OrderManager

A Database for Managing Products and Orders - JDBC

By: Yongliang(Sean) Tan & Yuzhou Wu

https://github.ccs.neu.edu/seantanty/cs5200_OrderManager_Project_Yongliang_Tan_and_Yuzhou_Wu

Agenda

- 1. Design(Data Definition Language)
  - Requirements
  - Decisions and assumptions
  - Specific design issues
- Functions
  - Data Manipulate Language
  - Data Query Language
- Testing
- Top Level Document
- Future: Issues and Improvements

Design
- Requirements

- Background : This is project is a database portion of a an application for managing products, product inventory, and customer orders for an online store.
- Database system used: JDBC – Derby.

- Key aspects:
  - manage information about products that can be sold to customers
  - track current inventories of products
  - process orders for products from customers

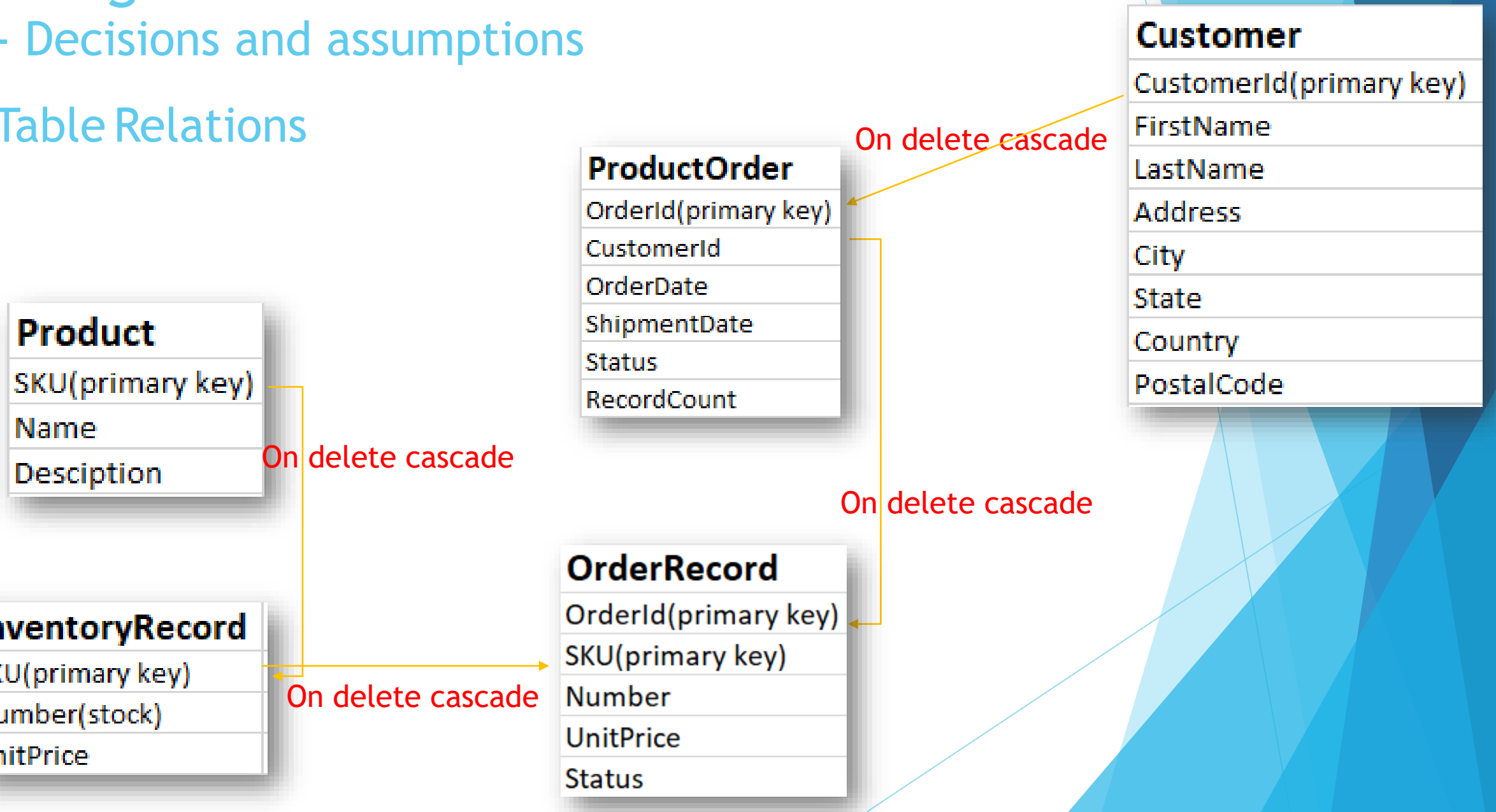
Design
- Requirements

- Product: SKU is a 12-character value of the form AA-NNNNNN-CC where A is an upper-case letter, N is a digit from 0-9, and C is either a digit or an upper case letter. For example, "AB-123456-0N“
- InventoryRecord: price per unit for the current inventory is a positive number with 2 digits after the decimal place
- Customer: payment information is not part of this database
- Order:
  - shipment date(date format, null means not shipped yet)
  - All items must be available in a single transaction to place an order**
- OrderRecord: the item must be available and the inventory is automatically reduced when an order record is created for an order

Design

- Decisions and assumptions

Table Relations



Design

- Decisions and assumptions

Data type and stored functions

Product		
SKU(primary key)		check (isSKU(SKU))
Name		varchar(32) not null
Desciption		varchar(512) not null

```
//create stored function isSKU
String createFunction_isSKU =
     "CREATE FUNCTION isSKU("
