```
MariaDB [test]> select * from teacher;
+-------+-----------+------+---------+-----------------+--------+
| tr_id | tr_name   | dept | subject | class_in_charge | salary |
+-------+-----------+------+---------+-----------------+--------+
|   101 | DEEPA     |    1 | CST104  | CSB             | 103000 |
|   106 | PARVATHY  |    1 | CSL101  | NULL            |  85000 |
|   128 | JOHNY     |    2 | MET203  | MEA             | 120000 |
|   217 | VARGHESE  |    2 | MEL106  | NULL            |  90000 |
|   304 | RESMI     |    3 | CVT405  | CVA             | 120000 |
+-------+-----------+------+---------+-----------------+--------+
5 rows in set (0.001 sec)

MariaDB [test]> select tr_id,tr_name from teacher where salary=120000
    -> UNION
    -> select tr_id,tr_name from teacher where dept=2;
+-------+-----------+
| tr_id | tr_name   |
+-------+-----------+
|   128 | JOHNY     |
|   304 | RESMI     |
|   217 | VARGHESE  |
+-------+-----------+
3 rows in set (0.002 sec)

MariaDB [test]> select tr_id,tr_name from teacher where salary=120000
    -> UNION ALL
    -> select tr_id,tr_name from teacher where dept=2;
+-------+-----------+
| tr_id | tr_name   |
+-------+-----------+
|   128 | JOHNY     |
|   304 | RESMI     |
|   128 | JOHNY     |
|   217 | VARGHESE  |
+-------+-----------+
4 rows in set (0.002 sec)

MariaDB [test]> select tr_id,tr_name from teacher where salary=120000
    -> INTERSECT
    -> select tr_id,tr_name from teacher where dept=2;
+-------+---------+
| tr_id | tr_name |
+-------+---------+
```


Nested Queries

A subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause. Subqueries can be used with SELECT, INSERT, UPDATE and DELETE statement along with operators like =, <, IN, BETWEEN etc

Syntax : Nested queries with the SELECT statement

SELECT column_name [, column_name] FROM table1 [, table2]
WHERE column_name OPERATOR [SELECT column_name [, column_name]
FROM table1 [, table2] [WHERE]) ;

Join Queries:

1. Inner Join : returns records with matching values in both tables
   SYNTAX : SELECT column_name(s) FROM table1 INNER JOIN table2
   ON table1.column_name = table2.column_name ;
2. Left Outer Join : returns all records from left table and the matched records from the right table
   SYNTAX : SELECT column_name(s) FROM table1 LEFT JOIN table2
   ON table1.column_name = table2.column_name.
3. Right Outer Join : returns all records from the right table and the matched records from the left table
   SYNTAX : SELECT column_name(s) FROM table1 RIGHT JOIN table2
   ON table1.column_name = table2.column_name ;
4. Full Outer Join : returns all records when there is a match in either left or right table
   SYNTAX : SELECT column_name(s) FROM table1 FULL OUTER JOIN table2 ON
   table1.column_name = table2.column_name WHERE condition;

RESULT : The queries were successfully executed and desired outputs were obtained.

```
MariaDB [company]> CREATE TEMPORARY TABLE TempEmployee (emp_id int, emp_name varchar (30), deptno int);
Query OK, 0 rows affected (0.002 sec)

MariaDB [company]> INSERT INTO TempEmployee (emp_id, emp_name, deptno) SELECT Ssn, Name, Dno FROM employee;
Query OK, 6 rows affected (0.001 sec)
Records: 6  Duplicates: 0  Warnings: 0

MariaDB [company]> SELECT * FROM TempEmployee WHERE deptno = 204;
+--------+----------+--------+
| emp_id | emp_name | deptno |
+--------+----------+--------+
|    100 | Joey     |    204 |
+--------+----------+--------+
1 row in set (0.001 sec)
```

01/01/22

EXPERIMENT NO. 11

TEMP TABLES

AIM:

Implementation of queries using temp tables.

THEORY

Temporary table allows to store temporary data

Syntax:

CREATE TEMPORARY TABLE table-name (column1, column2, ..., table constraints);

To create a temporary table whose structure is the same as an existing table

```
CREATE TEMPORARY TABLE temporary_table_name
SELECT * FROM original_table_name Limit 0;
```

