CS532 Project 2
Using PL/SQL and JDBC to
Implement the Retail Business Management System

(Due: April 24, 2019)

This project is to use Oracle's PL/SQL and JDBC to implement the RBMS application. Up to two students may work together for this project. Due to time constraint, only a subset of the database tables and a subset of the needed functionalities will be implemented in this project.

1. Preparation (5 points)

Due to the limited time we have for this project, we will use only the following tables:

Employees(eid, ename, telephone#)
Customers(cid, cname, telephone#, visits_made, last_visit_date)
Products(pid, pname, qoh, qoh_threshold, original_price, discnt_rate)
Suppliers(sid, sname, city, telephone#)
Supply(sup#, pid, sid, sdate, quantity)
Purchases(pur#, eid, pid, cid, ptime, qty, total_price)

In addition, the following table is also needed for this project:

Logs(log#, who, otime, table_name, operation, key_value)

Each tuple in the logs table describes who (the login name of a database user) has performed what operation (insert, delete, update) on which table (give the table name) and which tuple (as indicated by the value of the primary key of the tuple) at what time (date and time). Attribute log# is the primary key of this table.

Please use the following SQL DDL statements to create the seven tables required for this project. You can copy and paste these statements in a file, e.g., project2-script.sql, in your bingsuns account and use SQL> start project2-script to create these tables. Some of these tables were already created for Project 1. You can either recreate them (you need drop them first) or just create the new tables. Please also note that the tables are created in certain order such that by the time when a foreign key needs to be created, the corresponding primary key has already been created.

```
create table employees
(eid char(3) primary key,
ename varchar2(15),
telephone# char(12));

create table customers
(cid char(4) primary key,
cname varchar2(15),
telephone# char(12),
visits_made number(4),
last_visit_date date);
```

```
create table products
(pid char(4) primary key,
pname varchar2(15),
qoh number(5),
qoh_threshold number(4),
original_price number(6,2),
discnt_rate number(3,2) check(discnt_rate between 0 and 0.8));

create table purchases
(pur# number(6) primary key,
eid char(3) references employees(eid),
pid char(4) references products(pid),
cid char(4) references customers(cid),
qty number(5),
ptime date,
total_price number(7,2));

create table suppliers
(sid char(2) primary key,
sname varchar2(15) not null unique,
city varchar2(15),
telephone# char(12));

create table supply
(sup# number(4) primary key,
pid char(4) references products(pid),
sid char(2) references suppliers(sid),
sdate date,
quantity number(5));

create table logs
(log# number(5) primary key,
who varchar2(12) not null,
otime date not null,
table_name varchar2(20) not null,
operation varchar2(6) not null,
key_value varchar2(6));
```

You should populate the first six tables with appropriate tuples to test your program.

2. PL/SQL Implementation (55 points)

You need to create a PL/SQL package and other Oracle objects (sequences, triggers, ...) to implement this project. The following requirements and functionalities need to be implemented.

1. (3 points) Values for pur#, sup# and log# need to be generated by appropriate **sequences** automatically when new tuples are inserted into the corresponding tables. For each of these primary key attributes, if its data type is number(n), its values should have n digits. Implement a different sequence for each of these attributes.

2. (6 points) Display the tuples in each table. If you just want to run your package in the SQL*Plus environment, it is enough to create a procedure for each table. As an example, you can implement a procedure, say **show_products**, in your package to display all

products in the products table. You need to implement six procedures, one for each table. However, if you want to get the results to your JDBC interface program, you should create a function and use *ref cursor* (see sample program 3 for the use of *ref cursor*).

3. (4 points) Report the monthly sale information for any given product. For example, you can use a procedure, say **report_monthly_sale(prod_id)**, for this operation. For the given product id, you need to report the product name, the month (the first three letters of the month, e.g., FEB for February), the year (4 digits), the total quantity sold each month, the total dollar amount sold each month, and the average sale price (the total dollar amount divided by the total quantity) of each month. Only need to list the information for those months during which the given product has actual sales.