    + "     SKU varchar(12)"
    + " )  RETURNS BOOLEAN"
    + " PARAMETER STYLE JAVA"
    + " LANGUAGE JAVA"
    + " DETERMINISTIC"
    + " NO SQL"
    + " EXTERNAL NAME"
    + "     'OrderManager.isSKU'";
stmt.executeUpdate(createFunction_isSKU);
```

```
/**
 * Determines whether SKU is a valid SKU.
 *
 * @param SKU the SKU
 * @return true if SKU is a valid SKU
 */
public static boolean isSKU(String sku) {
        //The SKU is a 12-character value of the form AA-NNNNNN-CC
        //where A is an upper-case letter, N is a digit from 0-9, and C is either a digit or an upper-case letter.
        //For example, "AB-123456-0N".
        return (sku != null) && sku.matches("([A-Z]{2})-(\\d{6})-([A-Z0-9]{2})$");
}
```

Design

- Decisions and assumptions

Data type and stored functions

InventoryRecord	
SKU(primary key)	varchar(12) not null
Number(stock)	int not null, check (Number >= 0)
UnitPrice	decimal(18,2) not null, check (isUnitPrice(UnitPrice)

Unit price: default in USD

```
//create stored function isUnitPrice
String createFunction_isUnitPrice =
          "CREATE FUNCTION isUnitPrice("
    + "      UnitPrice decimal(18,2)"
    + " )  RETURNS BOOLEAN"
    + " PARAMETER STYLE JAVA"
    + " LANGUAGE JAVA"
    + " DETERMINISTIC"
    + " NO SQL"
    + " EXTERNAL NAME"
    + "      'OrderManager.isUnitPrice'";
stmt.executeUpdate(createFunction_isUnitPrice);
```

```
/**
 * Determines whether unitPrice is a valid unitPrice.
 *
 * @param unitPrice the unitPrice
 * @return true if unitPrice is a valid unitPrice
 */
public static boolean isUnitPrice(BigDecimal unitPrice) {
        //The UnitPrice is a positive number with 2 digits after the decimal place
        //boolean fail = (BigDecimal.valueOf(unitPrice).scale() > 2);
        //return (unitPrice > 0) && fail;
        boolean fail = unitPrice.scale() > 2;
        return (unitPrice.doubleValue() > 0.0) && (!fail);
}
```