RESULT

Successfully implemented queries using temp tables.

```
MariaDB [mec]> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.008 sec)

MariaDB [mec]> SELECT * FROM MARK;
+--------+--------+------+-------+
| ROLLNO | NAME   | MARK | GRADE |
+--------+--------+------+-------+
|      1 | aman   |    9 | A     |
|      2 | varma  |   10 | A+    |
|      3 | gopika |    8 | B+    |
|      4 | kumar  |    7 | B     |
+--------+--------+------+-------+
4 rows in set (0.000 sec)

MariaDB [mec]> COMMIT;
Query OK, 0 rows affected (0.000 sec)

MariaDB [mec]> UPDATE MARK SET NAME="pavan"
    ->          WHERE ROLLNO=1;
Query OK, 1 row affected (0.001 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [mec]> SELECT * FROM MARK;
+--------+--------+------+-------+
| ROLLNO | NAME   | MARK | GRADE |
+--------+--------+------+-------+
|      1 | pavan  |    9 | A     |
|      2 | varma  |   10 | A+    |
|      3 | gopika |    8 | B+    |
|      4 | kumar  |    7 | B     |
+--------+--------+------+-------+
4 rows in set (0.001 sec)

MariaDB [mec]> ROLLBACK;
Query OK, 0 rows affected (0.002 sec)

MariaDB [mec]> SELECT * FROM MARK;
+--------+--------+------+-------+
| ROLLNO | NAME   | MARK | GRADE |
+--------+--------+------+-------+
|      1 | aman   |    9 | A     |
|      2 | varma  |   10 | A+    |
|      3 | gopika |    8 | B+    |
|      4 | kumar  |    7 | B     |
+--------+--------+------+-------+
4 rows in set (0.000 sec)
```

10/01/22

EXPERIMENT NO. 12

SQL TCL COMMANDS

AIM:

Practice of SQL TCL Commands like Rollback, Commit, Savepoint.

THEORY:

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program. The following commands are used to control transactions:

1. **COMMIT** : This is the transactional command used to save changes invoked by a transaction to the database. This command saves all the transactions to the database since the last COMMIT or ROLLBACK command.
   SYNTAX: COMMIT;

2. **ROLLBACK** : This command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.
   SYNTAX: ROLLBACK;

```
MariaDB [mec]> SAVEPOINT P1;
Query OK, 0 rows affected (0.001 sec)

MariaDB [mec]> INSERT INTO MARK VALUES(5,"manu",8,"B+");
Query OK, 1 row affected (0.001 sec)

MariaDB [mec]> SAVEPOINT P2;
Query OK, 0 rows affected (0.000 sec)

MariaDB [mec]> UPDATE MARK SET MARK=5 WHERE ROLLNO=2;
Query OK, 1 row affected (0.001 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [mec]> SELECT * FROM MARK;
+--------+--------+------+-------+
| ROLLNO | NAME   | MARK | GRADE |
+--------+--------+------+-------+
|      1 | aman   |    9 | A     |
|      2 | varma  |    5 | A+    |
|      3 | gopika |    8 | B+    |
|      4 | kumar  |    7 | B     |
|      5 | manu   |    8 | B+    |
+--------+--------+------+-------+
5 rows in set (0.000 sec)

MariaDB [mec]> ROLLBACK TO P2;
Query OK, 0 rows affected (0.000 sec)

MariaDB [mec]> SELECT * FROM MARK;
+--------+--------+------+-------+
| ROLLNO | NAME   | MARK | GRADE |
+--------+--------+------+-------+
|      1 | aman   |    9 | A     |
|      2 | varma  |   10 | A+    |
|      3 | gopika |    8 | B+    |
|      4 | kumar  |    7 | B     |
|      5 | manu   |    8 | B+    |
+--------+--------+------+-------+
5 rows in set (0.000 sec)

MariaDB [mec]> ROLLBACK TO P1;
Query OK, 0 rows affected (0.001 sec)

MariaDB [mec]> SELECT * FROM MARK;
+--------+--------+------+-------+
| ROLLNO | NAME   | MARK | GRADE |
+--------+--------+------+-------+
|      1 | aman   |    9 | A     |
|      2 | varma  |   10 | A+    |
|      3 | gopika |    8 | B+    |
|      4 | kumar  |    7 | B     |
+--------+--------+------+-------+
4 rows in set (0.000 sec)
```


3. SAVEPOINT : It is a point in a transaction when we can roll the transaction back to a certain point without rolling back the entire transaction.

SYNTAX : SAVEPOINT SAVEPOINT_NAME ;

The ROLLBACK command is used to undo a group of transaction.

SYNTAX: ROLLBACK TO SAVEPOINT_NAME ;

4. RELEASE SAVEPOINT : It is used to remove the savepoint created.

SYNTAX: RELEASE SAVEPOINT SAVEPOINT_NAME ;

Once the SAVEPOINT has been released, ROLLBACK command will not undo the transactions performed since the last SAVEPOINT

5. SET TRANSACTION : It is used to initiate a database transaction. This command is used to specify characteristics for the transaction that follows.

