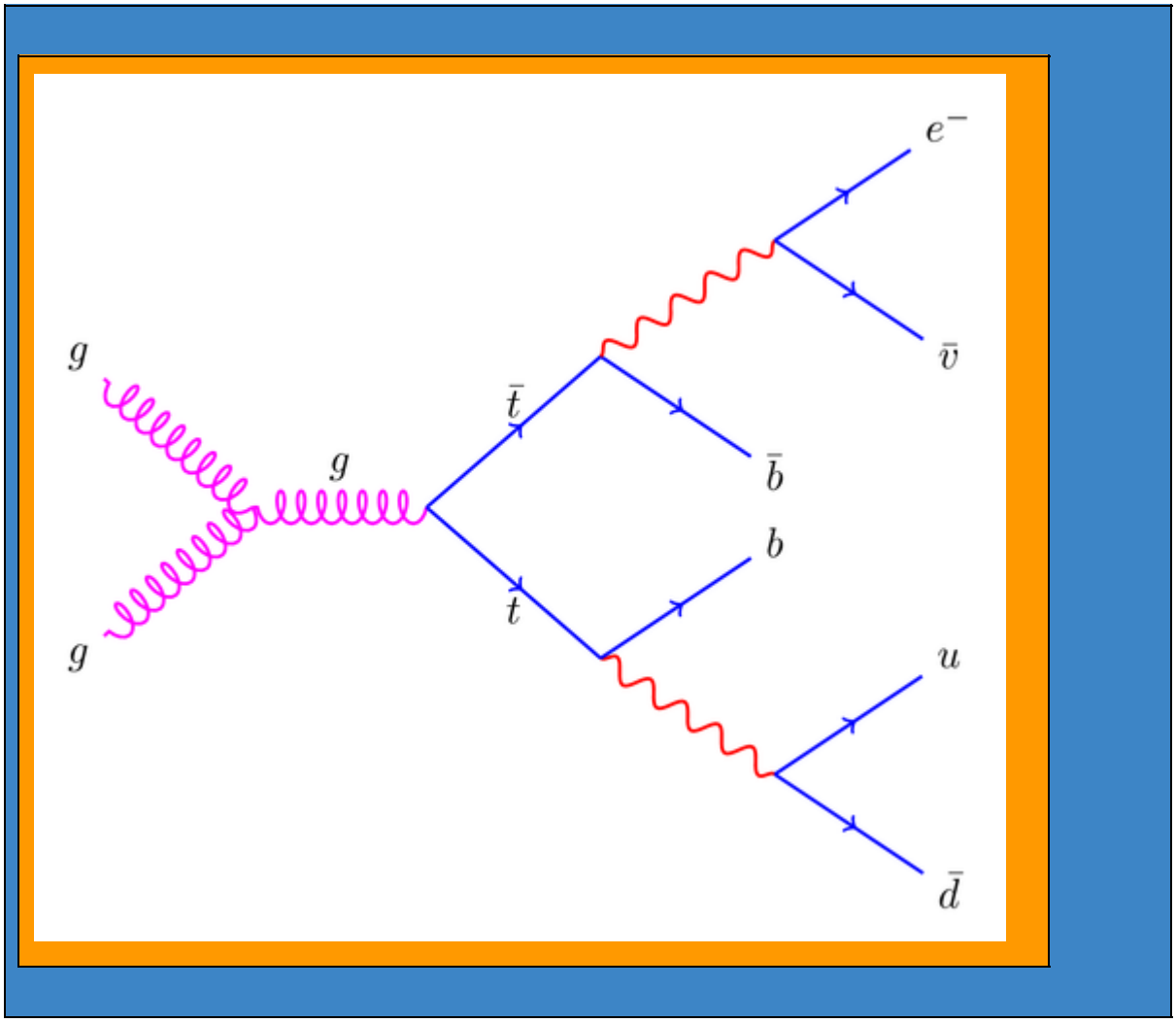
Ecommerce Database Project

CPS 510 - Group 02
Section 041
Fall 2019



Taimoor Ahmad 500770125
Brandon Haripersad 500769285
Jeffery Keith 500804619

Table of Contents

1. Design

1.1 Description

We are designing an ecommerce website. The ecommerce website has 3 models: the business owner (Admin), Buyer (Buyer), Seller (Seller).

- The business owner can control anything within the application.
- Buyer that creates Reviews (one to many), Checkouts (one to one).
- Seller that creates a Storefront (one to one) , Discount Codes (one to many), Products (one to many)

We also represent Products, Discount Codes, Reviews in our application.

- A Product belongs to a Seller and is found in a Checkout.
- A Discount Code belongs to a Seller and can be found in a Checkout.
- Reviews which are created by a Buyer and are linked to a certain product.

Every model within our database will have a unique ID for its primary key.
In total there are 8 entities and 9+ relationships.

1.2 ER Model

The initial ER model (Fig 1.) has been refined over the course of the semester and can be more clearly understood by the model created by SQL Developer in Fig 2.

