**ASSIGNMENT – 4**

**Q1. Is it possible to insert, update and delete within one select statement? If so, how can we do that? Provide an example for your answer.** *– 10 Points*

* MERGE statement can be used to implement insert, update, delete within one select statement by combining the three operations into a single table. Merge statement makes use of two tables, source table and target table, where target table is modified based on the data present in the source table.

**Syntax:**

MERGE INTO targetTable USING sourceTable ON mergeCondition

When MATCHED Then updateStatement

When NOT MATCHED BY TARGET Then insertStatement

When NOT MATCHED BY SOURCE Then deleteStatement

**Example:**

1. Table Creation:

CREATE TABLE EMPLOYEES (EmployeeID INT PRIMARY KEY, EmployeeName VARCHAR (25), EmployeeSalary INT);

INSERT INTO EMPLOYEES (EmployeeID, EmployeeName, EmployeeSalary) VALUES(1,'EmpA',5000);

INSERT INTO EMPLOYEES (EmployeeID, EmployeeName, EmployeeSalary) VALUES(2,'EmpB',6000);

INSERT INTO EMPLOYEES (EmployeeID, EmployeeName, EmployeeSalary) VALUES(3,'EmpF',7000);

INSERT INTO EMPLOYEES (EmployeeID, EmployeeName, EmployeeSalary) VALUES(4,'EmpG',8000);

INSERT INTO EMPLOYEES (EmployeeID, EmployeeName, EmployeeSalary) VALUES(5,'EmpH',9000);

Graphical user interface, text, application

Description automatically generated

CREATE TABLE UPDATED\_EMPLOYEES (EmployeeID INT PRIMARY KEY, EmployeeName VARCHAR (25), EmployeeSalary INT);

INSERT INTO UPDATED\_EMPLOYEES (EmployeeID, EmployeeName, EmployeeSalary) VALUES(1,'EmpA',5000);

INSERT INTO UPDATED\_EMPLOYEES (EmployeeID, EmployeeName, EmployeeSalary) VALUES(2,'EmpB',6000);

INSERT INTO UPDATED\_EMPLOYEES (EmployeeID, EmployeeName, EmployeeSalary) VALUES(6,'EmpF',7500);

INSERT INTO UPDATED\_EMPLOYEES (EmployeeID, EmployeeName, EmployeeSalary) VALUES(7,'EmpG',8500);

INSERT INTO UPDATED\_EMPLOYEES (EmployeeID, EmployeeName, EmployeeSalary) VALUES(8,'EmpH',9500);

Graphical user interface, text, application

Description automatically generated

1. MERGING TABLES AND USING COMMANDS

MERGE INTO EMPLOYEES TARGET

USING UPDATED\_EMPLOYEES SOURCE

ON (TARGET.EmployeeID = SOURCE.EmployeeID);

* UPDATE OPERATION
* WHEN MATCHED THEN UPDATE

SET TARGET.EmployeeName = SOURCE.EmployeeName,

TARGET TARGET.EmployeeSalary = SOURCE.EmployeeSalary;

* INSERT OPERATION
* WHEN NOT MATCHED BY TARGET

THEN INSERT (EmployeeID,EmployeeName,EmployeeSalary)

VALUES(SOURCE.EmployeeID,Source.EmployeeName,SOURCE.EmployeeSalary);

* DELETE OPERATION
  + WHEN NOT MATCHED BY SOURCE

THEN DELETE;

**Q2. What is the difference between LEAD and LAG functions? Explain different types of arguments that we can pass to these functions in detail. Using your example tables, show the usage of these functions.** *– 10 Points*

* LEAD and LAG are positional functions that can be used as a reference for data present above or below the current row in a table. They can be useful in creating reports.
* While LAG function provides access to data stored in rows above the current row, LEAD function provides access to data stored in rows below the current row.
* Both the LEAD and LAG functions take three arguments. While LAG() consists of arguments such as: name of column, attribute or expression from where the value is being fetched, the offset value which is the number of rows to skip above the current row and default value to be returned in case the value of the above row is empty.
* LEAD() consists of arguments: name of column or attribute from where the value is being fetched, the offset value which is the number of rows to skip below the current row and default value to be returned in case the value of the above row is empty.
* Only the first argument (name of the column or attribute) is a required argument that is needed in both the functions while other arguments (offset value and default value) are set to 1 and NULL respectively.

