Homework#4

CS364 Introduction to Database Systems, Spring 2017

Instructor: Jin S. Yoo

Due: April 30, 2017 (Sun)

- This homework has 5 parts.
- This homework description is based on Oracle. You may use other database systems if the system supports all functions needed for this homework. Otherwise, use Oracle.
- Make one file for submission. The file name will be CS364_HW4_YourLastName_YourFirstName.doc (/pdf)
- For SQL questions, show your execution result of the SQL (e.g., screen copy for the proof). Otherwise, you will get 0 point for the question.

Setup

- Download **hw4_schema_data.sql** to your working directory and review the database schema.
- The script file includes SQL statements to create 5 tables and populate some sample data into the tables.
- Execute the script file in your database account.
- Your account will have PRODUCT_T, ORDER_T, ORDERLINE_T, EMPLOYEE_T, and CUSTOMER_T.

Part I. View I (20 points)

1. Create a view (named with **OrderDetail**) which includes the following items: order id, order date, customer id, customer name, product id, product description, product price, ordered quality and the project order price (product price * ordered quantity). When you create the OrderDetail view, use the original column names from the tables except the last column named with "OrderPrice". The figure below shows a part of data which we can see via the view.

ORDERID	ORDERDATE	CUSTOMERID	CUSTOMERNAME	PRODUCTID	PRODUCTDESCRIPTION	PRODUCTSTANDARDPRICE	ORDEREDQUANTITY	ORDERPRICE
1007	27-OCT-10	11	American Euro Lifestyles	1	End Table	175	3	525
1001	21-OCT-10	1	Contemporary Casuals	1	End Table	175	2	350

2. Write a SQL query statement to list the details of all orders (using the OrderDetail view) in the descending order of the order date and the ascending order of customer id in the same order date.

3. Write a SQL query statement to list the number of order items and the total order price per each order using the OrderDetail view. The figure below shows a part of the query result.

Order Id	Number of Order Items	Total Order Price
1003	1	1125
1009	2	3700

4. Write a SQL statement to drop the OrderDetail view.

Part II. Sequence and View II (15 points)

1. Create a sequence object (named with **EmployeeIdSeq**) which starts with 100, is increased by 1. When the sequence value reaches 999, it restarts from 100 again.

2. Insert the following employee records into EMPLOYEE_T table.

<employeeName: Brandon Young, employeeCity: Fort Wayne, employeeState: IN, employeeZipcode: 46805, employeeDateHired: system current date>.

<employeeName: Bailey Whitehill, employeeCity: Fort Wayne, employeeState: IN, employeeZipcode: 46805, employeeDateHired: system current date>.

The employeeId is automatically generated with a rule: YYYY-MM-unique number from EmployeeIdSeq object, where YYYY is a current year (e.g., 2017) and MM is a current month (e.g., 04).

3. Create a view (named with **EmployeeIN**) of employees who work in Indiana (IN). The view includes employeeId, employeeName, EmployeeCity, and EmployeeState. When creating the view, use WITH CHECK OPTION.

4. Insert the following employee record into EMPLOYEE_T table via EmployeeIN view.

<employeeName: Sebastian Andaluz, employeeCity: Fort Wayne, employeeState: IN>.
<employeeName: Sebastian Andaluz, employeeCity: Fort Wayne, employeeState: IN>.

If you have an error for this insertion, write the reason.
Next query EmployeeIN and also query EMPLOYEE_T to make sure the record is newly inserted.

5. Change the employeeState of 'Brandon Young' to 'CA' via EmployeeIN view. What happen?

Part III. Stored Procedure (20 points)

1. Write one SQL query statement to list the order frequency of customers <u>in descending order of the order frequency</u>. The query result includes customer id, customer name, customer postal code and order frequency (i.e., the total number of orders of the customer). The figure below shows a part of the query result.

CUSTOMERID	CUSTOMERNAME	CUSTOMERPOSTALCODE	Order Frequency
1	Contemporary Casuals	32601-2871	3
15	Mountain Scenes	84403-4432	2
12	Battle Creek Furniture	49015-3401	1

2. Create a `TopKCustomer_T` table with no data with the following SQLs.

```
DROP TABLE TopKCustomer_T;

CREATE TABLE TopKCustomer_T
AS
SELECT CUSTOMERID, CUSTOMERNAME, CUSTOMERPOSTALCODE
FROM CUSTOMER_T
WHERE 1=2;

ALTER TABLE TopKCustomer_T
ADD (Crank NUMBER, OrderFrequency Number, RankGenerateDate Date);
```

Check the schema of `TopKCustomer_T` table created. You can use the following statement simply to see the table schema.

```
Describe TopKCustomer_T;
```

3. Write a stored procedure (named **Top*K*Customer**) to generate the top *k* customers based on their order frequency, where an arbitrary integer *k* is given in the execution of the procedure. The execution result of the procedure generates a record of a rank (e.g., 1, 2, 3.), order frequency, customer id, customer name, customer postal code and the rank generate date (i.e., current system date), and inserts the query result into `TopKCustomer_T`.

The SQL query statement in the Top*K*Customer procedure would be similar with your Part II Q1 answer except including the customer rank and the rank generate date.

You can give the rank when you fetch each record in the procedure SQL. If your SQL query includes a proper ORDER BY clause (with desc), the first returned product record would be ranked to 1, the second record be ranked to 2, the third record to 3, and so on. You do not need advanced RANK() or DENSE_RANK() functions.