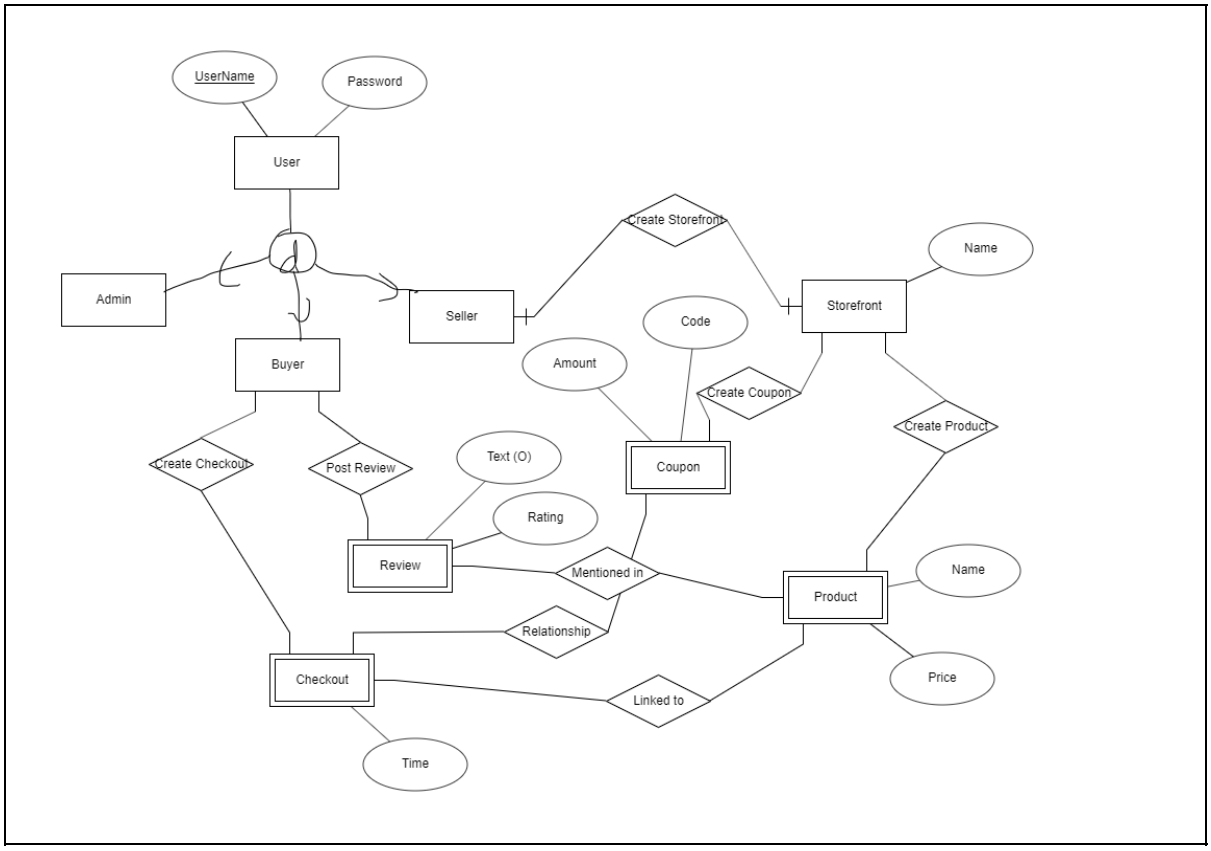


Fig 1. Initial ER Model

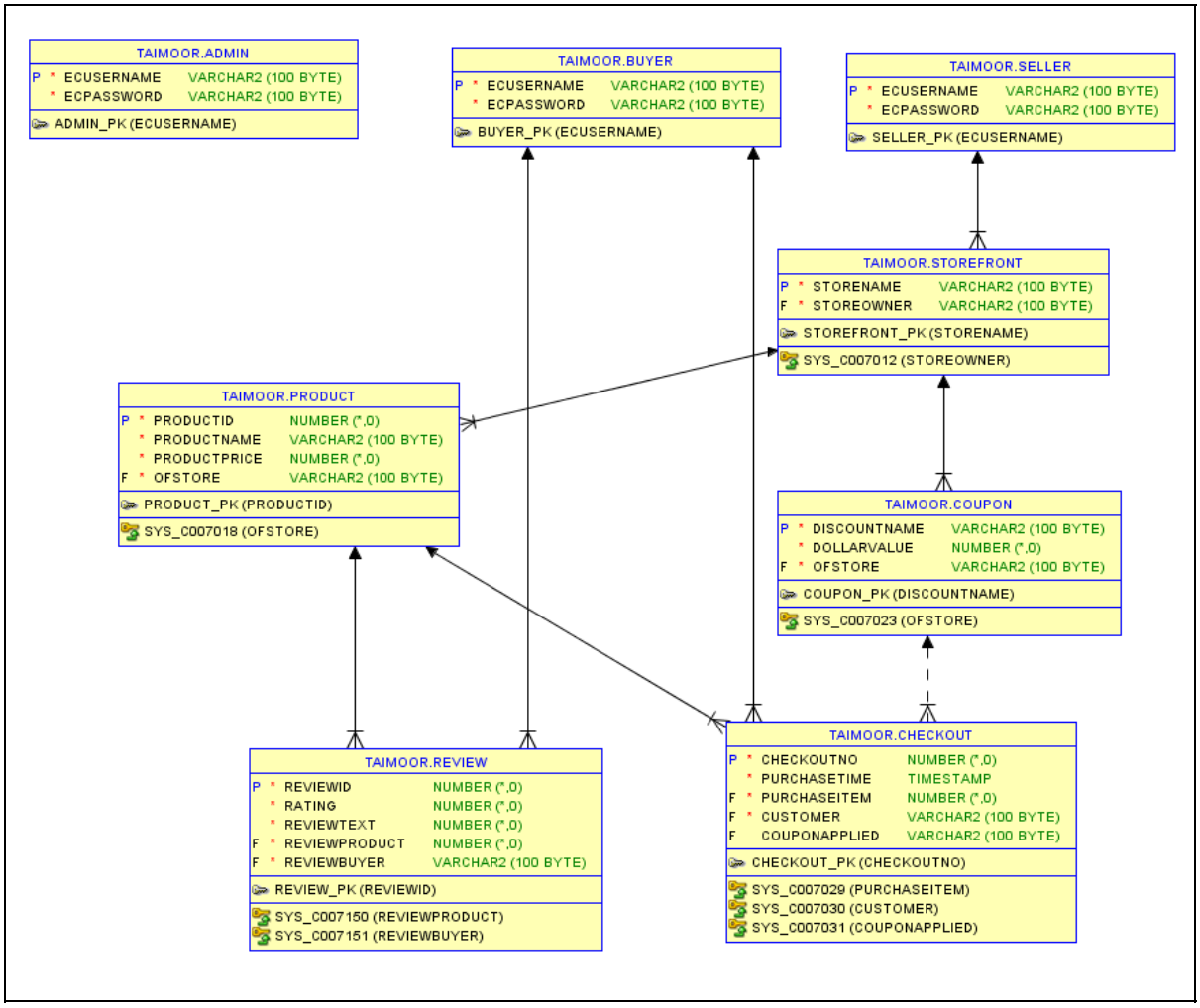


Fig 2. Refined ER Model

1.3 Schema Design

After creating the ER model for the database we constructed the schema by using DDL tools. Construction of the tables which reflect the relationships in the ER model were done using SQL.

The following are the SQL instructions for creating the tables for the relationships in the ER Model:

```
CREATE TABLE Admin(
    ECUsername varchar2(100) NOT NULL,
    ECPassword varchar2(100) NOT NULL,
    PRIMARY KEY (ECUsername)
);
```

```
CREATE TABLE Buyer(
    ECUsername varchar2(100) NOT NULL,
    ECPassword varchar2(100) NOT NULL,
    PRIMARY KEY (ECUsername)
);
```

```
CREATE TABLE Seller(
    ECUsername varchar2(100) NOT NULL,
    ECPassword varchar2(100) NOT NULL,
    PRIMARY KEY (ECUsername)
);
```

```
CREATE TABLE Storefront(
    StoreName varchar2(100) NOT NULL,

    /* Has a reference to the owner of the store */
    StoreOwner varchar2(100) not null,
    FOREIGN KEY (StoreOwner) REFERENCES Seller(ECUserName),
    PRIMARY KEY(StoreName)
);
```

```
CREATE TABLE Product(
    ProductID int NOT NULL,
    ProductName varchar2(100) NOT NULL,
    ProductPrice int NOT NULL,

    /* Has a reference to the store the product belongs to */
    OfStore varchar2(100) not null,
    FOREIGN KEY (OfStore) references Storefront(StoreName),

    primary key (ProductID)
);