SYNTAX:

SET TRANSACTION [READ WRITE | READ ONLY] ;

RESULT :

The SQL TCL commands were successfully executed.

```
MariaDB [mec]> CREATE USER 'EMP1' IDENTIFIED BY '123;'
    -> ;
Query OK, 0 rows affected (0.024 sec)
MariaDB [mec]> SELECT * FROM MARK;
+--------+--------+------+-------+
| ROLLNO | NAME   | MARK | GRADE |
+--------+--------+------+-------+
|      1 | aman   |    9 | A     |
|      2 | varma  |   10 | A+    |
|      3 | gopika |    8 | B+    |
|      4 | kumar  |    7 | B     |
+--------+--------+------+-------+
4 rows in set (0.000 sec)

MariaDB [mec]> GRANT SELECT,INSERT ON MARK TO EMP1;
Query OK, 0 rows affected (0.005 sec)

MariaDB [mec]> REVOKE INSERT ON MARK TO EMP1;
ERROR 1064 (42000): You have an error in your SQL syntax.
t line 1
MariaDB [mec]> REVOKE INSERT ON MARK FROM EMP1;
Query OK, 0 rows affected (0.002 sec)
```

mysql -u root -p.

grant grant option on *.* to 'mec'@'localhost';

flush privileges; exit.

mysql -u mec -p.

13/01/22

27

EXPERIMENT NO.13

SQL DCL COMMANDS

AIM:

To practice SQL DCL commands for granting and revoking user privileges.

THEORY:

DCL COMMANDS: Data control language (DCL) is used to access the stored data. It is mainly used to revoke/grant access to database.

1) Grant command.

It is employed to grant a privilege to a user. It allows specified users to perform specified tasks.

SYNTAX: GRANT privilege_name on objectname to user;

Privilege names are SELECT, UPDATE, DELETE, INSERT, ALTER, ALL. Object name is table name and user is the one who is granted the privileges.

2) REVOKE Command

It is employed to revoke a privilege from a user. It helps the owner to cancel previously granted permissions.

SYNTAX: REVOKE privilege_name on objectname from User.

Privilege names are SELECT, UPDATE, DELETE, INSERT, ALTER, ALL. Object name is table name. User is the one whose privileges are removed.

RESULT:

SQL DCL commands were successfully executed.

```
MariaDB [test]> select * from teacher;
+-------+----------+------+---------+-----------------+---------+
| tr_id | tr_name  | dept | subject | class_in_charge | salary  |
+-------+----------+------+---------+-----------------+---------+
|   101 | DEEPA    |    1 | CST104  | CSB             |  103000 |
|   106 | PARVATHY |    1 | CSL101  | NULL            |   85000 |
|   128 | JOHNY    |    2 | MET203  | MEA             |  120000 |
|   217 | VARGHESE |    2 | MEL106  | NULL            |   90000 |
|   304 | RESMI    |    3 | CVT405  | CVA             |  120000 |
+-------+----------+------+---------+-----------------+---------+
5 rows in set (0.003 sec)

MariaDB [test]> create view block
    ->          as select tr_id,tr_name
    ->          from teacher;
Query OK, 0 rows affected (0.015 sec)

MariaDB [test]> select * from block;
+-------+----------+
| tr_id | tr_name  |
+-------+----------+
|   101 | DEEPA    |
|   106 | PARVATHY |
|   128 | JOHNY    |
|   217 | VARGHESE |
|   304 | RESMI    |
+-------+----------+
5 rows in set (0.014 sec)

MariaDB [test]> drop view block;
Query OK, 0 rows affected (0.007 sec)
```

17/01/22

EXPERIMENT NO. 14

VIEWS AND ASSERTIONS

AIM:

To practice SQL commands for creation of views and assertions.

THEORY:

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

CREATING VIEWS

Database views are created using CREATE VIEW statement. Views can be created from a single table, multiple tables or another view.

SYNTAX: CREATE VIEW view_name AS SELECT column1, column2... FROM table_name WHERE [condition];

DELETING VIEW

A view can be deleted using Drop View statement.

SYNTAX: DROP VIEW view_name;

SQL ASSERTION:

When a constraint involves 2 (or) more tables, the table constraint mechanism is sometimes hard and results may not be as expected. To cover such a situation SQL supports the creation of assertions that are constraints not associated with only one table. And an assertion statement should ensure a certain condition to exist always in database

```sql
CREATE OR REPLACE PACKAGE assert
IS
PROCEDURE this_condition (
    condition_in IN BOOLEAN
   , msg_in IN VARCHAR2
   , display_call_stack_in IN BOOLEAN DEFAULT FALSE
   , null_means_failure_in IN BOOLEAN DEFAULT TRUE
);
PROCEDURE is_in_range (
      date_in IN DATE
    , low_date_in IN DATE
    , high_cate_in IN DATE
    , msg_in IN VARCHAR2
    , display_call_stack_in IN BOOLEAN DEFAULT FALSE
   );
END assert;
```

