        try
        {
            return updateSQL(statement);
        }
        catch(Exception e)
        {
            System.out.println(e);
            return false;
        }
    }
    else
        return false;
}

public Boolean buySystems(String system, int qty) throws SQLException
{
    if(systemsInStock(system) >= qty)
    {
        ResultSet rs = getSystem(system);
        if(rs.next())
        {
            ArrayList<String> list = new ArrayList();
            String comp;
            for(int i = 2 ; i < rs.getMetaData().getColumnCount() ; i++)
            {
                if(!((comp = rs.getString(i)).equals("")))
                list.add(comp);
            }
            while(!list.isEmpty())
            {
             if(buyComponents(list.get(0),qty))
                list.remove(0);
                else
                 return false;
            }
            return true;
        }
```

```
        else
            return false;
    }
    else
        return false;
}

public int getCompRestock(String component) throws SQLException
{
    String statement =
    "SELECT component.name, component.qty, stock_rules.pref_qty "
  + "FROM component, stock_rules "
  + "WHERE component.name = stock_rules.name AND component.name ='"+component+"';";
    ResultSet rs = executeSQL(statement);
    if(rs.next())
    {
        int restock = rs.getInt("pref_qty")-rs.getInt("qty");
        if(restock > 0)
            return restock;
        else
            return 0;
    }
    else
        return 0;

}

public Boolean restockAll()
{
    ResultSet rs = getAllComponentNames();
    try
    {
        String[] comps = resultSetToArray(rs);
        for(String comp : comps)
        {
            int restock = getCompRestock(comp);
            if(restock>0)
```

```
            {
                String statement=
                "UPDATE component "
               +"SET qty = qty + "+restock+" "
               +"WHERE name = '"+comp+"';";
                updateSQL(statement);
            }
        }
        return true;
    }
    catch (SQLException ex)
    {
        System.out.println(ex);
        return false;
    }
  }
}
```