2017-4-27

# Database Final Project

## Online Shopping System

Fan Yang: yang.fan7@husky.neu.edu
Yan Li: li.yan9@husky.neu.edu
Dawei Zhang: zhang.daw@husky.neu.edu
Yangfu Zeng: zeng.yang@husky.neu.edu

# 1.Problem Description

### 1.1 Introduction

Nowadays online shopping is widely used in our daily life, which provides us a lot of convenience. Online shopping is a form of electronic commerce which allows consumers to directly buy goods or services from a seller over the Internet using a web browser. Consumers find a product of interest by visiting the website of the retailer directly or by searching among alternative vendors using a shopping search engine, which displays the same product's availability and pricing at different e-retailers.

### 1.2 Objective

In our project, we will design an online shopping platform. There are two main actors: Seller and Buyer. There are some critical functions.

For Buyer:

1. Create personal profile, address and payment methods;
2. Browse the sellers' listings, select their products then add them into shopping cart;
3. Modify and remove items in the shopping cart
4. Create new orders and check out.
5. Create new address and delete existing addresses;
6. Create new payment method and delete existing payment methods.
7. List buying history.

For Seller:

1. Post their products and list them;
2. Modify the information of items and delete existing items;
3. Search sales history.

In general, we would like to build up an "online shopping" database management system to imitate the real online shopping platform. All the functions will be accomplished on the website.

Methods

This website is built with AngularJS/Bootstrap as front-end, NodeJS as back-end. Of course, the database is MySQL.

All database queries locate in serverside services:

/server/services

buyer.service.server.js

seller.service.server.js

user.service.server.js

Each database query is implemented as a transaction to guarantee integrity.

An overlook of all serverside APIs is at:

/server/api.server.js

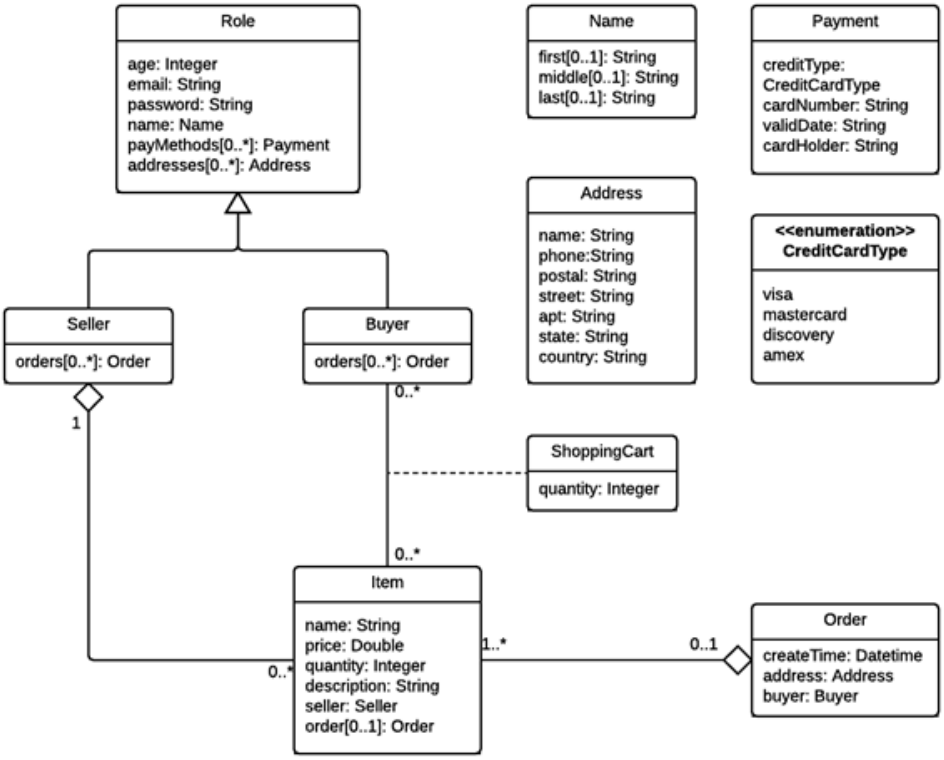
Each API entry maps to a handler defined in the three services.