```

```
CREATE TABLE Coupon(
    DiscountName varchar2(100) not null,
    DollarValue int not null,

    /* Has a reference to the store the coupon belongs to */
    OfStore varchar2(100) not null,
    FOREIGN KEY (OfStore) references Storefront(StoreName),

    primary key (DiscountName)
);
```

```
CREATE TABLE Checkout(
    CheckoutNo int NOT NULL,
    PurchaseTime timestamp NOT NULL,

    /* Has a reference to the customer who bought, and a coupon code if one was used */
```

```
    PurchaseItem int not null,
    FOREIGN KEY (PurchaseItem) references Product(ProductID),

    Customer varchar2(100) not null,
    FOREIGN KEY (Customer) references Buyer(ECUserName),

    CouponApplied varchar2(100),
    FOREIGN KEY (CouponApplied) references Coupon(DiscountName),
    PRIMARY KEY (CheckoutNo)
);
```

```
CREATE TABLE review(
   ReviewID int NOT NULL,
   Rating int NOT NULL,
   ReviewText varchar2(1000),
   ReviewProduct int not null,
   ReviewBuyer varchar2(100),
   FOREIGN KEY (ReviewProduct) REFERENCES Product(ProductID),
   FOREIGN KEY (ReviewBuyer) REFERENCES Buyer(ECUserName),
   PRIMARY KEY (ReviewNo)
);
```

Fig 3. SQL Schema Design

2. Implementation

2.1 Views and Queries Demo

For assignment 4, we demonstrated the execution of simple quarries on our database system. During the week of Sept 31, we designed 9 basic queries to show that our database was working and displaying the correct information.

