CSCI-3901-Assignment6-Problem2

Name: Pratheep Kumar Manoharan

Banner Id: B00837436

Files:

common:	Utility.java
constants:	ConnectionProperties.java, ConstantsClass.java
control:	InventoryControl.java, InventoryControlImpl.java
db:	Queries.java, DBHandler.java, DBHandlerImpl.java
exception:	OrderException.java
plainobjects:	PurchaseOrder.java

MainClass.java

CSCI-3901-Assignment6-Problem1.pdf
CSCI-3901-Assignment6-Problem2.pdf
CSCI-3901-Assignment6-Problem2-Argument.pdf
SQL_Statements_Problem2.sql

Objective:

To learn how to make meaningful changes to a database.

Assumptions:

1. Date passed as parameter to Issue_reorder method cannot be
    a. Invalid date
    b. Future date
2. Shipped order cannot be shipped again.
3. While shipping order, the inventory cannot be updated negative.
4. Reorder cannot be placed again for the same date for the same supplier.
5. Reorder can be placed again for the same date for different supplier.
6. Same order cannot be received more than once.

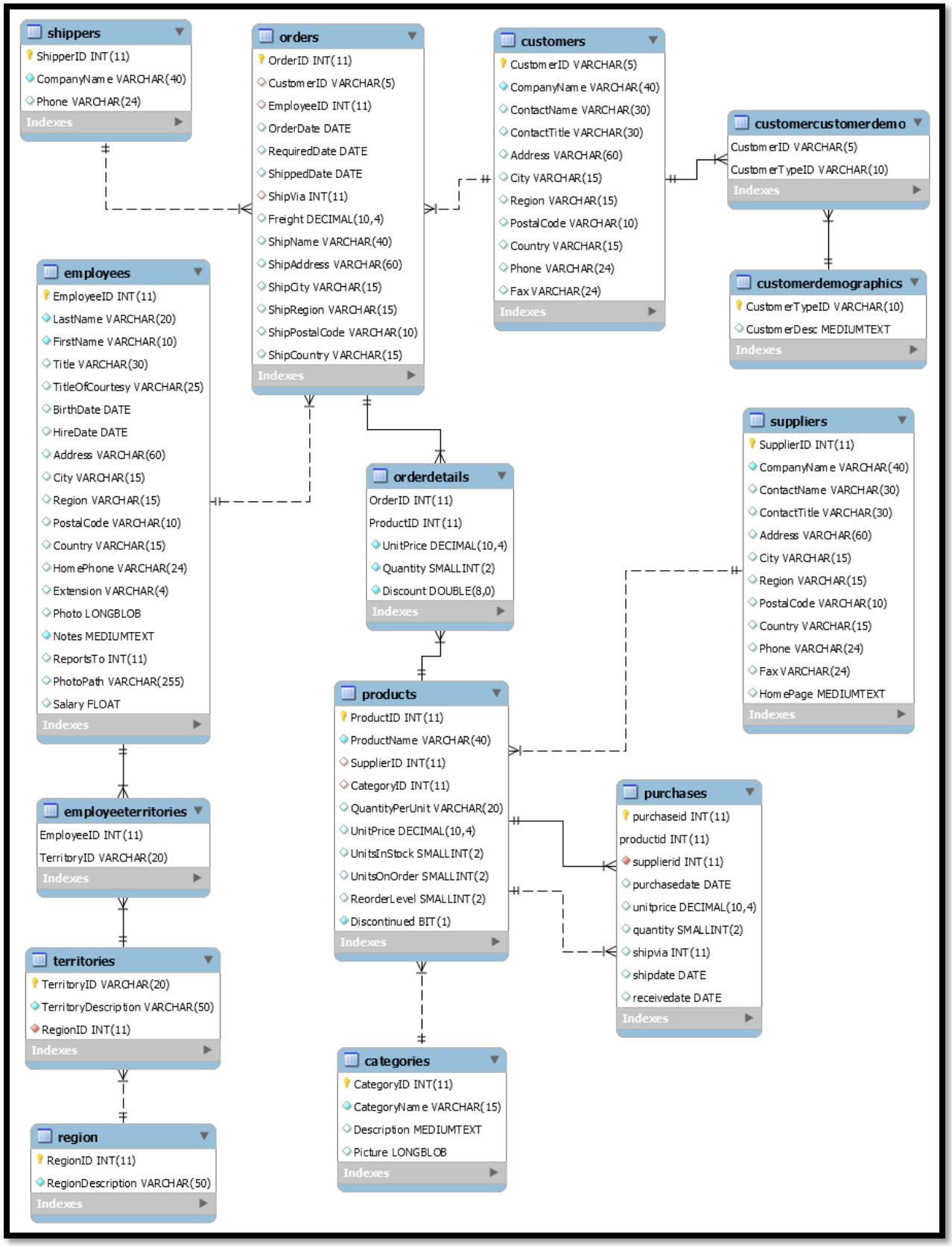
Dependencies used:

mysql-connector-java-8.0.19.jar – to establish connection between Java and SQL.