Package created

```sql
CREATE OR REPLACE PACKAGE BODY assert
IS
   PROCEDURE this_condition (
      condition_in IN BOOLEAN
    , msg_in IN VARCHAR2
    , display_call_stack_in IN BOOLEAN DEFAULT FALSE
    , null_means_failure_in IN BOOLEAN DEFAULT TRUE
   )
   IS
   BEGIN
      IF NOT condition_in
         OR (null_means_failure_in AND condition_in IS NULL)
      THEN
         IF display_call_stack_in
         THEN
            DBMS_OUTPUT.put_line ('ASSERTION VIOLATION! ' || msg_in);
            DBMS_OUTPUT.put_line ('Path taken to assertion violation:');
            DBMS_OUTPUT.put_line (DBMS_UTILITY.format_call_stack);
         END IF;

         raise_application_error (-20000, 'ASSERTION VIOLATION! ' || msg_in);
      END IF;
   END;

    PROCEDURE is_in_range (
       date_in IN DATE
     , low_date_in IN DATE
     , high_date_in IN DATE
     , msg_in IN VARCHAR2
     , display_call_stack_in IN BOOLEAN DEFAULT FALSE
    )
    IS
    BEGIN
       this_condition (TRUNC (date_in) BETWEEN TRUNC (low_date_in)
                                           AND TRUNC (high_date_in)
                 , msg_in
                 , display_call_stack_in
                  );
    END is_in_range;
END assert;
```

Package Body created.

```sql
BEGIN
      assert.is_in_range (
         1,
         ADD_MONTHS (SYSDATE+1, -60),
         SYSDATE+1,
         'date in is out of range'
      );
END;
```

```
ORA-20000: ASSERTION VIOLATION! date_in is out of range ORA-06512: at "SQL_[illegible].ASSERT", line 24
ORA-06512: at "SQL_[illegible].ASSERT", line 61
ORA-06512: at "SQL_[illegible].CALC_TOTALS", line 11
ORA-06512: at "SQL_[illegible].CALC_TOTALS", line 19
ORA-06512: at line 2
ORA-06512: at "SYS.DBMS_SQL", line 1721
```


DBMS always checks the assertions whenever modifications are done in the corresponding table.

SYNTAX:

CREATE ASSERTION [assertion_name] CHECK ([condition]);

RESULT

Implemented SQL commands to create view and assertion.

```
 3  DECLARE
 4   X NUMBER(5);
 5  BEGIN
 6   SELECT TEST_NO INTO X
 7   FROM SAMPLE
 8   WHERE SL=1
 9   FOR UPDATE OF TEST_NO;
10
11   IF MOD(X,2) = 0 THEN
12   UPDATE SAMPLE
13   SET EVEN_OR_ODD='even'
14   WHERE SL=1;
15
16   END IF;
17   COMMIT;
18  END;
```

SL	TEST_NO	EVEN_OR_ODD
1	2	even
2	3	-
3	5	-
4	10	-
5	120	-

```
 3  DECLARE
 4   X NUMBER(5);
 5  BEGIN
 6   SELECT TEST_NO INTO X
 7   FROM SAMPLE
 8   WHERE SL=3
 9   FOR UPDATE OF TEST_NO;
10
11   IF X=4 THEN
12   UPDATE SAMPLE
13   SET EVEN_OR_ODD='even'
14   WHERE SL=3;
15
16   ELSIF X=5 THEN
17   UPDATE SAMPLE
18   SET EVEN_OR_ODD='odd'
19   WHERE SL=3;
20
21   END IF;
22   COMMIT;
23  END;
```

SL	TEST_NO	EVEN_OR_ODD
1	2	even
2	3	odd
3	5	odd
4	10	-

```
 3  DECLARE
 4   X NUMBER(5);
 5  BEGIN
 6   SELECT TEST_NO INTO X
 7   FROM SAMPLE
 8   WHERE SL=2
 9   FOR UPDATE OF TEST_NO;
10
11   IF MOD(X,2) = 0 THEN
12   UPDATE SAMPLE
13   SET EVEN_OR_ODD='even'
14   WHERE SL=2;
15
16   ELSE
17   UPDATE SAMPLE
18   SET EVEN_OR_ODD='odd'
19   WHERE SL=2;
20
21   END IF;
22   COMMIT;
23  END;
```

SL	TEST_NO	EVEN_OR_ODD
1	2	even
2	3	odd
3	5	-
4	10	-

```
 3  DECLARE
 4   X NUMBER(5);
 5   COUNTER NUMBER :=5;
 6  BEGIN
 7   SELECT TEST_NO INTO X
 8   FROM SAMPLE
 9   WHERE SL=4
10   FOR UPDATE OF TEST_NO;
11
12   WHILE COUNTER < X LOOP
13   COUNTER := COUNTER + 1;
14   DBMS_OUTPUT.PUT_LINE('COUNTER = '||COUNTER);
15   END LOOP;
16
17   IF COUNTER=X THEN
18   UPDATE SAMPLE
19   SET EVEN_OR_ODD='even'
20   WHERE SL=4;
21   END IF;
22   COMMIT;
23  END;
```

```
Statement processed.
COUNTER = 6
COUNTER = 7
COUNTER = 8
COUNTER = 9
COUNTER = 10
```

20/01/22

EXPERIMENT NO. 15

CONTROL STRUCTURES

AIM:

Implementation of various control structures like IF-THEN, IF-THEN-ELSE, IF-THEN-ELSIF, CASE, WHILE using PL/SQL

THEORY:

Following are the control structured used in PL/SQL

IF THEN - Simplest form of IF statement associating a condition with a sequence of statements enclosed by the keywords THEN and END IF. The sequence of statements is executed only if condition is true. If the condition is false or null the IF statement does nothing.