The simple queries can be seen below:

```
1. Select * from ecommerce_user;
2. select * from ecommerce_user where username = 'Jeff';
3. select * from BUYER where password = 'mypass';
4. select * from storefront where storeno = 123;
5. select * from coupon where storeno = 123;
6. select * from product where productprice < 200;
```

```
7. select * from coupon where couponno = 432;
8. select * from coupon where couponname = '5% off';
9. select * from product;
```

Fig 4. Simple SQL Queries

Then during the week of Oct 7, we implemented 2 VIEWS as well as making our queries more complex by creating JOIN statements.

These queries can be seen below along with a screenshot of them running correctly:

```
/* Queries relating to user management */
1. SELECT * FROM ecommerce_user;
2. SELECT * FROM ecommerce_admin WHERE UserName = "TaimoorAhmad";
3. SELECT * FROM seller WHERE Password = "JeffRocks123";
4. SELECT * FROM buyer WHERE UserName = "Brandon" AND Password =
   "TestPassword123";

/* Queries relating to checkouts and reviews */
5. SELECT * FROM checkout WHERE trans_no = 100;
6. SELECT * FROM review WHERE buyer.UserName = "TaimoorAhmad";

/* Queries relating to products and storefronts and coupons */
7. SELECT * FROM coupon WHERE CouponDiscount >= 4.99;
8. SELECT * FROM product WHERE ProductPrice >= 99.99 AND
   ProductPrice <= 149.99;
9. SELECT * FROM storefront WHERE seller.UserName = "SirJeff";

10.  CREATE VIEW coups AS ( SELECT * from storefront where storeno
   = 123);
11.  CREATE VIEW prods as (select * from product where
   productprice < 1000);
```

Fig 5. Advanced SQL Queries

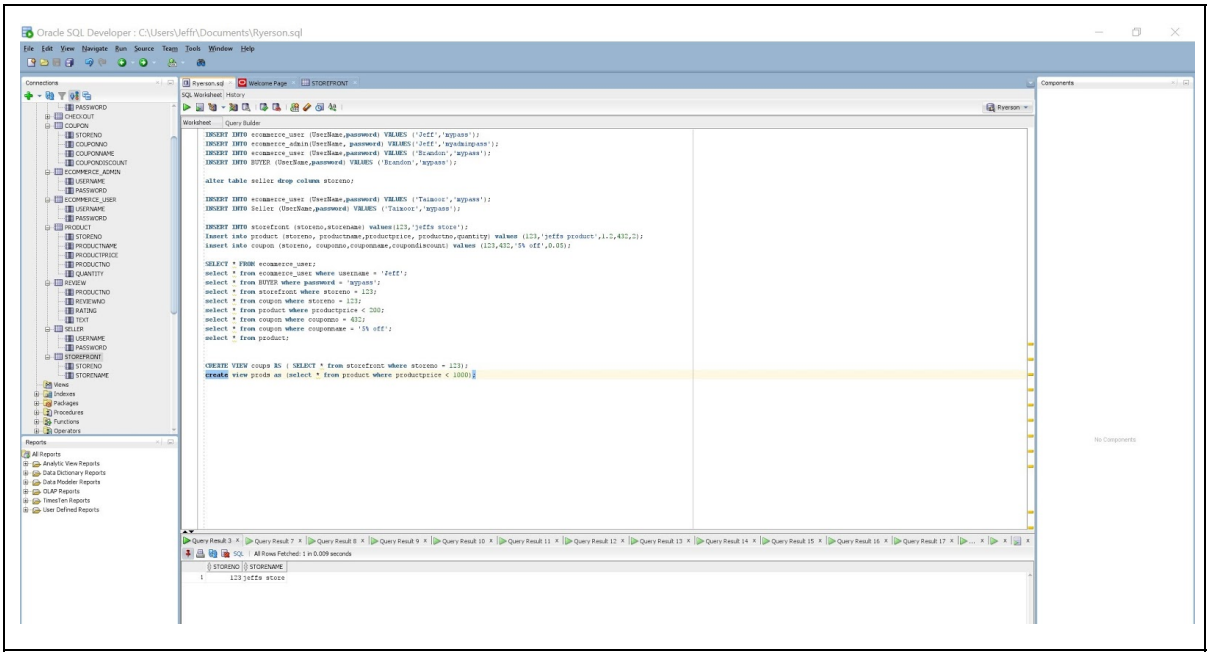


Fig 6. SQL Queries in SQL Developer

2.2 Relational Algebra

We have also created relational algebra statements for each of our simple SQL queries from Fig 4. The relational algebra statements that correspond to the SQL statements are listed in Fig 6.

