External Documentation

- Assignment 6

(Version 1.0)

Table of Contents

1. Overview

The goal of this assignment is to modify the existing database of course lab and map the orders to payments in our database modification.

The program will update and extract data under 4 different method
call which are as provided.

- **void reconcilePayments(Connection database)**

  Accept the database connection as parameter and modify the order table by mapping the possible payments to order(s).

- **boolean payOrder(Connection database, float amount, String cheque_number, ArrayList<Integer> orders)**

  Accept the amount, cheque number and list of order and perform certain validation to check if those order can be mapped to the amount and check provided. If possible, Check number is updated into the payment table first and then the orders are updated with the check number.
- **ArrayList<Integer> unpaidOrders(Connection database);**

  After Payments are Reconcile, this method would be called. It will Return all the Orders which have not been paid yet.

- **ArrayList<Integer> unknownPayments(Connection database);**

  After Payments are Reconcile, this method would be called. It will Return all the payments which are not linked to any order and are unknown.

The solution delivered is based on several important factors as the implementation of data, the structure used for storing the details fetched from the database, the queries used to fetch and update the details.

2. Files and external data

Below is the SQL file storing the SQL command to alter the table definition before reconciling the payments.

- **UpdateTable.sql:** SQL file storing queries to alter the payment table- adding the index to the check number and order table- adding the check number as the foreign key to order table.

Below are the java files for the Implementation of the required operation:

- **PaymentManagement.java**: This file acts like a service layer which will accept the call from the driver class and surpass the control to respective dao class, once dao class methods return the value, it will return the values to the driver class.

  N.B: Reconcile Payment method from the PaymentManager.java will be called only once the structure of the order table is changed.

- **ReconcilePaymentDao.java**: Implement methods to perform the operation in given sequence
  - **getCustomerPayment()** – get all the list of payments for customers.
  - **mapOrdertopayment()** – get all the list of orders for the customer, and try to map payments to order(s). call the checkMapping() method to check whether payments and order can be mapped or not. When Mapped, update the order records in order table with check Number.
  - **checkMapping()** – check mapping for the payment with list of orders. If any found returns the list of orders to be mapped.

- **PaymentQueryDao.java**: Implement methods:
  - **Pay Order**: to make the payment for the orders with the given amount and check number. It verifies
    1. If the list of order provided are of same customer.
    2. The Amount of all the order should be equal to amount passed as parameter.
    3. All the order provided in list should be unpaid. If any order already has check number, method will return false and exit.
    4. If the check number passed is not the existing check number.

  If all the above condition is verified, then the check number entry is made in the payment table and further orders are update.
  If any exception is encountered will insertion or updating the transaction is revoked and false is returned.

  - **Unknown Payments**: This method executes a query on the payment table and return list of all the payment whose check number are not linked to any of the order.
  - **Unpaid orders**: This method executes a query on the payment table and return list of all the order which are not paid and whose status are not cancelled or disputed.

- **Customer.java:** Basic Bean Class providing the implementation/blueprint of customer table with attribute as customer number, list of orders and list of payments.

- **Payment.java:** Basic Bean Class providing the implementation/blueprint of the payment table. It consists of payment amount, check number and payment date.
- **Order.java:** Basic Bean Class providing the implementation/blueprint of the order table. It consists of order amount, order number and order date.
- **UserDefinedException.java:** Customized Exception class to handle system exception in customised way.

3. Data structures and their relationship with each other

The program heavily uses the List data Structure to store and process data retrieved from the database.

Most commonly data structure was Array List to store list of payments, list of orders, list of customers along with other few to store integer and string as provided below:

- **List of Customer:** We are storing all the Customer in the list retrieved from the database.

- **List of Payments**: We are storing all the payments in the list retrieved from the database.

- **List of orders**: We are storing all the Employees in the list retrieved from the database.
- **List<Integer>of payment Id**: We are storing all the payment Id in the list retrieved from the database and returning it to method call from driver class for “UnknownPayment” method call.

- **List<Integer>of order Id**: We are storing all the order Id in the list retrieved from the database and returning it to method call from driver class for “unpaidOrder” method call.

4. Approach and Design Elements

Below provided are the steps or the approach we followed for <u>reconciling the payments</u>:

- Initially retrieved all the payment order by the customer, in order to get list of payment for every customer.
- Store all the customer into a list. Iterate the list and retrieve all the related orders in a list for every customer from the order table.
- With list of orders and list of payments for a given customer, map all the orders where one-to-one mapping (one check number paying for one order) of payment check number and order is possible.
- Once one to one payment is done, check for the combination of order in chronological ways where multiple order is being paid by the same payment (check Number).
- While mapping, make sure that orders with order date later to payment date of payment are not considered for mapping.
- In case where the multiple order is to be update with the same check number, we form a substring to be used in SQL query String. Subquery string would be in a way of “where order in (orderid1, order2)”. With this we don’t have to hit the database every time to update single order.

Below provided are the steps or the approach we followed for <u>Paying the Order</u>:

- With the Received Arguments/parameter i.e. list of order id, iterate through the list and perform below validation:
    1. If the list of order provided are of same customer.
    2. The Amount of all the order should be equal to amount passed as parameter.
    3. All the order provided in list should be unpaid. If any order already has check number, method will return false and exit.
    4. If the check number passed is not the existing check number.

- Once the above validation is completed, update the payment detail i.e. check number, customer number and amount with currents date into payment table.
- Once insertion is successful. Update the orders in order table with the check number.
- In case if any of the transaction throws exception, revoke and rollback the previous transaction and throw return as false.

5. Degree of Efficiency

- Since there are possibility of one-to many mappings for check number and order, we are considering all the permutation and combination of orders whose payment can be done through the single payment. Reconcile payment is covering all the possible positive scenario through which mapping can be done.
- For paying the order, we have considered all the validation as explained in previous section, once validation are passed then only the transaction are made.
- For any failed update or insert in transaction, we rollback the whole transaction and return appropriately.
- While reconciling we have optimized the query by forming substring by concatenating order id's and using all of them at once, instead of hitting the database again and again with each single ordered to be updated.