SYNTAX: IF Condition THEN
Sequence_Of_Statements
END IF;

IF THEN ELSE - Second form of IF statement adding the keyword ELSE followed by an alternate sequence of statements. The sequence of statement in the ELSE clause is executed only if condition is false.

SYNTAX: IF condition THEN
Sequence of statements 1
ELSE
Sequence of statements 2
END IF;

IF THEN ELSIF - Used to select an action from several mutually exclusive alternatives. This form uses the keyword ELSIF to

```
3  DECLARE
4   X NUMBER(5);
5  BEGIN
6   SELECT TEST_NO INTO X
7   FROM SAMPLE
8   WHERE SL=5
9   FOR UPDATE OF TEST_NO;
10
11  CASE X
12  WHEN 99 THEN
13  UPDATE SAMPLE
14  SET EVEN_OR_ODD='odd'
15  WHERE SL=5;
16
17  WHEN 120 THEN
18  UPDATE SAMPLE
19  SET EVEN_OR_ODD='even'
20  WHERE SL=5;
21
22  ELSE
23  UPDATE SAMPLE
24  SET EVEN_OR_ODD='err'
25  WHERE SL=5;
26  END CASE;
27
28  COMMIT;
29 END;
```

SL	TEST_NO	EVEN_OR_ODD
1	2	-
2	3	-
3	5	-
4	10	-
5	120	-

SL	TEST_NO	EVEN_OR_ODD
1	2	even
2	3	odd
3	5	odd
4	10	even
5	120	even

introduce additional conditions,

SYNTAX: IF Condition1 THEN
Sequence of statements1
ELSIF Condition2 THEN
sequence of statements2
ELSE
Sequence of statements3
END IF;

CASE: This selects one sequence of statements to execute. To select the sequence CASE statement uses a selector rather than multiple Boolean expressions.

SYNTAX: CASE Selector
WHEN expression1 THEN Sequence of Statements1
WHEN expression2 THEN sequence of Statements2
:
WHEN expressionN THEN Sequence of statementsN
[ELSE sequence-of-statements N+1;]
END CASE [label name];

WHILE: WHILE loop statement associates a condition with a sequence of statements enclosed by the keywords LOOP and END LOOP

SYNTAX: WHILE condition LOOP
Sequence of statements
END LOOP;

RESULT:

Successfully implemented control structures.

SQL Worksheet

```
DECLARE
    p NUMBER(4);
    q NUMBER(4);
    r NUMBER(4);
    s NUMBER(4);
PROCEDURE proc(a NUMBER, b NUMBER, c OUT NUMBER, d OUT NUMBER, e OUT NUMBER, f OUT NUMBER) IS
BEGIN
    c:= a+b;
    d:= a-b;
    e:= a*b;
    f:= a/b;
END;

BEGIN
    proc(6, 2, p, q, r, s);
    dbms_output.put_line(p||' '||q||' '||r||' '||s);
END;
```

```
Statement processed.
8 4 12 3
```

ITEM_ID	ITEM_NAME	PRICE
1	BISCUIT	30
2	PEN	10
3	PENCIL	5

Download CSV
3 rows selected.

SQL Worksheet

```
CREATE OR REPLACE TRIGGER display_price_changes
BEFORE DELETE OR INSERT OR UPDATE ON SHOP
FOR EACH ROW
WHEN (NEW.ITEM_ID > 0)
DECLARE
   price_diff number;
BEGIN
   price_diff := :NEW.PRICE  - :OLD.PRICE;
   dbms_output.put_line('Old price: ' || :OLD.PRICE);
   dbms_output.put_line('New price: ' || :NEW.PRICE);
   dbms_output.put_line('Price Difference: ' || price_diff);
END;
/
```

```
Trigger created.
```

24/01/22

32

EXPERIMENT NO. 16

PROCEDURES, TRIGGERS & FUNCTIONS

AIM:

To create Procedures, Triggers and Functions.

THEORY

PROCEDURE:

A procedure is a named PL/SQL block which performs one or more specific tasks. This is similar to a procedure in other programming languages. A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists of the declaration section, execution section, and exception section similar to a general PL/SQL block.

Three ways to pass the parameters to procedures are:

IN type	Used to send values to stored procedures.
OUT type	To get values from stored procedures. Similar to a return type in functions.
IN OUT type	Used to send values and get values from stored procedures.