1. RA = **σ**(ecommerce_user)
2. RA = **σ**UserName=’Jeff’ (ecommerce_user)
3. RA = **σ**password=’mypass’ (BUYER)
4. RA = **σ**storeno=123 (storefront)
5. RA = **σ**storeno=123 (coupon)
6. RA = **σ**productprice < 200 (product)
7. RA = **σ**couponno=432 (coupon)
8. RA = **σ**couponname=’5% off’ (coupon)
9. RA = **σ**(product)

Fig 7. Relational Algebra for Simple Queries

Likewise, we created relational algebra statements for the advanced queries listed in Fig 5. These are listed below in Fig 8.

1. RA = **σ**(ecommerce_user)
2. RA = **σ**UserName=’TaimoorAhmed’ (ecommerce_user)
3. RA = **σ**password="JeffRocks123" (seller)
4. RA = **σ**UserName=’Brandon’(**σ**password=”TestPassword123” (buyer))
5. RA = **σ**trans_no=100 (checkout)
6. RA = **σ**Username = ’TaimoorAhmed’(review X buyer)

7. RA = **σ**CouponDiscount >= 4.99 (coupon)
8. RA = **σ**ProductPrice >= 99.99(**σ**ProductPrice <= 149.99 (product))
9. RA = **σ**Username = 'SirJeff'(storefront x seller)

Fig 8. Relational Algebra for Advanced Queries

2.3 Unix Shell

For assignment 5, we selected 5 interesting queries and implemented them into a UNIX shell program that allowed us to run our selected queries by simply typing in a number. We then demoed the working program to the TA.

Below is the copy of the code implemented:

```
#!/bin/sh
function Pause() {
    read -p "$*"
} # ...


MainMenu() {

    while [ "$CHOICE" != "START" ]; do
        clear
        echo
"================================================================="
        echo "| Oracle All Inclusive Tool |"
        echo "| Main Menu - Select Desired Operation(s): |"
        echo "| <CTRL-Z Anytime to Enter Interactive CMD Prompt> |"
        echo
"-----------------------------------------------------------------"
        echo " $IS_SELECTEDM M) View Manual"
        echo " "
        echo " $IS_SELECTED1 1) Select all products that are
Headphones and order them by LIKE"
        echo " $IS_SELECTED2 2) Select all users who have placed an
order by EXISTS"
        echo " $IS_SELECTED3 3) Get avg price of products from
BrandonMart by AGGREGATE"
        echo " $IS_SELECTED4 4) Get all purchases of a user by JOIN"
        echo " $IS_SELECTED4 5) Get all products from a store
excluding Headphones by MINUS"
        echo " "
        echo " $IS_SELECTEDX X) Force/Stop/Kill Oracle DB"
        echo " "
        echo " $IS_SELECTEDE E) End/Exit"
        echo "Choose: "
```

```
        read CHOICE
        if [ "$CHOICE" == "0" ]; then
            echo "Nothing Here"
        elif [ "$CHOICE" == "1" ]; then
            echo "select * from product where productname like
'Wireless%' order by productprice asc;" | sqlplus.exe taimoor/taimoor
            Pause
        elif [ "$CHOICE" == "2" ]; then
            echo "select ecusername from buyer where exists (select *
from checkout where buyer.ecusername = checkout.customer);" |
sqlplus.exe taimoor/taimoor
            Pause
        elif [ "$CHOICE" == "3" ]; then
            echo "select avg(productprice) from product where ofstore
= 'BrandonMart';" | sqlplus.exe taimoor/taimoor
            Pause
        elif [ "$CHOICE" == "4" ]; then
            echo "select ecusername, purchaseitem from buyer,
checkout where buyer.ecusername = checkout.customer and purchaseitem
= checkout.purchaseitem;" | sqlplus.exe taimoor/taimoor
            Pause
        elif [ "$CHOICE" == "5" ]; then
            echo "select * from product where ofstore = 'BrandonMart'
minus (select * from product where productname = 'Headphones');" |
sqlplus.exe taimoor/taimoor
            Pause

        elif [ "$CHOICE" == "E" ]; then
            exit
        fi
    done
}

#--COMMENTS BLOCK--
# Main Program
#--COMMENTS BLOCK--
ProgramStart() {
    StartMessage
    while [ 1 ]; do
        MainMenu

    done
}

ProgramStart
```

Fig 9. Unix Shell Script

2.3 Java GUI

For assignments 9 and 10 our table was put into BCNF/3NF form ahead of time. After formatting our table properly in Oracle SQL, we created a Java program

The Java program performs the same queries by running sqlplus64.exe and querying the database from Oracle SQL Developer.

