**\*\*\*\*\*\*\*\*\*\* EXPERIMENT: 07 \*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Problem Statement:**

Using the relation schemata established in Experiments - 02, 03, and 05, create and execute SQL programs for retrieving data using cursors.

**Author : Mehul Y Khandhadiya**

**Roll no : 55**

**Date : 22-Oct-2020**

**Queries Set**

**\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 01 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write SQL code to compile and execute a stored procedure - SHOW\_EMPLOYEE, to list employee details for the input variable ENO holding employee number. (Use EMPP Table)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 02 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write SQL code to compile and execute a stored procedure - ADD\_EMPLOYEE, to add a record to EMPP table. Check the existence of the created procedure using USER\_OBJECTS view. Use this procedure to insert following records.**

**7118, Your Name, 07-Jul-2020, Teaching Asst., 25000**

**7119, Atulya Bharat, 03-Aug-2005, Professor, 162000**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 03 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write SQL code to compile and execute the stored procedure - REMOVE EMPLOYEE, which will remove the employee record(s) from EMPP table when supplied with an input name phrase (entered always as lower case) indicating employee name (use EMPP table). If the matching employee is not found, an appropriate exception should be raised.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 04 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write SQL code to compile and execute the stored function - CHECK\_ITEM that will report status as 1 if items with mentioned P\_CODE are present in the inventory, otherwise reports status as 0. No exceptions to be handled.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 05 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write a SQL code to compile and execute the stored procedure - ADDITEM, that will insert an item in ITEMS table with given particulars - item code, item description, invoice date, quantity of purchase, minimum quantity, item price and supplier code.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 06 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write a SQL code to compile and execute the stored procedure - UPDATE\_ITEM, that will update particulars (quantity and/or cost) for an item in ITEMS table with given particulars - item code, quantity of purchase, and item price.**

**Report an error when the said item (to be updated) does not exist in ITEMS table (the NO\_DATA\_FOUND exception). Use the CHECK\_ITEM function created earlier.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 07 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Modify procedure in Query-06, as UPDATE\_ITEM\_ADD\_WHEN\_NOT\_FOUND such that when the mentioned item is not present in ITEMS, an item is entered into ITEMS with available particulars supplied in the procedure call.**

**The default values for item description, vendor code and minimum quantity as 'NEW ITEM ...', NULL and (quantity / 8) truncated respectively. Use ADD\_ITEM procedure created earlier.**

**You need not catch the NO\_DATA\_FOUND exception.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 08 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write a SQL code to compile and execute the stored procedure - SHOW ITEM that will list the item particulars for an item in ITEMS table when the item code is supplied as input.**

**Report an error when the said item to be updated does not exist in ITEMS. Use the CHECK\_ITEM function created earlier.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 09 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Modify the procedure in Query-08 as SHOW\_ITEM\_TMR\_E which will handle TOO\_MANY\_ROWS exception in SELECT query.**

**In addition to exceptions in Query-06 (NO\_DATA\_FOUND and OTHERS) the TOO\_MANY\_ROWS exception should be caught when a call to the procedure call -EXEC ADD\_ITEM( HH15P', 'NEW ITEM-2' ,150,NULL,25); fetches more than one row in the result set.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 10 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Now extend the procedure in Query-09 as SHOW\_ITEM\_TMR\_HANDL ED to print the rows returned by the SELECT query after catching the appropriate exception.number from console. You should only report the violations.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* VIVA-VOCE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Q1 – What is a cursor? List the steps associated with**

**implementing a cursor.**

**Cursor** is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML operations on Table by User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors.

**Implicit Cursor**

Whenever any DML operations occur in the database, an implicit cursor is created that holds the rows affected, in that particular operation. These cursors cannot be named and, hence they cannot be controlled or referred from another place of the code. We can refer only to the most recent cursor through the cursor attributes.

**Explicit Cursor**

Programmers are allowed to create named context area to execute their DML operations to get more control over it. The explicit cursor should be defined in the declaration section of the PL/SQL block, and it is created for the 'SELECT' statement that needs to be used in the code.

Below are steps that involved in working with explicit cursors.

* **Declaring the cursor**

Declaring the cursor simply means to create one named context area for the 'SELECT' statement that is defined in the declaration part. The name of this context area is same as the cursor name.

* **Opening Cursor**

Opening the cursor will instruct the PL/SQL to allocate the memory for this cursor. It will make the cursor ready to fetch the records.

* **Fetching Data from the Cursor**

In this process, the 'SELECT' statement is executed and the rows fetched is stored in the allocated memory. These are now called as active sets. Fetching data from the cursor is a record-level activity that means we can access the data in a record-by-record way.

Each fetch statement will fetch one active set and holds the information of that particular record. This statement is same as 'SELECT' statement that fetches the record and assigns to the variable in the 'INTO' clause, but it will not throw any exceptions.

* **Closing the Cursor**

Once all the record is fetched now, we need to close the cursor so that the memory allocated to this context area will be released.

**Q2 – What is an active set?**

The set of rows returned by a SQL query is called the result set. This result set is called **Active Data Set** because data in cursor is ready to undergo any kind of processing**.** The size of cursor is the same as the size required by the number of rows in **Active Data Set.**

**Q3 – What is a cursor for loop? Why it is advantageous?**

The cursor FOR LOOP statement is an elegant extension of the numeric FOR LOOP statement.

The numeric FOR LOOP executes the body of a loop once for every integer value in a specified range.  Similarly, the cursor FOR LOOP executes the body of the loop once for each row returned by the query associated with the cursor.

A nice feature of the cursor FOR LOOP statement is that it allows you to fetch every row from a cursor without manually managing the execution cycle i.e.,  OPEN, FETCH, and CLOSE.

The cursor FOR LOOP implicitly creates its loop index as a record variable with the row type in which the cursor returns and then opens the cursor.

In each loop iteration, the cursor FOR LOOP statement fetches a row from the result set into its loop index. If there is no row to fetch, the cursor FOR LOOP closes the cursor.

The cursor is also closed if a statement inside the loop transfers control outside the loop, e.g., EXIT and GOTO, or raises an exception.

**Q4 - Why it is a good practice to close a cursor?**

After we are done with our working with server we should always close the cursor because closing the cursor means releasing the allocated memory. Hence the memory occupied by the cursor will be releases as soon as we close it.

Syntax - CLOSE c\_customers;

**INFERENCE:** From this expt., we learnt about cursors in PL/SQL. We learnt about its uses, its advantages, attributes etc. After that we learnt about its implementation how to declare a cursor how to use it and etc. We implemented using some queries and the results were recorded and analyzed.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***