SYNTAX: CREATE [OR REPLACE] PROCEDURE procedure-name [(<Argument> {IN, OUT, IN OUT} <Datatype> ---)]

IS

Declaration section <variable, constant>;

BEGIN

Execution section

EXCEPTION

Exception section

END

```
1  UPDATE SHOP SET PRICE = PRICE + 10;
2
3
```

```
3 row(s) updated.
Old price: 30
New price: 40
Price Difference: 10
Old price: 10
New price: 20
Price Difference: 10
Old price: 5
New price: 15
Price Difference: 10
```

```
1  DECLARE
2     a NUMBER;
3     b NUMBER;
4     c NUMBER;
5  FUNCTION findMax(x IN NUMBER, y IN NUMBER)
6  RETURN NUMBER
7  IS
8     z NUMBER;
9  BEGIN
10    IF x > y THEN
11       z:= x;
12    ELSE
13       z:= y;
14    END IF;
15    RETURN z;
16 END;
17
18 BEGIN
19    a:= 79;
20    b:= 82;
21    c := findMax(a, b);
22    dbms_output.put_line(' Maximum of (79,82): ' || c);
23 END;
24 /
25
```

```
Statement processed.
Maximum of (79,82): 82
```


FUNCTIONS : The major difference between a procedure and a function is A function must always return a value, but a procedure may or may not return a value.

SYNTAX : CREATE [OR REPLACE] FUNCTION function_name [Parameters]
RETURN return_datatype ; {IS, AS}
Declaration_section <Variable, Constant> ;
BEGIN
Execution_section
Return return_Variable ;
EXCEPTION
Exception_section
Return return_Variable ;
END ;

TRIGGER - Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to data manipulation statements, database definition statements or a database operation (LOGON, LOGOFF, SHUTDOWN)

SYNTAX : CREATE [OR REPLACE] TRIGGER trigger_name {BEFORE | AFTER | INSTEADOF}
{INSERT [OR] | UPDATE [OR] | DELETE} [OF Col_name]
ON table_name [REFERENCING OLD AS O NEW AS n] [FOR EACH ROW]
WHEN (Condition)
DECLARE
Declaration-Statements
BEGIN
Executable-Statements
EXCEPTION
Exception-handling Statements
END ;
/

RESULT :
The queries were successfully executed and desired outputs were obtained.

```
CREATE OR REPLACE PACKAGE c_package AS
   -- Adds a customer
   PROCEDURE addCustomer(c_id   customers.cid%type,
   c_name customers.cname%type,
   c_sal  customers.salary%type);

   -- Removes a customer
   PROCEDURE delCustomer(c_id  customers.cid%TYPE);
   --Lists all customers
   PROCEDURE listCustomer;

END c_package;
```

Package created.

```
DECLARE
   code customers.cid%type:= 7;
BEGIN
   c_package.addcustomer(6, 'Ebin', 4500);
   c_package.addcustomer(7, 'Thomas', 4500);
   c_package.listcustomer;
   c_package.delcustomer(code);
   c_package.listcustomer;
END;
```

```
CREATE OR REPLACE PACKAGE BODY c_package AS
   PROCEDURE addCustomer(c_id  customers.cid%type,
      c_name customers.cname%type,
      c_sal   customers.salary%type)
   IS
   BEGIN
      INSERT INTO customers (cid,cname,salary)
         VALUES(c_id, c_name,c_sal);
   END addCustomer;

   PROCEDURE delCustomer(c_id   customers.cid%type) IS
   BEGIN
      DELETE FROM customers
      WHERE cid = c_id;
   END delCustomer;

   PROCEDURE listCustomer IS
   CURSOR c_customers is
      SELECT  cname FROM customers;
   TYPE c_list is TABLE OF customers.cname%type;
   name_list c_list := c_list();
   counter integer := 0;
   BEGIN
      FOR n IN c_customers LOOP
      counter := counter +1;
      name_list.extend;
      name_list(counter) := n.cname;
      dbms_output.put_line('Customer (' ||counter|| ') '||name_list(counter));
      END LOOP;
   END listCustomer;

END c_package;
```

Package Body created

27/01/22

EXPERIMENT NO. 17

PACKAGES

AIM :

To create packages.

THEORY :

Packages are schema objects that groups logically related PL/SQL types, variables and subprograms having 2 mandatory parts- 1. Package Specification 2. Package body or definition

Package Specification

It is the interface to the package. It just DECLARES the types, variables, constants, exceptions, cursors and subprograms that can be referenced from outside the package. It contains all information about the package content but excludes the code for the subprograms. All objects plaud in the specification are called public objects. Any subprogram not in specification but coded in the package body is called private object.