**Syntax:**

* LEAD (fetchedValue,offsetValue,default) OVER (PARTITION BY partitionExpression…) ORDER BY column ASC|DESC;
* LAG (fetchedValue,offsetValue,default) OVER (PARTITION BY partitionExpression…) ORDER BY column ASC|DESC;

**Example (DEFAULT):**

* SELECT EmployeeName, EmployeeSalary, LEAD(EmployeeSalary) OVER (ORDER BY EmployeeSalary) as next\_salary from EMPLOYEES;

Text

Description automatically generated with medium confidence

* SELECT EmployeeName, EmployeeSalary, LAG(EmployeeSalary) OVER (ORDER BY EmployeeSalary) as previous\_salary from EMPLOYEES;

Text

Description automatically generated with medium confidence

**Example (WITH CUSTOM ARGUMENTS):**

* SELECT EmployeeName, EmployeeSalary, LEAD(EmployeeSalary,2,0) OVER (ORDER BY EmployeeSalary) as next\_salary from EMPLOYEES;

A picture containing text

Description automatically generated

* SELECT EmployeeName, EmployeeSalary, LAG(EmployeeSalary,2,0) OVER (ORDER BY EmployeeSalary) as previous\_salary from EMPLOYEES;

A picture containing text

Description automatically generated

**Q3. Explain the below terms with an example–**

**a) PL/SQL Anonymous Block** *-****4 points***

**b) Cursor** -***4 points***

**c) Pl/SQL parameter modes (IN, OUT, INOUT)** -***4 points***

**d) Instead of triggers** -***4 points***

**e) AUTONOMOUS\_TRANSACTION pragma** -***4 points***

**- a) PL/SQL Anonymous blocks are Procedural Language SQL blocks or block-structured language that have not been given any specific names. They are usually used for testing purposes and are for one-time use only since PL/SQL anonymous blocks are not stored in Oracle Database servers.**

**Eg: Begin**

**DBMS\_OUTPUT.put\_line(‘Hello World!’);**

**End;**

**b) Cursor is a specialized construct of Procedural Language SQL that acts as a pointer and is used to hold rows returned by the SQL statements or queries. There are two types of cursors in PL/SQL:**

* Implicit Cursors: These cursors are automatically created by Oracle when a SQL query is implemented, and when explicit cursors do not exist. The cursors and the data that it retains cannot be controlled by the users.
* Explicit Cursors: Unlike Implicit cursors, Explicit cursors are defined by programmers and thus the cursor and its data can be controlled by programmers. It is created to hold multiple rows that are returned by a SQL statement or query. Thus, Explicit cursors should be defined in the declaration block and is created on select statement returning multiple rows.

c) There are three parameter modes in PL/SQL:

* IN Parameter: IN formal parameters are initialized to the actual parameters with which it was called. It cannot be modified with a new value by the called program, although it can still be referenced. The actual parameters retain its value prior to the program call.
* OUT Parameter: OUT formal parameters are initialized to the actual parameters with which it was called. Unlike IN parameters, the called program can modify and reference the value of OUT parameters. In the case of a handled exception’s occurrence, the actual parameters set the last value to which the formal parameters were set, while in the case of an unhandled exception’s occurrence, the actual parameters retain its value prior to the program call.
* IN-OUT Parameter: Like IN Parameter, IN-OUT formal Parameters can be initialized to the actual parameters with which it was called. While like OUT Parameter, it can be modified with a new value by the called program. In the case of a handled exception’s occurrence, the actual parameters set the last value to which the formal parameters were set, while in the case of an unhandled exception’s occurrence, the actual parameters retain its value prior to the program call.

d) INSTEAD OF Triggers are a type of trigger that implements a view to update the data in a table which is otherwise unmodifiable with the help of DML statements. It skips the DML statements automatically.