Entity relationship diagram:

- Added new table "purchases". The productid and the supplierId columns are mapped with the productid and the supplierid columns of this new table purchases. The primary key foreign key relationship is created.
- The primary key is formed with the combination of the purchaseid and the productid columns.
- There is no other change done in the database.

Please find below the entity relationship diagram.



Sequence of operations:

Ship order:

1. Check whether the order is already shipped.
    a. If it is already shipped, then an error message is printed to user.
    b. If it is not shipped, then shipped date will be updated with current date.
2. Check whether we have enough stock to ship.
    a. If we do not have enough stock, then issue reorder is called to place reorder with current date.
    b. If we have enough stock, then proceed to ship.
3. Update the inventory in products table, subtract the quantity shipped from unitsinstock column.
4. When we do not have stock to ship, the reorder quantity is calculated by adding reorder level and the quantity difference of the order.

Issue reorder:

1. Check whether we have any products to reorder.
    a. If no products to reorder, end of execution.
    b. If we have products to reorder, then fetch the details from database.
2. Check whether we have already placed reorder on the same date to the same suppliers.
    a. If we have already placed the reorder, end of execution.
    b. If we are yet to place the reorder, then proceed with reorder.
3. Insert the reorder details into the new table.
4. Update the inventory of reorder in products table, add the quantity of reorder in the unitsonorder column.
5. Return the number of suppliers that we placed order.

Receive order:

1. Check whether we have already received the order.
    a. If yes, then end of execution.
    b. If no, then proceed with receive order.
2. Update the inventory of received order in products table, add the quantity received in unitsinstock column and subtract the received quantity in unitsonorder column.

Technical Code Flow:

MainClass.java

1. The main method is the starting point of the program.
2. The flow of the program is controlled by this class.
3. The main method will get the input from the user and pass it to the controller layer of the program.

ConnectionProperties.java:

Class is used to store the database credentials which is used in DBHandlerClass to connect to the database.

ConstantsClass.java:

Class is used to store all the constants of the program.

Queries:

Each query is created in a separate method in this class.

OrderException.java:

The user defined exception class to carry the exception message and reference to the users.

PurchaseOrder.java:

Java object created to store the purchase table details.

Utility.java:

1. A class to have the common functionalities of the program.
2. convertDateToString method is used in both controller layer and database layer. So, this method is created in Utility and the classes extends this class to use the method.
3. printString method is used in all parts of the program to print the message to the user. So, this method is added in the common class and it is used in all places.
4. To impose reusability of the method, this class is introduced.

InventoryControl.java:

Interface of the controller layer. It connects the main class to the rest of the program.

IventoryControlImpl.java:

1. The implementation class of the controller layer of the program.
2. The basic validations on the user keyed input is done in this layer.
3. Ship_order gets the orderNumber as input and passes that to the database layer. After successful shipping, it will print a success message to the users.
4. dataValidator method validates the user provided date before proceeding to the database layer.
5. *Issue_reorder method*

    a. Gets the user keyed in date as input.

b. Validates the date.
c. If the date is invalid, then the execution stops with an error message to the user.
d. If the date is valid, then the date is converted into a format accepted by database.
e. The converted date is passed to the database layer as a String in that format.
f. The return value from the database layer is passed to the main class.
g. All exceptions of the method are caught and printed to the user in this method.

6. Receive_order method gets the internal order reference as input. It passes the reference to the database layer. After successful execution, the method prints the success message to the user.

DBHandler.java:

The interface of the database layer of the program. All database activities are invoked through this interface.

DBHandlerImple.java:

1. getConnection method is used to create the database connection. This method returns a created database connection which can be used to execute queries in database.

2. *shipOrder method*

a. Gets the orderNumber as input.
b. Gets the database connection.
c. The connection is created with autocommit set as false.
d. Verify the orderNumber – whether it is a valid reference or already shipped.
e. In case failure, the execution stops there.
f. If the input is valid, then check for the available inventory to ship.
g. If the inventory is not available, then the system will call issue_reorder method for this supplier alone with current date as parameter.
h. If the inventory is available, then the shipping details are updated in products table.
i. After all steps, the system will the commit the transaction at the end.

3. *issueReorder method*

a. Gets the date and purchase order object as input.

b. The purchase order object will be null when it the issue reorder method is called by user. The object will have necessary values when it is called from ship order method.
c. Gets the database connection.
d. Fetches the purchases that are already made for the given date. Stores all the purchase references in a list.
e. Retrieves the product information to place reorder.
f. When the issue reorder method is called directly, the reorder is placed for all the suppliers with lesser stock.
g. When the issue reorder method is called from ship order method, the reorder is placed only for that product's supplier for which the stock is not available to ship.
h. The date and supplier combination is validated whether the reorder is already placed for this combination.
i. The created details are persisted into a new purchases table.
j. The reorder inventory details are updated in the products table.
k. The number of suppliers the reorder is placed is returned to the calling method.

4. generatePurchaseId method is used to generate the purchase Id uniquely by combining the date and supplier id which joined with product id is unique for the given day.

5. *receiveOrder method*

a. Gets the purchase Id as the input.
b. Gets the database connection.
c. Checks whether the purchase Id is valid, or the purchase Id is already received.
d. The receive date is updated in purchases table.
e. The received inventory details are updated in the products table.