4. (7 points) Add tuples to all tables with input from the keyboard except the logs table (The logs table will be populated by the application automatically). In this project, you are only required to implement procedures to add tuples into the purchases table and the products table. As an example, you can use a procedure, say **add_purchase(e_id, p_id, c_id, pur_qty)**, in your package to add a tuple in the purchases table, where e_id, p_id, c_id and pur_qty are parameters of the procedure. Note that for this example, the pur# of any newly added purchase should be automatically generated by your sequence. In addition, total_price should be computed based on the data in the database automatically and ptime should be the current date (use sysdate).

5. (10 points) Add a tuple to the logs table automatically whenever any table is modified. To simplify, you are only required to consider the following modifications (events): (1) insert a tuple into the purchases table; (2) update the qoh attribute of the products table; (3) update the visits_made attribute of the customers table, and (4) insert a tuple into the supply table. When a tuple is added to the logs table due to the first event, the table_name should be "purchases", the operation should be "insert" and the key_value should be the pur# of the newly inserted purchase. When a tuple is added to the logs table due to the second event, the table_name should be "products", the operation should be "update" and the key_value should be the pid of the affected product. When a tuple is added to the logs table due to the third event, the table_name should be "customers", the operation should be "update" and the key_value should be the cid of the affected customer. When a tuple is added to the logs table due to the fourth event, the table_name should be "supply", the operation should be "insert" and the key_value should be the sup# of the newly inserted supply. Adding tuples to the logs table should be implemented using **triggers**. You need to implement four triggers for this task, one for each event.

6. (4 points) Before a purchase is made (i.e., before a tuple is added into the purchases table), your program needs to make sure that, for the involved product, the quantity to be purchased is equal to or smaller than the quantity on hand (qoh). Otherwise, an appropriate message should be displayed (e.g., "Insufficient quantity in stock.") and the purchase request should be rejected.

7. (16 points) After adding a tuple to the purchases table, the qoh column of the products table should be modified accordingly, that is, the qoh of the product involved in the purchase

should be reduced by the quantity purchased. If the purchase causes the qoh of the product to be below qoh_threshold, your program should perform the following tasks:

1) print a message saying that the current qoh of the product is below the required threshold and new supply is required;
2) automatically order supply for the product (i.e., add a new tuple to the Supply table): the sup# is generated by a sequence, the pid is the pid of the product involved in the purchase, the sid is the sid of a supplier who has supplied this product before (there should be such information in the current Supply table; if multiple suppliers have supplied this product before, use the smallest sid), the quantity to order is M + qoh + 5, where M is the minimum value for quantity such that M + qoh > qoh_threshold, and use sysdate for sdate;
3) increase qoh of the product by the quantity ordered;
4) print another message showing the new value of the qoh of the product; and
5) the insertion of the new tuple in the purchases table may cause the visits_made of the involved customer to be increased by one and the last_visit_date should also have a new date if the purchase is made on a new date. Use triggers to implement the update of qoh, displaying the messages, and updates of the visits_made and last_visit_date.

8. (5 points) You need to make your package user friendly by designing and displaying appropriate messages for all exceptions. For example, if someone wants to find the purchases of a customer but entered a non-existent customer id, your program should report the problem clearly.

3. Interface (30 points)

Implement an interactive and menu-driven interface (textual menus) using Java and JDBC (see sample programs). Your interface program should utilize as many of your PL/SQL code as possible. Note that messages that are printed by the dbms_output package in your PL/SQL package or triggers are generally not visible when you run your Java/JDBC application. You may need to regenerate these messages in your Java program.

A nice GUI (Graphical User Interface) or Web interface will receive a 5-point bonus.

4. Project Report (10 points)

Your project report should provide the following information:

1. The objective and usage of each procedure and function and every other object you create need to be explained clearly.

2. Your code needs to be well documented with in-line comments.

5. Hand-ins, Demo and Grading

1. Each team needs to submit all **source code**, including both PL/SQL and Java, to blackboard by 11:59 pm, April 24, 2019 (Project 2 submission folder). Make sure to include information about the team members.

2. Each team needs to hand in ONE hard copy of the **Project Report** including entire PL/SQL code (including the package, triggers, and sequences) during the demo on April 26. Only one copy for each team is needed.

3. Each team is required to demonstrate the completed project to the instructor or TA. More instructions on demo will be provided before the demo.

4. The grading will be based on the quality of your code and the documentation and based on how successful of your demo is.

All reports must be typed and printed out.