Design

- Decisions and assumptions

Data type and stored functions

Customer		
CustomerId(primary key)		int not null
FirstName		varchar(32) not null
LastName		varchar(32) not null
Address		varchar(128) not null
City		varchar(32) not null
State		varchar(32), check (isValidStateOrNull(State))
Country		varchar(32) not null, check (isValidCountry(Country))
PostalCode		int not null

```
//create stored function isValidStateOrNull
String createFunction_isValidStateOrNull =
          "CREATE FUNCTION isValidStateOrNull("
    + "     State varchar(32)"
    + " )  RETURNS BOOLEAN"
    + " PARAMETER STYLE JAVA"
    + " LANGUAGE JAVA"
    + " DETERMINISTIC"
    + " NO SQL"
    + " EXTERNAL NAME"
    + "     'OrderManager.isValidStateOrNull'";
stmt.executeUpdate(createFunction_isValidStateOrNull);
```

```
//state enum to support isValidStateOrNull function
enum State{
        AL,AK,AZ,AR,CA,CO,CT,DE,FL,GA,HI,ID,IL,IN,IA,KS,KY,LA,ME,MD,MA,MI,MN,MS,MO,MT,NE
        ,NV,NH,NJ,NM,NY,NC,ND,OH,OK,OR,PA,RI,SC,SD,TN,TX,UT,VT,VA,WA,WV,WI,WY,DC;
}

/**
 * Determines whether state is a valid state or null.
 *
 * @param state the state
 * @return true if state is a valid state or null
 */
public static boolean isValidStateOrNull(String state) {
        if (state == null) {
                return true;
        } else {
                for(State s: State.values()) {
                        if(s.name().equals(state)) {
                                return true;
                        }
                }
                return false;
        }
}
```

Design

- Decisions and assumptions

Data type and stored functions

https://github.com/TakahikoKawasaki/nv-i18n

```
//create stored function isValidCountry
String createFunction_isValidCountry =
          "CREATE FUNCTION isValidCountry("
    + "      Country varchar(32)"
    + " )  RETURNS BOOLEAN"
    + " PARAMETER STYLE JAVA"
    + " LANGUAGE JAVA"
    + " DETERMINISTIC"
    + " NO SQL"
    + " EXTERNAL NAME"
    + "      'OrderManager.isValidCountry'";
stmt.executeUpdate(createFunction_isValidCountry);
```

```
/**
 * Determines whether country is a valid country name.
 *
 * @param country the country
 * @return true if country is a valid country name
 */
public static boolean isValidCountry(String country) {
        for(CountryCode c: CountryCode.values()) {
                if(c.getName().equals(country)) {
                        return true;
                }
        }
        return false;
}
```

Design

- Decisions and assumptions

Data type and stored functions

ProductOrder		
OrderId(primary key)		int not null
CustomerId		int not null
OrderDate		date not null
ShipmentDate		date
Status		int default 0 (0 pending, -1 incomplete, 1 complete)
RecordCount		int not null

Design

- Decisions and assumptions

Data type and stored functions

OrderRecord		
OrderId(primary key)		int not null
SKU(primary key)		varchar(12) not null
Number		int not null, check (Number >= 0)
UnitPrice		decimal(18,2) not null, check (isUnitPrice(UnitPrice))
Status		int not null default 0

```
//create stored function isUnitPrice
String createFunction_isUnitPrice =
          "CREATE FUNCTION isUnitPrice("
    + "      UnitPrice decimal(18,2)"
    + " )  RETURNS BOOLEAN"
    + " PARAMETER STYLE JAVA"
    + " LANGUAGE JAVA"
    + " DETERMINISTIC"
    + " NO SQL"
    + " EXTERNAL NAME"
    + "      'OrderManager.isUnitPrice'";
stmt.executeUpdate(createFunction_isUnitPrice);
```

```
/**
 * Determines whether unitPrice is a valid unitPrice.
 *
 * @param unitPrice the unitPrice
 * @return true if unitPrice is a valid unitPrice
 */
public static boolean isUnitPrice(BigDecimal unitPrice) {
        //The UnitPrice is a positive number with 2 digits after the decimal place
        //boolean fail = (BigDecimal.valueOf(unitPrice).scale() > 2);
        //return (unitPrice > 0) && fail;
        boolean fail = unitPrice.scale() > 2;
        return (unitPrice.doubleValue() > 0.0) && (!fail);
}
```

