**Project Documentation**

*Course: FIN307 Database Management Systems and Financial Applications*

*Professor:* 崔来中

*TA:* 兰夫铭

Team members: 11912736 肖媛媛, 11913040 Nguyen The Hong Hanh

Project topic: Database Design Using MySQL and PHP/JAVA to Implement the Retail Business Management System

**1. Preparation:**

*1.1 Connect to the database*

First, we need to get the information related to our database from console, including database’s name, username and password.

**public static void** main(String[] args) {  
 Scanner sc = **new** Scanner(System.in);  
 System.out.print(**"Enter DB name: "**);  
 String db = sc.next();  
 System.out.print(**"Enter username: "**);  
 String username=sc.next();  
 System.out.print(**"Enter password: "**);  
 String password=sc.next();  
 connectDB(db, username, password);  
}

The connectDB method is where the java class is connected to mySQL database as well as where queries are created, handled, or passed to be executed in our database. Inside the connectDB method, we create a Driver to get the connection from the given information, which will be wrapped in a try-catch block to avoid connection failure error.

Class.forName(**"com.mysql.cj.jdbc.Driver"**);  
Connection myConn=DriverManager.getConnection(**"jdbc:mysql:// localhost:3306/"**+db,username,password);

*1.2 Create tables*

After the database is connected, we could start setting up the structure for our database by creating tables. Firstly, we put CREATE TABLE queries into a hash map of which keys are tables’ names and corresponding values are queries. An example for “employees” table is given as follows:

HashMap<String, String> createTableQuery = **new** HashMap<String, String>();  
createTableQuery.put(**"employees"**,**"create table employees (eid varchar(3) not null, ename varchar(15),city varchar(15), primary key(eid));"**);

Secondly, we have to get the list of tables in our database and compare it to the list of table needed. If any table is missed, we will get the CREATE TABLE query corresponding to that table executed in SQL. If all tables have already been created, no such queries will be executed.

ArrayList<String> tablesInDB = **new** ArrayList<>();  
**while**(rs.next()) {  
 tablesInDB.add(rs.getString(**"tables\_in\_"**+db));  
}  
**for** (**int** i=0;i<tables.length;i++){  
 **boolean** exist=**false**;  
 *//Check to see if the table exists* **if**(tablesInDB.size()>0){  
 **for**(**int** j=0;j<tablesInDB.size();j++){  
 **if**(tables[i].equals(tablesInDB.get(j))){  
 exist=**true**;  
 **break**;  
 }  
 }  
 }  
 *//If not exists, create one. Otherwise, pass* **if**(exist==**false**){  
 Statement createTable = myConn.createStatement();  
 createTable.executeUpdate(createTableQuery.get(tables[i]));  
 System.out.println(**"Created table "**+tables[i]);  
 }  
}

*1.3 Handle user’s input in console*

To the end of the method connectDB, we have written a block of code to handle any inputs from users. A “do… while” loop is used to make sure that the program still accepts orders from the user after either a successful or a failed query. It only quits if the message “exit” is passed. For a valid input of “show” or “report” procedures (task 1 & 2), a result set will be printed out to the console. For a valid input of “add” procedure (task 3), there will be implicit changes to the database and a message to notify if it’s successfully executed. For inputs that do not match any pre-defined procedures, we will have a warning message printed out and let the user enters another input.

Scanner sc = **new** Scanner(System.in);  
String query;  
**do**{  
 System.out.println(**"Enter your desired procedure (type \"exit\" to quit): "**);  
 query = sc.nextLine();  
 **try** {  
 **if** (query.startsWith(**"show"**) || query.startsWith(**"report"**)) {  
 ResultSet resultSet = stmt.executeQuery(**"call "** + query + **";"**);  
 ResultSetMetaData rsmd = resultSet.getMetaData();  
 **int** columnsNumber = rsmd.getColumnCount();  
 **while** (resultSet.next()) {  
 **for** (**int** i = 1; i <= columnsNumber; i++) {  
 System.out.print(resultSet.getString(i) + **" "**);  
 }  
 System.out.println();  
 }  
 } **else if** (query.startsWith(**"add"**)) {  
 stmt.executeUpdate(**"call "** + query + **";"**);  
 System.out.println(**"Procedure successfully executed."**);  
 } **else if** (!query.equals(**"exit"**)){  
 System.out.println(**"Something wrong with your procedure. Please try again."**);  
 }  
 } **catch** (SQLException e){  
 System.out.println(**"Something wrong with your procedure. Please try again."**);  
 }  
}**while**(!query.equals(**"exit"**));  
System.out.println(**"Database disconnected."**);