SYNTAX: CREATE PACKAGE Cust_sal AS
PROCEDURE find_sal (c_id customers.id%type);
END Cust_sal;

Package Body

The package body has the codes for various methods declared in package specification and other private declarations which are hidden from the code outside the package. The CREATE PACKAGE BODY statement is used for creating the package body. Syntax to access elements - package_name.element_name

RESULT :

Successfully implemented packages.

```
create table employee22(ename varchar(20),ssn int primary key,salary int,experience int);
insert into employee22 values('Rani',100,10000,5);
insert into employee22 values('Roopa',101,11000,4);
insert into employee22 values('Amar',102,12000,6);
insert into employee22 values('Babu',103,12000,3);
insert into employee22 values('Anu',104,10000,7);
insert into employee22 values('Samuel',105,10000,5);
/
```

31/01/22

EXPERIMENT NO.18

CURSORS

AIM:

To create cursors

THEORY:

CURSORS

A cursor is a pointer to context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by an SQL statement. The set of rows the cursor holds is referred to as the active set.

DECLARING THE CURSOR

This defines the cursor with a name and the associated SELECT statement. For example -

```
CURSOR c_customerss IS
  SELECT id, name, address FROM customerss;
```

OPENING THE CURSOR

This allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, the above defined cursor will be opened by OPEN c_customerss;

```
declare
exp employee22.experience%type;
sal employee22.salary%type;
enam employee22.ename%type;
sn employee22.ssn%type;
cursor crs1 is  select ename,ssn,experience,salary from employee22 for update of salary;
Begin
open crs1;
loop fetch crs1 into enam,sn,exp,sal;
exit when crs1%notfound;
if exp>5 then
sal:=sal*1.1;
else
sal:=sal+500;
end if;
```

```
SQL> select *from employee22;

ENAME                SSN     SALARY EXPERIENCE
-------------------- ------- ------ ----------
Rani                 100     10000           5
Roopa                101     11000           4
Amar                 102     12000           6
Babu                 103     12000           3
Anu                  104     10000           7
Samuel               105     10000           5

6 rows selected.
```

```
SQL> select *fron employee22;

ENAME                SSN     SALARY EXPERIENCE
-------------------- ------- ------ ----------
Rani                 100     10500           5
Roopa                101     11500           4
Amar                 102     13200           6
Babu                 103     12500           3
Anu                  104     11000           7
Samuel               105     10500           5

6 rows selected.
```

```
SQL> @plsql/crs1.sql;
Rani 100 5 10500
Roopa 101 4 11500
Amar 102 6 13200
Babu 103 3 12500
Anu 104 7 11000
Samuel 105 5 10500

PL/SQL procedure successfully completed.
```


FETCHING THE CURSOR

Fetching the cursor involves accessing one row at a time. For example, row from the above opened cursor can be fetched as follows:

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

CLOSING THE CURSOR

Closing the cursor means releasing the allocated memory. The above opened cursor can be closed as follows:

```
CLOSE c_customerss;
```

RESULT:

Successfully implemented Cursors.

```
DECLARE
temp varchar(20);

BEGIN
SELECT student_id into temp from Students where Student_name='Pranav Jayashankar';
dbms_output.put_line('Value in "temp" : '|| temp);

EXCEPTION
WHEN no_data_found THEN
        dbms_output.put_line('ERROR: NO RECORD FOUND!');
END;
```

```
Statement processed
Value in "temp" : 48
```

```
DECLARE
temp varchar(20);

BEGIN
SELECT student_id into temp from Students where Student_name='Pranav';
dbms_output.put_line('Value in "temp" : '|| temp);

EXCEPTION
WHEN no_data_found THEN
        dbms_output.put_line('ERROR: NO RECORD FOUND!');
END;
```

```
Statement processed
ERROR: NO RECORD FOUND!
```

03/02/22

EXPERIMENT NO. 19

EXCEPTION HANDLING

AIM :

To create PL/SQL blocks for exception handling

THEORY :

An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error conditions. Two types of exceptions are :
1. System-defined 2. User-defined exceptions

Syntax for Exception Handling

```
DECLARE
    <declarations section>
BEGIN
    <executable commands>
EXCEPTION
    <exception handling goes here>
    WHEN exception1 THEN
      exception1-handling statements
    WHEN exception2 THEN
      exception2-handling statements
    ⋮
```

WHEN others THEN
exception 3 - handling statements
END;

Raising Exceptions

These are raised by database server automatically if there is any internal database error. But exceptions can be raised explicitly by the programmer by using command RAISE

SYNTAX: DECLARE
exception_name EXCEPTION;
BEGIN
IF condition THEN
RAISE exception_name;
END IF;
EXCEPTION
WHEN exception_name THEN
statement;
END;

User-defined Exceptions

PL/SQL allows to define own exceptions according to the need of one's program. A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure DBMS_STANDARD.RAISE_APPLICATION_ERROR

SYNTAX FOR DECLARING EXCEPTION - DECLARE
my-exception EXCEPTION;