Design

- Decisions and assumptions

Triggers & Stored procedure

```
/**
 * This function prints error message of inventory out of stock on calling.  Use in stored procedure
 *
 * @param SKU The SKU that is not sufficient to fulfill order
 * @param OrderId The Order ID that has certain OrderRecord exceed inventory stock
 */
public static void notSufficientInventory(String SKU, int OrderId){
        System.err.printf("OrderID %d, %s is out of stock\n", OrderId, SKU);
}
```

```
//create stored procedure notSufficientInventory
String createProcedure_notSufficientInventory =
          "CREATE Procedure notSufficientInventory("
    + "      SKU varchar(12),"
    + "      OrderId int"
    + " )   "
    + " PARAMETER STYLE JAVA"
    + " LANGUAGE JAVA"
    + " DETERMINISTIC"
    + " NO SQL"
    + " EXTERNAL NAME"
    + "      'OrderManager.notSufficientInventory'";
stmt.executeUpdate(createProcedure_notSufficientInventory);
```

https://db.apache.org/derby/docs/10.2/ref/

```
//check OrderRecord's number doesn't exceed inventory stock
String createTrigger_notSufficientInventory =
              "create trigger notSufficientInventory"
            + " no cascade before insert on OrderRecord"
            + " referencing new as insertedOrderRecord"
            + " for each row mode db2sql"
            + " when (insertedOrderRecord.Number > "
            + " (select InventoryRecord.Number from InventoryRecord where insertedOrderRecord.SKU = InventoryRecord.SKU))"
            + " call notSufficientInventory(insertedOrderRecord.SKU, insertedOrderRecord.OrderId)";
stmt.executeUpdate(createTrigger_notSufficientInventory);
```

Design

- Decisions and assumptions

Triggers & Stored procedure

```
//adding number in OrderRecord will decrease the respecting inventory stock number
String createTrigger_DecreaseInventoryAfterAddOrderRecord =
              "create trigger DecreaseInventoryAfterAddOrderRecord"
            + " after insert on OrderRecord"
            + " referencing new as insertedOrderRecord"
            + " for each row mode db2sql"
            + " when (insertedOrderRecord.Number <= "
            + " (select InventoryRecord.Number from InventoryRecord where insertedOrderRecord.SKU = InventoryRecord.SKU))"
            + " Update InventoryRecord "
            + " SET Number = Number - insertedOrderRecord.Number where insertedOrderRecord.SKU = InventoryRecord.SKU";
stmt.executeUpdate(createTrigger_DecreaseInventoryAfterAddOrderRecord);
```

```
//create trigger for deleting OrderRecord
String createTrigger_DeleteOrderRecord =
              "create trigger DeleteOrderRecord"
            + " after delete on OrderRecord"
            + " referencing old as deletedOrderRecord"
            + " for each row mode db2sql"
            + " Update InventoryRecord "
            + " SET Number = Number + deletedOrderRecord.Number where InventoryRecord.SKU = deletedOrderRecord.SKU";
stmt.executeUpdate(createTrigger_DeleteOrderRecord);
```

Design
- Specific design issues

- Restrictions:
  - Deleting of Products, Inventory, and Customer should be strictly restricted
- Order: All items must be available in a single transaction to place an order
  - Decided to add status field instead, let customer could order the products with sufficient inventory
  - The orderrecords which exceed current inventory stock will store in system, but that order will be marked as incompelte
  - Could add an email notification function later to enhance the sale

Function

- Data Manipulate Language

Product aspect

```
// insert into product
PreparedStatement insertRow_Product = conn.prepareStatement(
        "insert into Product values(?, ?, ?)");

// insert into inventoryRecord
PreparedStatement insertRow_InventoryRecord = conn.prepareStatement(
        "insert into InventoryRecord values(?, ?, ?)");
// update inventoryRecord
PreparedStatement updateRow_InventoryRecord = conn.prepareStatement(
        "update InventoryRecord set Number = ?, UnitPrice = ? where SKU = ?");
```

- **addProduct**: Insert data into table ‘Product’
- **addInventory**: Insert data into table ‘InventoryRecord’
- **updateInventory**: Update data in the table ‘InventoryRecord’

  eg. Change price, add inventory number

Function

- Data Manipulate Language

Product aspect

```
static public int checkInventoryExistence(PreparedStatement selectRow_InventoryRecord, String SKU) throws SQLException{
    ResultSet rs;
    int count = -1;
    selectRow_InventoryRecord.setString(1, SKU);
    rs = selectRow_InventoryRecord.executeQuery();
    if(rs.next()) count = rs.getInt(1);
    return count;
}
```

checkInventoryExistence

```
// check number of some product in InventoryRecord
PreparedStatement selectRow_InventoryRecord = conn.prepareStatement(
        "select Number from InventoryRecord where SKU = ?");
```

Three types of return

-1: means there is no record in inventory record before, so we should insert

0: means it was in the inventory record but it is now out of store

Other: the count of number present in the inventory record.