Database initializing script is at:

/server/db_init.sql

## 2. Requirement & Design

We list all of the use cases below, describe their relationship with the UML above and how to fulfill them by listing related classes and queries. We use queries like select, insert, update, delete to fulfill different functions in the system




## System

**Online Shopping Platform [OSP]**

| |
|---|
| Name: Online Shopping Platform |
| Description: The OSP is a website platform for Roles as seller and buyer to process commercial items such post item, buy an item, pay for the item and so forth. All these actions fulfill the normal function for e-commerce shopping processes. |

It is fulfilled by tables in the database system. They are Role, Seller, Buyer, Item, Order, Shopping Cart, Address, Name, payment and enumeration tables CreditCardType.

## System Actors:

**Seller**

**Description:** Seller is one of the roles in the system. Its main functions are about items, such as creating items, viewing created items, changing created items, deleting creating. Also, functional include creating, updating or deleting its personal information. Besides, the seller can view all his/her orders history.

**Buyer**

**Description:** Seller is another role in the system. Its main functions are about items already created by the seller, such as searching created items, adding created items to shopping cart, modifying the quantities of items in shopping cart, checking out and paying for the shopping cart. Also, functions include creating, updating or deleting its personal information like a seller. Besides, the seller can view all his/her buying orders history.

**Role**

**Description:** It is a super class. It includes seller and buyer. Both of them have attributes about personal information like age, email, password, name, payment methods and address.

## Use Cases:

### Seller

### Post a new item [Seller]

Description:
A seller wants to post a new item to the system for sale.

Step-by-step Description:
1. [Seller] Seller fills in the item information, including name, quantity, price, description, etc.
2. [System] The system saves the new item in the database, returns a confirmation message.

Related Classes: Item, Seller

Query:

- Command: insert
- Contend: name, price, quantity, description, seller, id

These contend then are changed in the item table. The relationship between item and seller are composition.

### List posted items [Seller]

Description:
A seller wants to check all items posted on the system.

| |
|---|
| Step-by-step Description:<br>1. [Seller] Seller requests to list item information posted on the system including name, quantity, price, description, etc.<br>2. [System] The system confirms the request and shows all posted items information. |

Related Classes: Role, Item, Seller

Query:

- Command: Select
- Contend: name, price, quantity, description, seller, id

This item information in item table are listed.

## Change information of item [Seller]

| |
|---|
| Description:<br>A seller wants to change or updates an item information. |
| Step-by-step Description:<br>1. [Seller] Seller updates the item information, including name, quantity, price, description, etc.<br>2. [System] The system saves the updated item information in the database, returns a confirmation message. |

Related Classes: Role, Item, Seller

Query:

- Command: update
- Contend: name, price, quantity, description, seller, id

This item information in item table are changed.

**Delete an item [Seller]**

| |
|---|
| Description:<br>A seller wants to delete an item to the system. |
| Step-by-step Description:<br>1. [Seller] Seller chooses items to delete.<br>2. [System] System delete the information of this item in the database, returns a confirmation message. |

Related Classes: Role, Item, Seller

Query:

- Command: update

- Contend: name, price, quantity, description, seller, id

This item information in item table are deleted.

**Search sales history [Seller]**

| |
|---|
| Description:<br>Seller searches all sales history. |
| Step-by-step Description:<br>1. [Seller] The seller chooses sales history function in the system.<br>2. [System] The system searches all sales history, returns information including sale data, production, quantities, total price, addresses of the buyer, etc. |

Related Classes: Order, Item, Seller

Query:

- Command: Select
- Contend: CreateTime, address, buyer, name, price, quantity, description, seller, id, order.

This information are select to show. The relationship between order and item are one to many compositions.

## Buyer：

**Create personal Information [Buyer]**

| |
|---|
| Description:<br>A buyer can create his personal information and save. |
| Step-by-step Description:<br>1. [Buyer] The Buyer create his own email address, age, first name, middle name and last name. Then click save button.<br>2. [System] The system saves the new item in the database. |

Related Classes: Buyer, address, role, name

Query:

- Command: insert
- Contend: all attributes in role, buyer and address table

Insert new information in these tables to create new buyer information. Add foreign key to tables to created new subclass and multivalued attributes.

**Search for target item [Buyer]**

| |
|---|
| Description:<br>A buyer searches his target item with keywords and sees results. |

| |
|---|
| Step-by-step Description:<br>1. [Seller] Buyer inserts keyword of the target item in searching bar.<br>2. [System] Search items related with keywords in a database system, make a list of items including the photos, price, names, etc., then return results to Buyer. |

Related Classes: Buyer, Item

Query:

- Command: select
- Contend: name, price, quantity, description, seller, id

Item table information specified by id are listed here to Buyer.