Eg: CREATE [OR REPLACE] TRIGGER customTrigger INSTEAD OF {INSERT|UPDATE|DELETE} ON customView;

(where customerTrigger and customView are trigger and view names respectively).

e) Autonomous transactions pragma is a compiler directive of autonomous transactions that allows users to implement an independent transactions as well as implement commit, rollback on those operations without affecting the commit, rollbacks or state of the calling transaction.

Q4. Create procedure with appropriate arguments to perform updates and inserts on department table (Department name will be unique). Make sure to upload script execution test cases for all the combinations to prove the validation is successfully working and upload screenshots for each question proving the test cases.

A. CREATE DEPT TABLE AND INSERT 6 RECORDS INTO IT (refer to Instructions for Schema) *-****6 points***

B. INSERT THE DEPARTMENT IF NAME DOESN'T EXISTS *-****6 points***

C. UPDATE THE DEPARTMENT LOCATION IF NAME EXISTS *-****6 points***

D. RAISE ERROR IF THE DEPARTMENT NAME IS INVALID (NULL, ZERO LENGTH) *-****6 points***

E. RAISE ERROR IF THE DEPARTMENT NAME IS A NUMBER *-****6 points***

F. ACCEPTED LOCATIONS SHOULD BE AS BELOW *-****6 points***

                MA, TX, IL, CA, NY, NJ, NH, RH

G. DEPARTMENT ID SHOULD BE AUTO-GENERATED *-****6 points***

H. LENGTH OF THE DEPARTMENT NAME CANNOT BE MORE THAN 20 CHARS *-****6 points***

I. WHILE INSERTING THE DEPARTMENT NAME CONVERT EVERYTHING TO CAMEL CASE *-****6 points***

J. MAKE SURE DEPARTMENT NAME IS UNIQUE *-****6 points***

**Ans:**

**CODE:**

**---------------------------------------------------------------------------------------------------------------------**

**CREATE TABLE DEPARTMENT(**

**dept\_id number(5) NOT NULL PRIMARY KEY,**

**dept\_name varchar(40) NOT NULL,**

**dept\_location varchar(40) NOT NULL);**

**INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (101,'Finance','MA');**

**INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (102,'Marketing','TX');**

**INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (103,'Human Resource','IL');**

**INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (104,'Security','CA');**

**INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (105,'Sales','NY');**

**INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (106,'Operations','NH');**

**select \* from department;**

**CREATE OR REPLACE PROCEDURE dept\_proc(departmentNames IN DEPARTMENT.dept\_name%TYPE,departmentLocations IN DEPARTMENT.dept\_location%TYPE)**

**IS**

**countUnique Number;**

**location\_not\_found EXCEPTION;**

**dept\_Not\_Unique EXCEPTION;**

**dept\_name\_invalid EXCEPTION;**

**dept\_cannot\_be\_number EXCEPTION;**

**INVALID\_ERROR EXCEPTION;**

**v\_loc varchar(5);**

**BEGIN**

**IF (length(departmentNames) = 0) THEN**

**RAISE INVALID\_ERROR;**

**ELSIf departmentNames is null or length(departmentNames)>20 then**

**raise dept\_name\_invalid;**

**ELSIF**

**isNumberFunc(departmentNames)=1**

**then raise dept\_cannot\_be\_number;**

**ELSIF**

**departmentLocations NOT IN('MA', 'TX', 'IL', 'CA', 'NY', 'NJ', 'NH', 'RH')**

**then raise location\_not\_found;**

**ELSIF (v\_loc=departmentLocations) THEN**

**update department set dept\_location=departmentLocations where dept\_name = departmentNames;**

**ELSE**

**merge into department dept**

**using (select departmentNames Name,departmentLocations Location from dual) dept\_Names**

**on (initcap (dept.dept\_Name) = initcap(dept\_Names.Name))**

**WHEN MATCHED THEN UPDATE set dept.dept\_Location =dept\_Names.Location where b.Location IN('MA', 'TX', 'IL', 'CA', 'NY', 'NJ', 'NH', 'RH')**