Hint:

```
BEGIN
   … …

   LOOP …
        … …
        INSERT INTO TopKCustomer_T VALUES (…);
        EXIT WHEN …… -- NEED a condition to exit the loop, e.g.,the number
                        of fetched records is the same with k(top k)
   END LOOP;
   …;
END;
```

Last, execute the procedure with k=2 using the following commend.

```
DELETE FROM TopKCustomer_T;   -- Optional to clean up TopKCustomer_T
                                 table before executing the procedure.
EXECUTE TopKCustomer(2);
```

Clearly show that your procedure is working.

For example, for the validation, query the `TopKCustomer_T` query;

```
SELECT * FROM TopKCustomer_T;
```

Running example 1.

```
DELETE FROM TopKCustomer_T;

EXECUTE TopKCustomer(2);

SELECT CRANK, ORDERFREQUENCY, CUSTOMERID,
   CUSTOMERNAME, CUSTOMERPOSTALCODE, RANKGENERATEDATE
FROM TopKCustomer_T;
```

CRANK	ORDERFREQUENCY	CUSTOMERID	CUSTOMERNAME	CUSTOMERPOSTALCODE	RANKGENERATEDATE
1	3	1	Contemporary Casuals	32601-2871	04-DEC-16
2	2	15	Mountain Scenes	84403-4432	04-DEC-16

Running example 2.

```
DELETE FROM TopKCustomer_T;

EXECUTE TopKCustomer(3);

SELECT CRANK, ORDERFREQUENCY, CUSTOMERID,
   CUSTOMERNAME, CUSTOMERPOSTALCODE, RANKGENERATEDATE
FROM TopKCustomer_T;
```

CRANK	ORDERFREQUENCY	CUSTOMERID	CUSTOMERNAME	CUSTOMERPOSTALCODE	RANKGENERATEDATE
1	3	1	Contemporary Casuals	32601-2871	04-DEC-16
2	2	15	Mountain Scenes	84403-4432	04-DEC-16
3	1	12	Battle Creek Furniture	49015-3401	04-DEC-16
3	1	4	Eastern Furniture	07008-3188	04-DEC-16
3	1	2	Value Furniture	75094-7743	04-DEC-16
3	1	5	Impressions	94206-4056	04-DEC-16
3	1	11	American Euro Lifestyles	07508-5621	04-DEC-16
3	1	8	California Classics	96915-7754	04-DEC-16
3	1	3	Home Furnishings	12209-1125	04-DEC-16

Part IV. Database trigger (20 points)

1. Review the schema of PRODUCT_T table. Additionally, create the **AuditProduct_T** table and **AuditProductPrice_T** table below.

The purpose of AuditProduct_T table is for recording any change in the PRODUCT_T table. The *ChangeStatus* column has only three possible values, 'Insert', 'Update' and 'Delete'. The *ChangeDate* column will have a date (and time) when the change is happened.

```
CREATE TABLE AuditProduct_T
( ProductID     number(11),
  ProductLineId number(11),
  ProductDescription varchar2(50,
  ProductFinish   varchar2(20),
  productStandardPrice number(6,2),
  ChangeDate     date,
  ChangeStatus   varchar2(10) Constraint Status_C
           CHECK (ChangeStatus IN ('Insert','Update','Delete'));
```

The purpose of AuditProductPrice_T table is for recording any change in the product price of PRODUCT_T table. The *productOldPrice* column is for the price before the change and the *productNewPrice* column has the changed price. The *ChangeDate* column will have a date (and time) when the change is happened.

```
CREATE TABLE AuditProductPrice_T
( ProductID     number(11),
  productOldPrice number(6,2),
  productNewPrice number(6,2),
  ChangeDate     date);
```

2. Create a trigger (named with **AduitProductTrig**) for the following purpose. Whenever any change happens in the PRODUCT_T table, the trigger will insert the affected product record with the change date (current date) and change status ('Insert', 'Update' or 'Delete') into the AuditProduct_T table. In the case of a price change of an existing product, you also need to record the price change information into the AuditProductPrice_T table.

You may create one trigger which can handle all three event cases or three different triggers for each event case.
To distinguish each event in one trigger, you may use the following CASE statement in the trigger body:

```
CASE
   WHEN INSERTING THEN
    ….                          ← one insert statement is enough.
   WHEN UPDATING THEN
    ….
   WHEN DELETEING THEN
    ….                          ← one insert statement is enough.
END CASE;
```

In the insert and update cases, you need to use a **:new** prefix tag in Oracle, e.g., :new.ProductID, to access the new/updated product record. To access the deleted product record, you need to use a **:old** tag in Oracle, e.g., :old.ProductID.

3. Show the **AduitProductTrig** trigger you created is working with all three cases; insert, update and delete operations in PRODUCT_T table. You may follow the following steps for the verification.

- Query PRODUCT_T table
- Insert a new product record (e.g., product id = 77) into PRODUCT_T.
- Query PRODUCT_T table and AuditProduct_T table.
- Update the price of the newly inserted product (e.g., product id = 77) .
- Query PRODUCT_T table, AuditProduct_T table, and AuditProductPrice_T table
- Delete the newly inserted product record (e.g., product id = 77)
- Query PRODUCT_T table and AuditProduct_T table.

Part V. JDBC Programming (25 points)

1. **Develop a JDBC program** to list the products (OrderFrequency, ProductID, ProductDescription, and ProductFinish) in order of their popularity among the products of a certain product finish.

For example, when the product finish is given with "Natural Ash",

Order Frequency	Product Id	Description	Finish
4	3	Computer Desk	Natural Ash
2	7	Dining Table	Natural Ash
2	2	Coffee Table	Natural Ash

For example, when the product finish is given with "Cherry",

Order Frequency	Product Id	Description	Finish
2	1	End Table	Cherry
1	5	Writers Desk	Cherry

Clearly show your program is working (e.g., with an execution screen shot). Also summit your program code. (If possible, simply include the Java program code in your homework world/pdf file.)