The Java library used for graphics is “Swing”. It is a wrapper around the command line calls to sqlplus64.exe.

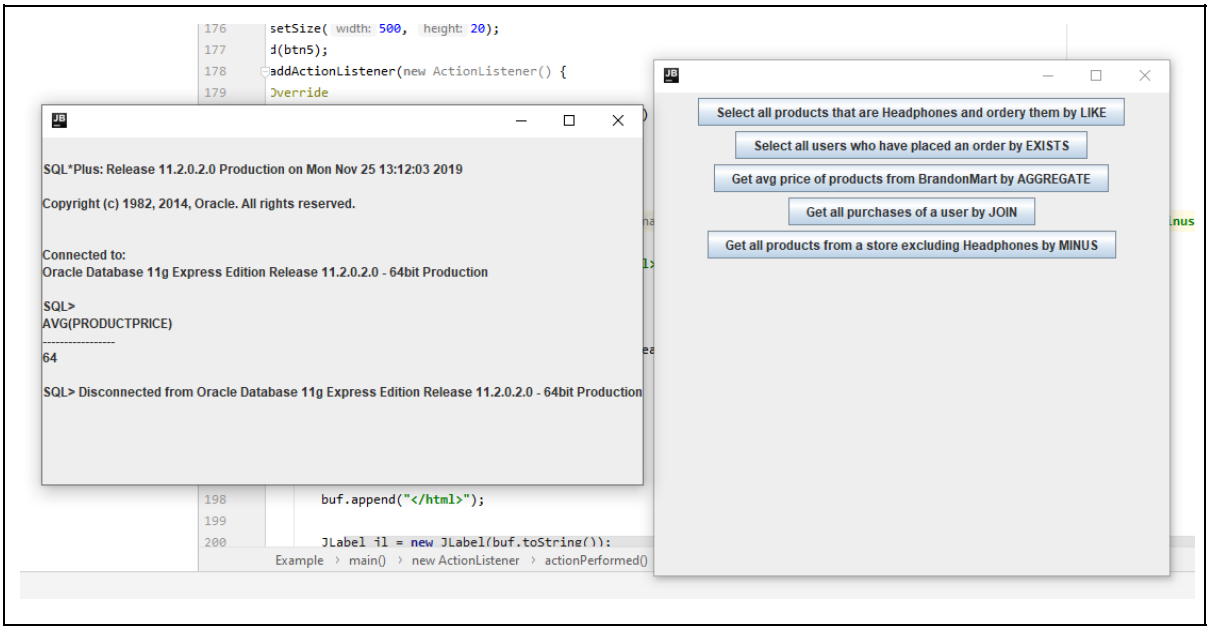


Fig 10. Java GUI

```
// Snippet of code from our application (reading the query and running it)
btn1.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent actionEvent) {
       JFrame inner = new JFrame();
       inner.setSize(500, 500);

       try {
           Process p = Runtime.getRuntime().exec("wsl echo \"select * from
product where productname like 'Wireless%' order by productprice asc;\" |
sqlplus.exe taimoor/taimoor");

           StringBuilder buf = new StringBuilder("<html>");
           String line;

           BufferedReader in = new BufferedReader(
                   new InputStreamReader(p.getInputStream()) );
           while ((line = in.readLine()) != null) {
```

```
                //System.out.println(line);
                buf.append(line).append("<br>");
            }
            in.close();

            buf.append("</html>");

            System.out.println(buf.toString());

            JLabel jl = new JLabel(buf.toString());
            inner.add(jl);
            inner.pack();
            inner.validate();
        } catch (IOException e) {
            e.printStackTrace();
        }

        inner.setLayout(new FlowLayout());
        inner.setVisible(true);
    }
});
```

Fig 11. Java Code Snippet

3. Normalization

3.1 Normalization and Functional Dependencies

For assignment 6, we begin the process of normalizing all our tables to be in 3NF and BCNF. To begin this process, we first had to list all the Functional Dependencies for each table and give a brief explanation as to why we believe this is true.

Below is a copy of the report:

Admin
- ECUsername -> ECPassword
    - ECUsername is the Primary Key in the Admin table and is a unique identifier, ECUsername is needed to confirm password.

Buyer
- ECUsername -> ECPassword
    - ECUsername is the Primary Key in the Buyer table and is a unique identifier, ECUsername is needed to confirm password.

Seller
- ECUsername -> ECPassword
    - ECUsername is the Primary Key in the Seller table and is a unique identifier, ECUsername is needed to confirm password.

StoreFront
- ECUsername -> StoreOwner
  - ECUsername is referenced from Seller table and used to create a Storefront, ECUsername is required to create any kind of Storefront.
- StoreOwner -> StoreName
  - Every StoreOwner is linked to their own unique StoreName.
- StoreName -> OfStore
  - OfStore requires the creation of a StoreFront and thus can be identified through the StoreName.