**2. MySQL Implementation:**

Before creating any procedures, functions, or triggers, we first drop the existing ones to ensure errors resulting from collapses and procedures/functions/triggers are updated as soon as there is any systematic change. An example regarding DROP queries is provided as follows (stmt is a Statement object):

stmt.executeUpdate(**"drop procedure if exists show\_employees;"**);

*2.1 Write a stored procedure to show the tuples in each table:*

A stored procedure provides a shorthand and reusable way to execute a certain block of SQL code. Our desiring result for this task is to show all tuples for each table. Therefore, we have created six procedures corresponding to six tables with SELECT queries inside the procedures.

stmt.executeUpdate(**"create procedure show\_employees() "**+**"begin "**+**"select \* from employees; "**+**"end"**);  
stmt.executeUpdate(**"create procedure show\_customers() "**+**"begin "**+**"select \* from customers; "**+**"end"**);  
stmt.executeUpdate(**"create procedure show\_suppliers() "**+**"begin "**+**"select \* from suppliers; "**+**"end"**);  
stmt.executeUpdate(**"create procedure show\_products() "**+**"begin "**+**"select \* from products; "**+**"end"**);  
stmt.executeUpdate(**"create procedure show\_purchases() "**+**"begin "**+**"select \* from purchases; "**+**"end"**);  
stmt.executeUpdate(**"create procedure show\_logs() "**+**"begin "**+**"select \* from logs; "**+**"end"**);

*2.2 Write a procedure to report the monthly sale information for any given product.*

The monthly sale report contains information from two tables:

(1) products: pid, pname.

(2) purchases: pid, ptime, qty, total\_price.

Therefore, we need to add a join condition on pid to our select query to connect two tables. After that, we should reformat the data and use mathematical operations to calculate the desired results.

stmt.executeUpdate(**"create procedure report\_monthly\_sale(pid varchar(4)) "**+**"begin "**+**"select pname, date\_format(ptime,'%b'),"** +**" year(ptime),sum(qty),sum(total\_price),sum(total\_price)/sum(qty) from purchases pur, products pro "** +**"where pur.pid=pid and pur.pid=pro.pid group by year(ptime), month(ptime); "**+**"end"**);

*2.3 Write procedures to add tuples into the purchases table and the products table.*

An add procedure should contain an insert query.

(1) For the purchases table, we need to provide the information regarding: pur\_no, c\_id, e\_id, p\_id, pur\_qty of the input purchase.

(2) For the products table, we need to provide the information regarding: p\_id, p\_name, qoh, qoh\_theshold, original\_price, discnt\_rate, s\_id of the input product.

For example, the add\_product procedure is demonstrated as follows:

stmt.executeUpdate(**"create procedure add\_product(p\_id varchar(4),p\_name varchar(15),qoh int(5),qoh\_threshold int(5),original\_price decimal(6,2),discnt\_rate decimal(3,2),s\_id varchar(2))"** +  
 **" begin "** +  
 **" insert into products (pid, pname, qoh, qoh\_threshold, original\_price, discnt\_rate, sid)"** +  
 **" values(p\_id,p\_name,qoh,qoh\_threshold,original\_price,discnt\_rate,s\_id);"**+**"end"**);

*2.4 Implement the function of logs table using triggers*

A trigger is a stored program with queries which fire automatically in respond to a specific event. Queries to be executed inside a trigger are wrapped between a “begin” and an “end”. The following three triggers with insert queries are to add a log containing information about any change that has been made to tables “products”, “customers” and “purchases”.

(1) Trigger for updating products

*//trigger for log product*

stmt.execute(**"create trigger trigger\_update\_products after update on products for each row"** +

**" begin"** +

**" insert into logs(who, time, table\_name, operation, key\_value)"** +

**" values(user(), now(), 'products', 'update', new.pid);"** +

**" end"**);

(2) Trigger for updating customer

stmt.execute(**"create trigger trigger\_update\_customer after update on customers for each row"** +

**" begin"** +

**" insert into logs(who, time, table\_name, operation, key\_value)"** +

**" values(user(), now(), 'customers', 'update', new.cid);"** +

**" end"**);

(3) Trigger for inserting purchase

stmt.execute(**" create trigger trigger\_insert\_purchases after insert on purchases for each row"** +

**" begin"** +

**" insert into logs (who, time, table\_name, operation, key\_value)"** +

**" values(user(), now(), 'purchases', 'insert', new.pur);"** +

**"end"**);

*2.5 Make sure the qoh is sufficient*

To handle the case when the quantity of a product in stock is less than the purchase quantity, we will perform an examination by selecting the quantity on hand and making a comparison before the purchase is made. We also create a global variable containing this message so that it can be printed out in Java by simply selecting that variable. After that, we will use “select res, msg” to show the message in an error dialog when connecting to the PHP interface.