**WHEN NOT MATCHED THEN INSERT (dept\_id,dept\_name,dept\_location ) VALUES (seq\_id.nextval,INITCAP(departmentNames),departmentLocations);**

**End if;**

**EXCEPTION**

**WHEN dept\_name\_invalid THEN**

**raise\_application\_error (-20001,'deptname greater then 20 or it is null.');**

**WHEN dept\_cannot\_be\_number THEN**

**raise\_application\_error (-20002,'deptname cannot be number');**

**WHEN location\_not\_found THEN**

**raise\_application\_error (-20003,'location not found in list');**

**WHEN dept\_Not\_Unique THEN**

**raise\_application\_error (-20004,'Deptname not unique');**

**WHEN INVALID\_ERROR THEN**

**raise\_application\_error (-20005,'Deptname length cannot be zero');**

**commit;**

**END;**

**/**

**--INPUTS**

**exec dept\_proc(‘Education dept','TX'); -- (B)**

**exec dept\_proc(' Finance','TX'); -- (C)**

**exec dept\_proc('NULL','TX'); -- (D) [ERROR]**

**exec dept\_proc(55,'CA'); -- (E) [ERROR]**

**exec dept\_proc('Health','RH'); -- (F)**

**exec dept\_proc(‘Health’,'AZ'); --(F) [ERROR]**

**exec dept\_proc('Science&WelfareDepartment','MA'); -- (H)**

**exec dept\_proc('maths dept,'CA'); -- (I)**

**select \* from department;**

**CREATE OR REPLACE FUNCTION isNumberFunc (s\_string IN VARCHAR2)**

**RETURN INT**

**IS**

**v\_num NUMBER;**

**BEGIN**

**v\_num := TO\_NUMBER(s\_string);**

**RETURN 1;**

**EXCEPTION**

**WHEN VALUE\_ERROR THEN**

**RETURN 0;**

**END isNumberFunc;**

**CREATE SEQUENCE sequence\_id**

**MINVALUE 1**

**START WITH 1**

**INCREMENT BY 1**

**CACHE 10;**

**---------------------------------------------------------------------------------------------------------------------**

1. CREATE TABLE DEPARTMENT(

dept\_id number(5) NOT NULL PRIMARY KEY,

dept\_name varchar(40) NOT NULL,

dept\_location varchar(40) NOT NULL);

INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (101,'Finance','MA');

INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (102,'Marketing','TX');

INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (103,'Human Resource','IL');

INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (104,'Security','CA');

INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (105,'Sales','NY');

INSERT INTO DEPARTMENT (dept\_id,dept\_name,dept\_location) values (106,'Operations','NH');

Graphical user interface, application

Description automatically generated

1. Education department didn’t exist, so it was inserted into the table

Graphical user interface, text, application, email

Description automatically generated

1. Finance department location changed from MA to TX since it already existed in table

Graphical user interface, application

Description automatically generated

1. Error raised since department name cannot be null

Graphical user interface, application

Description automatically generated

1. Error raised since department name cannot be a number

Graphical user interface, application, Word

Description automatically generated

1. Entry of Health department is accepted since RH location is in list of locations accepted by system

Graphical user interface, application, Word

Description automatically generated

Whereas error is raised when location of Health department is changed since ‘AZ’ location is not in the list

Graphical user interface, application, Word

Description automatically generated

1. Education department has its id auto generated once it is inserted into the table

Graphical user interface, application

Description automatically generated

1. Since length of department name cannot be more than 20 characters, it is substituted by variable Science12 as prompted to change the dept name to the user

Graphical user interface, application, table

Description automatically generated

1. Maths department converted to camel case upon entry into table

Graphical user interface, table

Description automatically generated

1. Department names in table are unique, upon adding another department with same name and location, no changes are made.

Graphical user interface, table

Description automatically generated