Product
- ProductID -> ProductName, ProductPrice, OfStore
  - ProductID defines the ProductName, ProductPrice and is unique to every StoreFront belongs to, thus ProductID defines the StoreName it belongs to as well.

Coupon
- DiscountName -> DollarValue, OfStore
  - DiscountName is defines the dollar amount that is applied in the checkout, thus DollarValue is dependant on the DiscountName. Also every DiscountName is unique to every StoreFront, thus DiscountName can uniquely identify OfStore.

Checkout
- ECUsername -> CheckoutNo
  - Customer is dependant on an ECUsername being defined and ECUsername is can define the name of the Customer in CheckOut. Also Customer can identify the the CheckoutNo, meaning that transitively ECUsername is a functional requirement for CheckoutNo.
- CheckoutNo -> PurchaseTime, PurchaseItem, CouponApplied
  - PurhaseTime, PurhaseItem and the CouponApllied can all be identified using the CheckoutNo.
- PurchaseItem -> ProductID; ProductID -> PurchaseItem
  - ProductID can be identified from the PurchaseItem and also ProductItem can also be identified from PurchaseItem. Therefore, both values are functionally dependant on each other.
- CouponApplied -> DiscountName; DiscountName -> CouponApllied
  - DiscountName can be identified from the CouponApplied and also the other way. CouponApplied can be uniquely identified through the DiscountName therefore making them both functionally dependent of each other.

Fig 12. Functional Dependencies

3.2 Normalization into 3NF

After examining our SQL tables and creating an ER diagram using SQL Developer, we checked the relations within our database.
We found that our database was designed from the ground up without any redundancy, so no changes had to be made. Every reference to another table was a way to grab the data from that table, rather than storing redundant data.

