Assignment 1 – Stored Procedures

Submission

Your submission will consist of two files:

- A single text-based SQL file
  – and –
- An output file demonstrating that your stored procedures work

Reminders

- 1 – Make sure you use proper indentations
- 2 – Make sure you fully comment your stored procedure
- 3 – Make sure you include error handling and test your error handling code

Tasks

1 – (0%) This is preparation – It is very important to ensure you have successfully ingested the data into both tables and have the exact number of rows specified below. Double check this.

- Create the EMPLOYEE table and STAFF table based on DDL in the dbs501-assignment-DDL file
- Take file dbs501-assignment-employee and load this data into an employee table
- Take file dbs501-assignment-staff and load this data into a staff table
- Make sure all records have been successfully loaded or your result sets may be incorrect
- Perform a SELECT COUNT(*) from both tables to ensure there is an exact match with rows

- You should have 35 records in STAFF and 42 records in EMPLOYEE

You can use INSERT, IMPORT, LOAD – whatever you prefer – to ingest the data

2 – (25%) Write a stored procedure called *salary* for the EMPLOYEE table which takes, as input, an employee number and a rating of either 1, 2 or 3.

- The stored procedure should perform the following changes:
- If the employee was rated a 1 – they receive a $10,000 salary increase, additional $300 in bonus and an additional 5% of salary as commission
- If the employee was rated a 2 – they receive a $5,000 salary increase, additional $200 in bonus and an additional 2% of salary as commission
- If the employee was rated a 3 – they receive a $2,000 salary increase with no change to their variable pay
- Use the “old” salary (before increase) to calculate the “new” bonus and/or commission amount
- Make sure you handle two types of errors: (1) A non-existent employee – and – (2) A non-valid rating. Both should have an appropriate message.
- The stored procedure should return the employee number, previous compensation and new compensation (all three compensation components showed separately)
- EMP   OLD SALARY   OLD BONUS   OLD COMM   NEW SALARY   NEW BONUS   NEW COMM
- Demonstrate that your stored procedure works correctly by running it 5 times: Three times with a valid employee number and a 1 rating, 2 rating and 3 rating. Once with an invalid employee number. Once with an invalid rating level.

WHAT TO HAND IN: A copy of your stored procedure and the output of the 5 calls described above.

3– (25%) - Write a stored procedure for the EMPLOYEE table which takes employee number and education level upgrade as input - and - increases the education level of the employee based on the input.
Valid input is:

- “H” (for high school diploma) – and – this will update the edlevel to 16
- “C” (for college diploma) – and – this will update the edlevel to 19
- “U” (for university degree) – and – this will update the edlevel to 20
- “M” (for masters) – and – this will update the edlevel to 23
- “P” (for PhD) – and – this will update the edlevel to 25
- Make sure you handle the error condition of incorrect education level input – and – non-existent employee number
- Also make sure you never reduce the existing education level of the employee. They can only stay the same or go up.
- A message should be provided for all three error cases.
- When no errors occur, the output should look like:
- EMP   OLD EDUCATION   NEW EDUCATION

WHAT TO HAND IN: A copy of your stored procedure and the output of the a set of calls which test all input conditions and error handling. Total of 8 calls and 8 output.

4 – (50%) – For the following use the same tables that you have been using for the labs. Make sure that the data has been DELETEd and re-loaded so you have a fresh copy of the data in each table.

find_customer (customer_id IN NUMBER, found OUT NUMBER);

This procedure has an input parameter to receive the customer ID and an output parameter named found.

This procedure looks for the given customer ID in the database. If the customer exists, it sets the variable *found* to 1. Otherwise, the *found* variable is set to 0.
Make sure to handle the situation when no data was found.

This stored procedure will print out an appropriate message about customer ID being found or not found, using the found variable.

find_product (product_id IN NUMBER, price OUT products.list_price%TYPE);

This procedure has an input parameter to receive the product ID and an output parameter named price.

This procedure looks for the given product ID in the database. If the product exists, it stores the product's list_price in the variable *price*. Otherwise, the *price* variable is set to 0.

Make sure to handle the situation where no data is found.

This stored procedure will print out an appropriate message about the price of the product, using the variables described.

add_order (customer_id IN NUMBER, new_order_id OUT NUMBER)

This procedure has an input parameter to receive the customer ID and an output parameter named new_order_id.

To add a new order for the given customer ID, you need to generate the new order Id. To calculate the new order Id, find the maximum order ID in the orders table and increase it by 1.

This procedure inserts the following values in the orders table:
new_order_id
customer_id (input parameter)
'Shipped' (The value for the order status)
56 (The sales person ID)
sysdate (order date which is the current date)

This stored procedure will print out the information associated with the order being added including the information above and appropriate text summarizing the order information.

add_order_item (orderId IN order_items.order_id%type,
itemId IN order_items.item_id%type,
productId IN order_items.product_id%type,
quantity IN order_items.quantity%type,
price IN order_items.unit_price%type)

This procedure has five IN parameters. It stores the values of these parameters to the table order_items.

This procedure needs to handle errors such as an invalid order ID

display_order (orderId IN NUMBER)

This procedure has an input parameter to receive the order ID and no output parameters.

This procedure will display the order items associated with a particular order ID.

The information to be displayed should include:

- Order ID
- Customer ID

Then should display a row for each item in the order including:

- Item ID
- Product ID
- Quantity
- Price

Then should print a statement showing the total price of the entire order.

There should be an appropriate message for a non-existent order ID

```
master_proc (task IN NUMBER,
             parm1 IN NUMBER)
```

This procedure is a master procedure for four of the five procedures above.

If task = 1, then, call find_customer(parm1)

If task = 2, then, call find_product(parm1)

If task = 3, then, call add_order(parm1)

If task = 4, then, call display-order(parm1)

In all cases, parm1 is the single input parameter required for the specific function.

You need to handle appropriate error messages here as well.

What to include in your SQL file:

In your SQL file, you should include all of your:

CREATE OR REPLACE PROCEDURE commands (5)

CALL commands (14)

What to include in your OUTPUT file:

In your output file, you should include:

1 – find_customer – with a valid customer ID

2 – find_customer – with an invalid customer ID

3 – find_product – with a valid product ID

4 – find_product – with an invalid product ID

5 – add_order – with a valid customer ID

6 – add_order – with an invalid customer ID

7 – add_order_item – should execute successfully 5 times

8 – add_order_item – should execute with an invalid order ID

9 – display_order – with a valid order ID which has at least 5 order items

10 – display_order – with an invalid order ID

For 1 – 6 and 9 – 10 – your call should be to the master_proc procedure. It will call the actual procedure itself.