Function

- Data Manipulate Language

Customer aspect

- addCustomer
- updateCustomer

FirstName
LastName
Address
City
State
Country
Postalcode
CustomerId

Function

- Data Manipulate Language

Order aspect

```
// insert to ProductOrder
PreparedStatement insertRow_Order = conn.prepareStatement(
        "insert into ProductOrder values(?, ?, ?, ?, ?, ?)");
```

```
static public void addOrder(PreparedStatement stmt_Order, int CustomerId,
        int OrderId, int Count) throws SQLException {

    long millis=System.currentTimeMillis();
    Date orderDate=new java.sql.Date(millis);

    stmt_Order.setInt(1,CustomerId);
    stmt_Order.setInt(2,OrderId);
    stmt_Order.setDate(3, orderDate);
    stmt_Order.setDate(4, null);
    stmt_Order.setInt(5, 0);
    stmt_Order.setInt(6, Count);

    stmt_Order.execute();
    System.out.printf("%s added to Order\n", OrderId);
}
```

CustomerId
OrderId
OrderDate
ShipmentDate
Status
Count

addOrder

Add data to table ‘ProductOrder’
First set shipmentDate to null and status to 0 (pending)
These two parameter will change after we load all OrderRecord

Count: the number of orderRecords related to this order

Function

- Data Manipulate Language

Order aspect

```
PreparedStatement insertRow_OrderRecord = conn.prepareStatement(
        "insert into OrderRecord values(?, ?, ?, ?, ?)");
```

addOrderRecord

OrderId
SKU
Number
UnitPrice
Status

```
static public void addOrderRecord(Connection conn, PreparedStatement stmt_OrderRecord, int OrderId, String SKU,
        int Number, double UnitPrice) throws SQLException {
    stmt_OrderRecord.setInt(1, OrderId);
    stmt_OrderRecord.setString(2, SKU);
    stmt_OrderRecord.setInt(3, Number);
    stmt_OrderRecord.setDouble(4, UnitPrice);;

    PreparedStatement checkInventoryNumber = conn.prepareStatement("select number from InventoryRecord where SKU = ?");
    checkInventoryNumber.setString(1, SKU);
    ResultSet rs = checkInventoryNumber.executeQuery();
    int InventoryNumber = 0;
    if(rs.next()) InventoryNumber = rs.getInt(1);
    if(InventoryNumber >= Number) stmt_OrderRecord.setInt(5, 1);
    else stmt_OrderRecord.setInt(5, 0);

    stmt_OrderRecord.execute();
    System.out.printf("OrderId %s with SKU %s added to OrderRecord\n", OrderId, SKU);
}
```

Status: if number is larger than the inventory number, Status will be 0 (fail). Otherwise, status will be 1(success).

Function

- Data Manipulate Language

Order aspect

Get number of records whose status is 1 for a specific order

```java
/**
 * DML function to get number of complete orderRecords
 * @param conn SQL connection to pass in
 * @param OrderId  order id
 * @throws SQLException  Potential SQL exception
 */
static public int checkOrder(Connection conn, int OrderId) throws SQLException {
    PreparedStatement checkStatus = conn.prepareStatement("Select Status from  OrderRecord where OrderId = ?");
    checkStatus.setInt(1, OrderId);
    int RecordCount = 0;
    ResultSet rs = checkStatus.executeQuery();
    while(rs.next())
        if(rs.getInt(1) == 1)RecordCount++;
    return RecordCount;
}
```

- **CancelOrder**

After we load all order records, we scan Order for another time. We will compare attribute ‘Count’ to this number of complete records. If not equal, we will cancel it.

- **PlaceOrder**

Then for the rest orders, it means that they are all complete and we should place these orders. So we need to set the ‘status’ in table ‘Order’ to 1(complete).