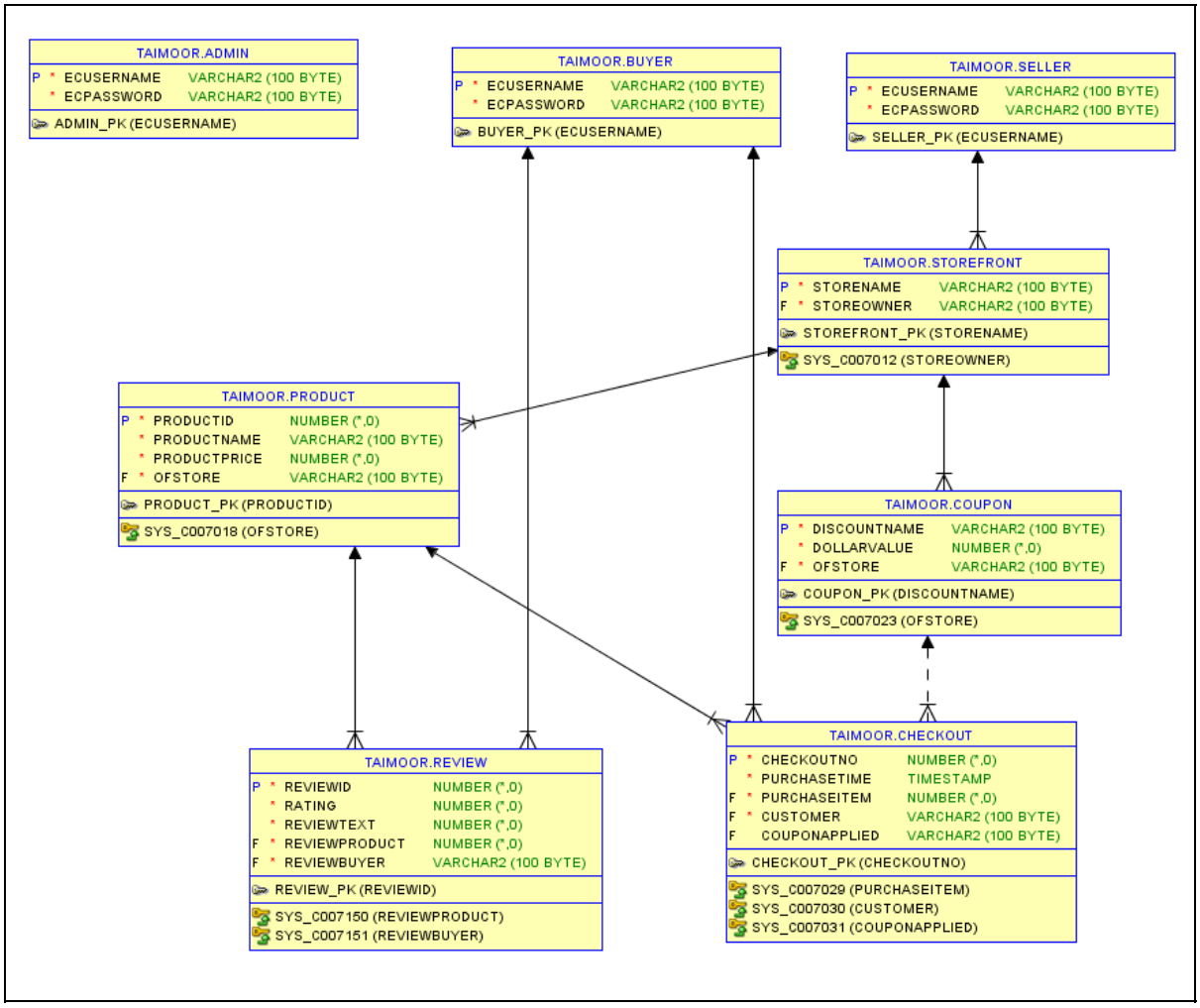


Fig 13. UML Diagram of Database

3.3 BCNF/3NF Algorithm

Same as in the previous assignment, our table was already in BCNF and 3NF form since it contains no redundant data: the references from data in other tables is done via the Primary Key.

See document on next page for further explanation.

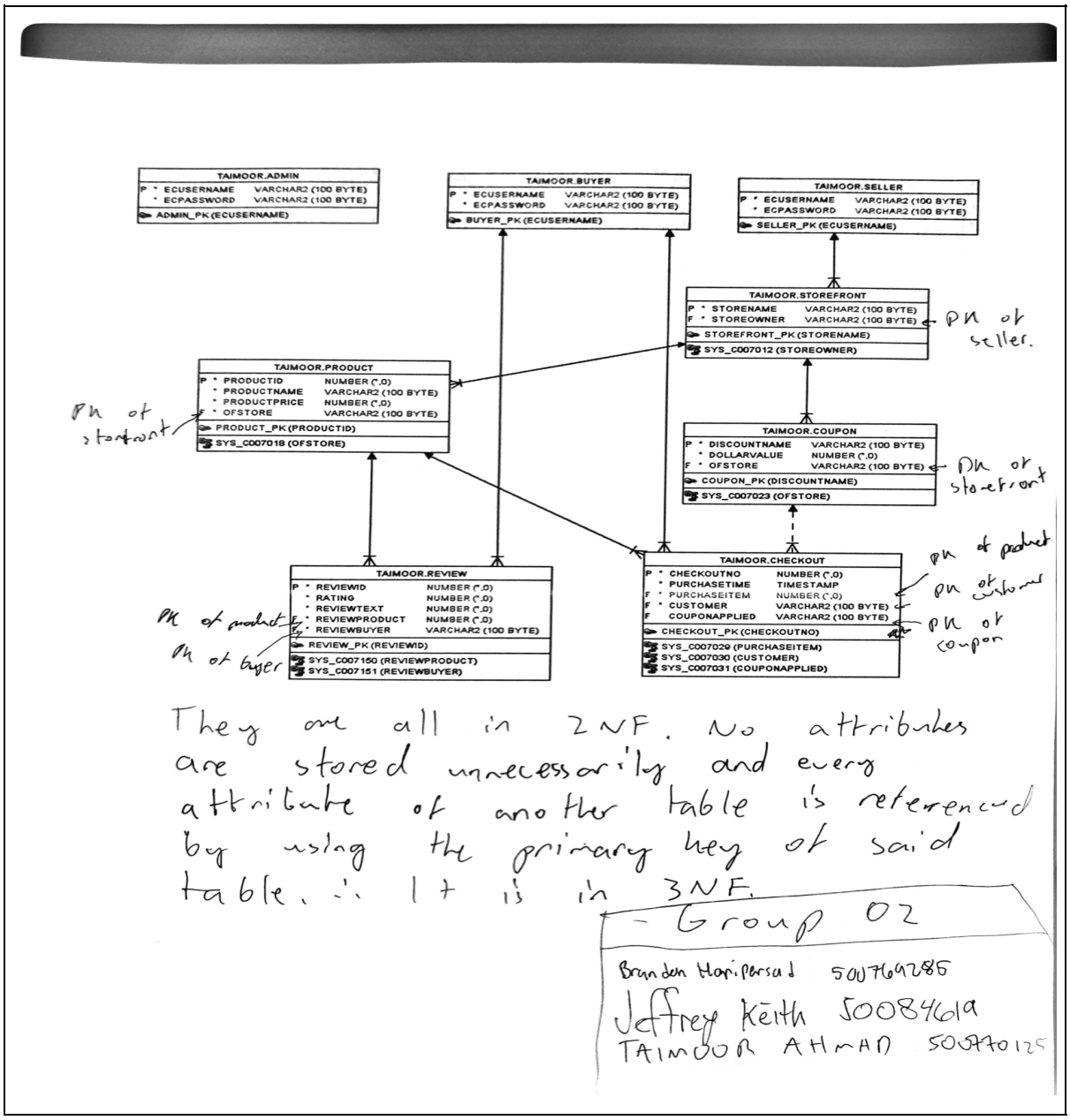


Fig 14. Document showing BCNF/3NF