Add items to the Shopping Cart [Buyer]

| |
|---|
| Description:<br>A buyer can select the item he wants to buy and put into the shopping cart. |
| Step-by-step Description:<br>1. [Buyer] The Buyer select target items, choose the quantity and click 'add to cart' button.<br>2. [System] The system save this information in the shopping cart. |

Related Classes: Buyer, Item, Order, ShoppingCart

Query:

- Command: Select, insert
- Contend: CreateTime, address, buyer, name, price, quantity, description, seller, quantity, id

This table selection information from item table then inserts new information into Order and ShoppingCart table. The association between Buyer and Item are a shopping cart.

Modify items quantity in the Shopping Cart [Buyer]

| |
|---|
| Description:<br>A buyer can modify the quantities of items he wants to buy in the shopping cart. |
| Step-by-step Description:<br>1. [Buyer] The Buyer modify the quantity in the shopping cart.<br>2. [System] The system calculates the new total. |

Related Classes: Buyer, ShoppingCart, item

Query:

- Command: update, select
- Contend: CreateTime, address, buyer, name, price, quantity, description, seller, quantity, id

Buyer select information from item table and update the quantity of shopping cart table. Then the total price changes.

**Remove items in the Shopping Cart [Buyer]**

| |
|---|
| Description:<br>A buyer can remove the items he used to select in the shopping cart. |
| Step-by-step Description:<br>1. [Buyer] The Buyer removes items in the shopping cart.<br>2. [System] The system calculates the new total. |

Related Classes: Buyer, ShoppingCart, item

Query:

- Command: delete
- Contend: CreateTime, address, buyer, name, price, quantity, description, seller, quantity, id

This is fulfilled by deleting foreign key around these tables.

**Checkout [Buyer]**

| |
|---|
| Description:<br>A buyer wants to purchase an item |
| Step-by-step Description:<br>1. [Buyer] The buyer check out his shopping cart.<br>2. [Buyer] The buyer confirms the shipping address<br>3. [Buyer] The buyer confirms the payment method<br>4. [System] The system accepts the order, updates items' information, credits buyer and seller, returns a confirmation message<br>5. [System] The system notify the seller about this order |

Related Classes: Buyer, Item, Order, ShoppingCart, payment methods, role

Query:

- Command: insert
- Contend: foreign

The system adds new information in the foreign key column between these tables to represent new subclass, association.

Add new addresses information [Buyer]

| |
|---|
| Description:<br>Buyer inserts his/her address information. |
| Step-by-step Description:<br>1. [Buyer] The buyer insert addresses information to the system.<br>2. [System] The system saves addresses information, returns a confirmation message to the buyer. |

Related Classes: Buyer, address, role

Query:

- Command: insert
- Contend: foreign keys

The system adds new information in the foreign key column between these tables to represent new subclass, composition.

**Delete existing addresses information [Buyer]**

| |
|---|
| Description:<br>Buyer deletes his/her address information. |
| Step-by-step Description:<br>1. [Buyer] The buyer choose which addresses information to delete.<br>2. [System] The system deletes addresses information, returns a confirmation message to the buyer. |

Related Classes: Buyer, address, role

Query:

- Command: delete
- Contend: foreign keys

The system deletes information in the foreign key column between these tables to represent new subclass, composition.

**Show the newest item information list [Buyer]**

| |
|---|
| Description:<br>Buyer finds the newest items which seller post on the system immediately. |

| |
|---|
| Step-by-step Description:<br>1. [System] The system presents out the latest 20 items automatically. |

Related Classes: Buyer, item

Query:

- Command: select
- Contend: name, price, quantity, description, seller, id

Select information from item table by specifying the last item id.

**Add new payment information [Buyer]**

| |
|---|
| Description:<br>Buyer inserts his/her payment information. |
| Step-by-step Description:<br>1. [Buyer] The buyer adds one payment method as a normal way of checking out.<br>2. [System] The system saves the payment method. |

Related Classes: Buyer, paymentmethod, role

Query:

- Command: insert
- Contend: foreign keys

Add foreign key between three tables to represent new subclass and multivalued attributes.

Delete existing payment information [Buyer]

| |
|---|
| Description:<br>Buyer deletes his/her payment information. |
| Step-by-step Description:<br>1. [Buyer] The buyer selects one payment method to be deleted.<br>2. [System] The system deletes the payment method. |

Related Classes: Buyer, paymentmethod, role

Query:

- Command: delete
- Contend: foreign keys

Delete foreign key between three tables to represent new subclass and multivalued attributes.