Function

- Data Manipulate Language

Order aspect

```
// cancel order
PreparedStatement cancel_Order = conn.prepareStatement(
        "update ProductOrder set Status = ? where OrderId = ?");

// place order
PreparedStatement place_Order = conn.prepareStatement(
        "update ProductOrder set Status = ?, ShipmentDate = ? where Status = ?");
```

```
static public void CancelOrder(PreparedStatement updateRow_Order,
        int status, int OrderId) throws SQLException {
    updateRow_Order.setInt(1, status);
    updateRow_Order.setInt(2, OrderId);

    updateRow_Order.execute();
    System.out.printf("Order %d cancelled.\n", OrderId);
}
```

```
static public void PlaceOrder(PreparedStatement updateRow_Order,
        int status, int oldStatus) throws SQLException {
    updateRow_Order.setInt(1, status);
    long millis=System.currentTimeMillis();
    Date shipmentDate=new java.sql.Date(millis);
    updateRow_Order.setDate(2, shipmentDate);
    updateRow_Order.setInt(3, oldStatus);

    updateRow_Order.execute();
}
```

Function

- Data Manipulate Language

Order aspect

```
// update shipment date
PreparedStatement updateShipment = conn.prepareStatement(
        "update ProductOrder set ShipmentDate = ? where OrderId = ?");
```

```
static public void updateShipment(PreparedStatement updateRow_Order,
        Date shippingDate, int OrderId) throws SQLException {
    updateRow_Order.setDate(1, shippingDate);
    updateRow_Order.setInt(2, OrderId);

    updateRow_Order.execute();
    System.out.printf("Order %d shippping time is:" + shippingDate, OrderId);
    System.out.printf("\n");
}
```

Function

- Data Query Language

Print all table

```
static void printProducts(Connection conn) throws SQLException {

static void printInventoryRecord(Connection conn) throws SQLException {

static void printCustomer(Connection conn) throws SQLException {

static void printOrder(Connection conn) throws SQLException {

static void printOrderRecord(Connection conn) throws SQLException {
```

Function

- Data Query Language

```
static void showDescription(Connection conn, String SKU) throws SQLException {

        PreparedStatement showDescription = conn.prepareStatement(
                    "select Name, Description from Product where sku = ?");

        showDescription.setString(1, SKU);
        ResultSet rs = showDescription.executeQuery();

        if (rs.next()) {
            String name = rs.getString(1);
            String description = rs.getString(2);
            System.out.printf("%s   %s \n",  name, description);
        }
}
```

Example:

```
Xiaomi 10       Another Chinese brand
```

Show the information about product & show customer information

```
static void showCustomerInformation(Connection conn, int id) throws SQLException {

    PreparedStatement showCustomerInformation = conn.prepareStatement(
                "select FirstName, LastName, Address, City, State, Country from Customer where CustomerId = ?");

    showCustomerInformation.setInt(1, id);
    ResultSet rs = showCustomerInformation.executeQuery();

    if (rs.next()) {
        String name = rs.getString(1) + " " + rs.getString(2);
        String detailedAddress = rs.getString(3) + ", " + rs.getString(4) + ", " +rs.getString(5) + ", "+
                rs.getString(6);
        System.out.printf("%s   %s \n",  name, detailedAddress);
    }
}
```

Example:

```
YUZHOU WU       5805 Charlotte Dr, San Jose, CA, United States
```

Function

- Data Query Language

Check shipment date

```
static void checkShippment(Connection conn, int orderId) throws SQLException {

        PreparedStatement checkShippment = conn.prepareStatement(
                                "select ShipmentDate from ProductOrder where OrderId = ?");

        checkShippment.setInt(1, orderId);
        ResultSet rs = checkShippment.executeQuery();

        if (rs.next()) {
                Date date = rs.getDate(1);
                if(date != null)System.out.printf("Order %d shipping time is: %s \n",orderId, date);
                else System.out.printf("Order %d pending, not shipped.\n", orderId);
        }
}
```

Example:

```
Order 1 shipping time is: 2019-06-20
Order 2 shipping time is: 2019-06-30
Order 7 pending, not shipped.
```