**" select qoh from products where pid=p\_id into res;"** +

**" if res<=pur\_qty then"** +

**" set msg = 'Insufficient quantity in stock!';"** +

**" set @db\_msg = 'Insufficient quantity in stock!';"** +

**" set res=0;"**+

**" select res, msg;"** +

**" leave add\_purchase;"** +

**" end if;"** +

*2.6 Update products and customers and when qoh is below the threshold*

According to the following block of code, whenever any purchase is made, the quantity will be subtracted from the quantity on hand. If the quantity on hand is less than the quantity on hand threshold, the quantity on hand will be updated.

**" update products set qoh=qoh-pur\_qty where pid=p\_id;"** +

**" select count(\*) from products where pid=p\_id and qoh<qoh\_threshold into res;"** +

**" if res>0 then"** +

**" set msg = 'The current qoh is: ';"** +

*//" select msg, qoh from products where pid=p\_id into @p;" +*

**" update products set qoh=2\*(qoh+pur\_qty) where pid=p\_id;"** +

**" select qoh/2+pur\_qty from products where pid=p\_id into @qoh\_incr;"** +

**" set msg = 'The qoh is increased by: ';"** +

**" set @db\_msg = concat('The qoh is increased by: ',@qoh\_incr);"** +

*//" select msg, qoh+2\*pur\_qty from products where pid=p\_id into @t;" +*

**" end if;"** +

*2.7 Error handling*

We have discovered three typical errors for the add purchase procedure: any of the input product, the customer and the employee is non-existent. Therefore, we have prepared three block of codes inside the add\_purchase() procedure to handle these errors. If one of these errors occurs, the corresponding error message will be set in a similar way to previous tasks.

*//7.1 detect whether the customer exists*

**" select count(cid) from customers where cid=c\_id into res;"** +

**" if res<1 then"** +

**" set msg = 'Customer not found!';"** +

**" set @db\_msg = 'Customer not found!';"** +

**" select res,msg;"** +

**" leave add\_purchase;"** +

**" end if;"** +

*//7.2 detect whether the product exist*

**" select count(pid) from products where pid=p\_id into res;"** +

**" if res<1 then"** +

**" set msg = 'Product not found!';"** +

**" set @db\_msg = 'Product not found!';"** +

**" select res,msg;"** +

**" leave add\_purchase;"** +

**" end if;"** +

*//7.3 detect whether the employee exist*

**" select count(eid) from employees where eid=e\_id into res;"** +

**" if res<1 then"** +

**" set msg = 'Employee not found!';"** +

**" set @db\_msg = 'Employee not found!';"** +

**" select res,msg;"** +

**" leave add\_purchase;"** +

**" end if;"** +

**3. PHP Interface:**

We have used PHP to create an interative interface. The following diagram illustrates the structure of the model:

* “conn.php” is where the connection between php and our database is established. This file is usually included in other php files whenever a query needs to be executed.
* “home.php” provides the main page of our database management system. It contains list of table names which lead to different pages containing the tables’ data. Moreover, there are also shortcuts which enable common procedures including “add\_purchase”, “add\_product” and “report\_montly\_sale”.
* The pair of two files, “table.php” and “table\_execute.php”, will help to fetch data from the database and create a table listing all data tupples for each procedure.
* “shop.php” and “shop\_execute.php”, materialize “add\_purchase” procedure. “shop.php” helps to present a page with different products and buttons to make purchases while “shop\_execute.php” will take in the purchase information and transfer the query to mysql to get executed. “shop\_execute.php” also handles exception during the execution process.
* Similarly, product.php and product\_execute.php oversee the “add\_product” procedure. The "product.php” file creates seven empty fields where user could enter the new product’s information, which will be used to create the query in “product\_execute.php”.
* The “report\_montly\_sale” procedure will be handled by “report.php” and “report\_execute.php”. “report.php” will create a page listing all the products with the basic information and report button connecting to “report\_execute.php” which will produce a sale report corresponding to that product if its purchase history exists.