List buying history [Buyer]

| |
|---|
| Description:<br>Buyer lists all his buying history. |
| Step-by-step Description:<br>1. [Buyer] The buyer chooses to buy history function in the system.<br>2. [System] The system searches all buying history, returns information including buying data, production, quantities, total price, addresses, etc. |

Related Classes: Buyer, Item, Order, ShoppingCart

Query:

- Command: Select
- Contend: CreateTime, address, buyer, name, price, quantity, description, seller, quantity, id

Select information from item table then inserts new information into Order and ShoppingCart table. The association between Buyer and Item are a shopping cart.

## 3. Implementation

For our whole system, we simply divide the use cases in two parts. The first one is focus on seller, the second on buyer. Here, we only take few use cases of our project as examples to present in our final report. Both seller and buyer are parts of the main class "Role". When we works on the initialization parts of the schema, we set up the foreign key role references on the role id.

```
create table if not exists `Seller`(
        `role` int primary key auto_increment,
        foreign key (`role`) references `Role`(`id`)
        on update cascade on delete cascade
);
```

**Seller: Post**

We cannot do everything of this project only based on the SQL. The schema and SQL query only specify the basic operations of what the seller and buyer can do in this system. That's the reason we introduce the api.server.js here to help us finish several works. So, we apply the app to define the api like services.seller.createItem and services.seller.listItem.

```
app.post('/api/seller/:uid/item/create', services.seller.createItem);
app.get('/api/seller/:uid/item/list', services.seller.listItems);
app.put('/api/seller/:uid/item/edit/:iid', services.seller.editItem);
app.delete('/api/seller/:uid/item/delete/:iid', services.seller.deleteItem);
app.get('/api/seller/:uid/order/list', services.seller.listOrders);
app.get('/api/seller/:uid/order/:oid/items', services.seller.getOrderItems);
```

The seller post use case is the basic one of those different use cases. We first create a new table in the db initial file called Item to let all the operations work well. One seller wants to start the selling process, the first thing has to finish is to post the item on the system. Seller fills in the item information, including name, quantity, price, description, etc. The system saves the new

item in the database, returns a confirmation message. Since we write a web system, we use js here to embed all SQL queries in the js file. The posting operation is wrote as createItem function in our project. In the server js file, we new a query in the function. And with the help of insert operation, we insert several basic item information into the system includes name, price, quantity, description and seller. Using the question mark to set the basic information when the seller input the item data. Then, the client can easily use this function with the uid and item parameters. The query shown as below:

```
"insert into `Item` " +
"(`name`, `price`, `quantity`, `description`, `seller`) values (?,?,?,?,?)",
```

In the end, we can make the connection between the web surface and our db system. The seller can register and login to our Mystore system with simple Id. Once the seller login the system, he or she can find the green button on the top of the home page "Your inventory". Post your new item when the user press the button can be realized. Several blanks we be listed in the page. All blanks are the basic information which we generated in the query and table schema such as name, price and quantity. After all the operations were finished, the item will be added into the personal database and seller can find his item on the scale or the item history.

**Deletion**

Same as the post use case we mentioned before. The deletion is also the basic operation we need to realize for the selling system. Because there is no other difference for the property of those two use cases, we don't need to create new tables. The api and item table are totally same. When someone wants to delete an item, it means the id will no longer exists after the deletion. What we wrote here is that decide the id and seller with the help of the question. The system does not know which seller wants to delete which item, the decision is all made by sellers. The query is shown as below:

```
query.add(
    "delete from `Item` where `id`=? and `seller`=? and `order` is null",
    [itemID, role]);
```

It is the most easiest operation for us since the deletion operation already exists in the SQL just like insert or update. We introduce the delete operation here and one line can finish the function we want. It is also simple for the seller. When the seller does not have this one item or no longer want to sell one item any more, he can just click the red delete button on the item scale page. The item will be deleted soon.

**Buyer**

Here are two use cases working on Address Class. Here is the definition of Address Class.

```
create table if not exists `Address`(
    `id` int primary key auto_increment,
    `name` varchar(200) not null,
    `phone` varchar(200) not null,
    `postal` varchar(200) not null,
    `street` varchar(200) not null,
```

```
    `apt` varchar(200) not null,
    `state` varchar(200) not null,
    `country` varchar(200) not null,
    `role` int,
    foreign key (`role`) references `Role`(`id`)
    on update cascade on delete set null
);
```

1. Add new addresses information [Buyer]

Description:

Buyer insert his/her address information.