Testing

Product and product inventory information before processing order

```
Product:
SKU             Name            Description
AA-000000-0A    iPhone XR       All-screen design...
AA-000000-0B    iPhone XS       super Retina in two sizes...
AA-000000-0C    iphone 8        No description
AA-000000-0D    iphone 7        No description
AA-000000-0E    iphone 6        No description
CN-000000-0A    HUAWEI P9       Chinese brand
CN-000000-0B    Xiaomi 10       Another Chinese brand
Product count: 7
-----------------------------------------------------------------------------------------------
InventoryRecord:
sku             Number  unitprice
AA-000000-0A    100     749.000000
AA-000000-0B    100     999.000000
AA-000000-0C    100     599.000000
AA-000000-0D    100     449.000000
AA-000000-0E    200     399.000000
CN-000000-0A    200     399.250000
CN-000000-0B    400     300.050000
Inventory count: 7
```

Testing

Customer and order information before processing order

```
Customer:
FirstName       LastName        Address                 City            State   Country         PostalCode      CustomerId
LINYI           GAO             5805 Charlotte Dr       San Jose        CA      United States   95123           4
SEAN            TAN             5805 Charlotte Dr       San Jose        CA      United States   95123           2
YUZHOU          WU              5805 Charlotte Dr       San Jose        CA      United States   95123           1
ZHONG           ZHUANG          5805 Charlotte Dr       San Jose        CA      United States   95123           3
Customer count: 4
-------------------------------------------------------------------------------------------------
Order:
CustomerId      OrderId         OrderDate       ShipmentDate            Status
4               8               2019-06-22      null            Pending
4               7               2019-06-22      null            Pending
4               6               2019-06-22      null            Pending
4               5               2019-06-22      null            Pending
3               4               2019-06-22      null            Pending
2               3               2019-06-22      null            Pending
1               2               2019-06-22      null            Pending
1               1               2019-06-22      null            Pending
Order count: 8
```

Testing

After processing order

```
---------------------------------------------------------------------------------------------------
InventoryRecord:
sku             Number  unitprice
AA-000000-0A    100     749.000000
AA-000000-0B    0       999.000000
AA-000000-0C    7       599.000000
AA-000000-0D    79      449.000000
AA-000000-0E    168     399.000000
CN-000000-0A    0       399.250000
CN-000000-0B    1       300.050000
Inventory count: 7
```

```
Order:
CustomerId      OrderId         OrderDate       ShipmentDate            Status
4               8               2019-06-25      2019-06-25              Complete
4               7               2019-06-25      null            Incomplete
4               6               2019-06-25      2019-06-25              Complete
4               5               2019-06-25      null            Incomplete
3               4               2019-06-25      2019-06-30              Complete
2               3               2019-06-25      null            Incomplete
1               2               2019-06-25      2019-06-30              Complete
1               1               2019-06-25      2019-06-20              Complete
Order count: 8
---------------------------------------------------------------------------------------------------

OrderRecord:
OrderId         SKU             Number  UnitPrice       Status
1               AA-000000-0C    18      599.0       Complete
1               AA-000000-0E    32      399.0       Complete
2               AA-000000-0C    50      599.0       Complete
3               AA-000000-0A    101     749.0       Incomplete
4               AA-000000-0B    100     999.0       Complete
5               AA-000000-0E    200     399.0       Incomplete
5               AA-000000-0C    25      559.0       Complete
5               AA-000000-0D    21      449.0       Complete
6               CN-000000-0A    200     399.25       Complete
7               CN-000000-0A    1       399.25       Incomplete
8               CN-000000-0B    399     300.05       Complete
OrderRecord count: 11
```

Top Level Document

- ReadMe file to give brief description of this project

- Java Docs:
- https://pages.github.ccs.neu.edu/seantanty/cs5200_OrderManager_Project_Yongliang_Tan_and_Yuzhou_Wu/

- Referencing using other's work, their Java Docs:
- http://takahikokawasaki.github.io/nv-i18n/

Future
- Issues and Improvements

- Order/OrderRecords
  - Current:
    - use a status field, need to call CancelOrder and PlaceOrder function
    - When order created, the order date is set as the system time
  - Concern: need to discuss with client about the business side's decision
- Update Order/OrderRecords
  - Current: didn't implement update function
  - Concern: Once order been placed, the customer could only cancel the whole order
- Back-ordering items:
  - Improvement: When adding inventory, we should first fulfill those incomplete OrderRecords
- Considering convert to use MySQL database instead

Q&A