RESULT:
The queries were successfully executed.

```
oracle@administrator-ThinkCentre-E73:~$ mongo
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.3
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2022-04-12T09:13:26.928+0530 I STORAGE  [initandlisten]
2022-04-12T09:13:26.928+0530 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger stora
2022-04-12T09:13:26.928+0530 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodnotes-filesystem
2022-04-12T09:13:26.928+0530 I STORAGE  [initandlisten]
on engine
2022-04-12T09:13:26.928+0530 I STORAGE  [initandlisten]
2022-04-12T09:13:34.900+0530 I CONTROL  [initandlisten]
2022-04-12T09:13:34.900+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2022-04-12T09:13:34.900+0530 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2022-04-12T09:13:34.900+0530 I CONTROL  [initandlisten]
2022-04-12T09:13:34.900+0530 I CONTROL  [initandlisten]
> use stud
switched to db stud
> db
stud
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> db.student
stud.student
> var mystudent=
... [
... {
... "Name" : "Smith",
... "Student_number" : 17,
... "Class" : 1,
... "Major" : "cs"
... },
... {
... "Name" : "Smith",
... "Student_number" : 17,
... "Class" : 1,
... "Major" : "cs"
... },
... {
... "Name" : "Brown",
... "Student_Number" : 8,
... "Class" : 2,
... "Major" : "cs"
... },
... ];
> db.student.insert(mystudent);
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
> db.student.find().forEach(printjson)
{
        "_id" : ObjectId("6254feba2afb4a8fd68830b2"),
        "Name" : "Smith",
        "Student_number" : 17,
        "Class" : 1,
        "Major" : "cs"
}
{
        "_id" : ObjectId("6254feba2afb4a8fd68830b3"),
        "Name" : "Brown",
        "Student_Number" : 8,
        "Class" : 2,
        "Major" : "cs"
}
```

01/02/22

39

EXPERIMENT No:20

NOSQL DATABASES AND CRUD

AIM:

Familiarisation of NOSQL database and CRUD operations.

THEORY:

NOSQL databases are non-tabular databases and store data differently than relational databases. NOSQL databases comes in a variety of types based on their data model. The main types are document, key-value, wide-column and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.

MongoDB provides a set of some basic but most essential operations that will help you to easily interact with the MongoDB server and these operations are known as CRUD operations.

1. Create:

The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database. Create operations are performed using the following methods

```
{
        _id" : ObjectId("6254feba2afb4a8fd68830b3"),
        "Name" : "Brown",
        "Student_number" : 8,
        "Class" : 2,
        "Major" : "cs"
}
> db.course
stud.course
> var mycourse=
... [
... {
...  "course_name" : "maths",
...  "course_number" : "math2410",
...  "credit_hours" : 3,
...  "department" : "math",
...  "batch" : 2019
... },
... {
...  "course_name" : "data structures",
2022-04-12T10:00:45.623+0530 E QUERY    [thread1] SyntaxError: missing : after property id @(shell):12:16
> var mycourse=
... [
... {
...  "course_name" : "maths",
...  "course_number" : "math2410",
...  "credit_hours" : 3,
...  "department" : "math"
... },
... {
...
...  "course_name" : "database",
...  "course_number" : "cs3380",
...  "credit_hours" : 3,
...  "department" : "cs"
... },
... ];
```

```
...  "credit_hours" : 3,
...  "department" : "math"
... },
... {
...
...  "course_name" : "database",
...  "course_number" : "cs3380",
...  "credit_hours" : 3,
...  "department" : "cs"
... },
... ];
> db.course.insert(mycourse);
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
> db.course.find().forEach(printjson)
{
        "_id" : ObjectId("6255039e2afb4a8fd68830b4"),
        "course_name" : "maths",
        "course_number" : "math2410",
        "credit_hours" : 3,
        "department" : "math"
}
{
        "_id" : ObjectId("6255039e2afb4a8fd68830b5"),
        "course_name" : "database",
        "course_number" : "cs3380",
        "credit_hours" : 3,
        "department" : "cs"
}
```

db.collection.insertOne() → Used to insert a single document in the collection

db.collection.insertMany() → Used to insert multiple documents in the collection.

2) Read

The Read operations are used to retrieve documents from the collection. It is used to query a collection for a document. Read operations are performed using the following method

db.collection.find() → Used to retrieve documents from the collection

3) Update

The update operations are used to update or modify the existing document in the collection. Update operations are performed using the following method

db.collection.updateOne() → Used to update a single document in the collection

db.collection.updateMany() → Used to update multiple documents in the collection.

db.collection.replaceOne() → Used to replace single document in the collection

4) Delete

The delete operations are used to delete or remove the documents from a collection. Delete operations can be performed using the following methods:

db.collection.deleteOne() → Used to delete a single document in the collection

db.collection.deleteMany() → Used to delete multiple documents from the collection.

RESULT:

The queries were successfully executed and desired outputs were obtained.