Step-by-step Description:

1. [Buyer] The buyer insert addresses information to system.
2. [System] The system saves addresses information, returns confirmation message to buyer.

First of all, we find the API

app.post('/api/buyer/:uid/address/create', services.buyer.createAddress);

(From dbms_project/server/api.server.js)

Here is part of the definition of function createAddress (From dbms_project/server/services/buyer.service.server.js)

```
        query.add(
            "insert into `Address` " +
            "(`name`,`phone`,`postal`,`street`,`apt`,`state`,`country`,`role`) " +
            "values (?,?,?,?,?,?,?,?)",
            [addr.name, addr.phone, addr.postal, addr.street, addr.apt, addr.state, addr.country,
role]);
        query.execute();
```

In practice, we fill in name, phone, postal, street, apt, state and country in to 'Address' Table, then click 'save'. All these information is saved in the server.

2. Delete existing addresses information [Buyer]

Description:

Buyer deletes his/her address information.

Step-by-step Description:

1. [Buyer] The buyer choose which addresses information to delete.
2. [System] The system deletes addresses information, returns confirmation message to buyer.

First of all, we find the API.

app.delete('/api/buyer/:uid/address/delete/:aid', services.buyer.deleteAddress);

(From dbms_project/server/api.server.js)

Here is part of the definition of function createAddress (From dbms_project/server/services/buyer.service.server.js)

```
  function deleteAddress(req, res) {
        query.add(
            "delete from `Address` where `role`=? and `id`=?",
```

```
            [role, addrID]);
        query.execute();
}
```

In practice, after created a new address, there is a 'delete' button under 'Action' column. Click this button when you want to delete this address.

## 4. Discussion

In the former part, we describe the basic requirements for this project. We design two roles with buyer and seller to separate different use cases. We set up a web to realize those requirements. For the seller, the basic operation is to post a new item. The seller can sign in the home page to Mystore. The first button is used to add new item with the input of name, price, quantity and the description. All those steps must be finished by seller. The item result will be shown in the item on scale. The scale or list shown below the add blank. The three basic requirements for posted items were all met with the action column. The delete operation will delete the whole part of one item. The edit operation can change the description, quantity and name conveniently. History of orders can be found on the right side of top bar shown on the seller home page. The history of orders could show the specific information of one order. It records the create time, buyer name, telephone number and home address.

The first line of customer home page is the searching bar. Typing the characters or letters of one item, it will search the proper item which satisfy the requirement of buyer. All items will be shown with the date order. The first item on the top line always be the newest item seller posted. The buyer can choose the quantity and add to personal shopping cart to make the decision later. The modification of the quantity will depend on the real quantity of that item. The system does not accept the amount of customer's order is larger than the actual amount. All those operations completed in the cart page of customer. Same as the delete function of delete, buyer can remove those orders they do not want in the shopping cart. The result added or removed shown in the list below the cart. Once one decide to buy the item in the shopping cart, the buyer can make the checkout. When it is the first time customers eager to buy something, they are required to type the address and payment information. The delete function can also be realized once they want to delete information they no longer used. The history item list can be found on other pages for buyer.

For the integrality of the system, we introduce the profile page to let both seller and buyer perfect the personal information. For the first time, seller or buyer register new account, they will be asked to add several personal information. The profile could show all details of the single customer or seller. We nearly meet all requirements in our UML and use cases which mentioned in the proposal. Our Mystore system is a real online store which can be used immediately. Although the system work is nearly done, we still want to make several improvements which could make our system more attractive to customers and sellers. The shipment sub system is what we intend to do but delete at the beginning of the project. Since it is a huge part of the whole system, we do not have enough time and ability to deal with it properly. We can still add this part in this system in the future.

## 5. Conclusion

After reviewing and testing our work, we have accomplished all the functions of all use cases. In addition, some extra function is also accomplished, such as registration and log-in, showing the newest items for buyers. As good as it is now, there can still be made many improvements.
From this project, we learned how to apply what we have learned in Database management class into practice, such as UML design, use case description, query writing, JDBC design, etc. Some other techniques, such as front-end design, client-server design, are also applied to this project, which help us better develop this online-shopping platform.

## 6.References


1. Angularjs API: https://docs.angularjs.org/api
2. Bootstrap API: http://getbootstrap.com/
3. Nodejs API: https://nodejs.org/dist/latest-v6.x/docs/api/
4. MySQL API: https://github.com/mysqljs/mysql
5. Online-Shopping: https://github.com/mysqljs/mysql