Experiment NO – 1

Aim: Introduction of DBMS (with basic commands DDL, DML, DCL, TCL)

Theory:

Database Management System (DBMS) refers to the technology solution used to optimize and manage the storage and retrieval of data from databases. DBMS offers a systematic approach to manage databases via an interface for users as well as workloads accessing the databases via apps.

DBMS has the following key components:

Software. DBMS is primarily a software system that can be considered as a management console or an interface to interact with and manage databases.

Data. DBMS contains operational data, access to database records and metadata as a resource to perform the necessary functionality.

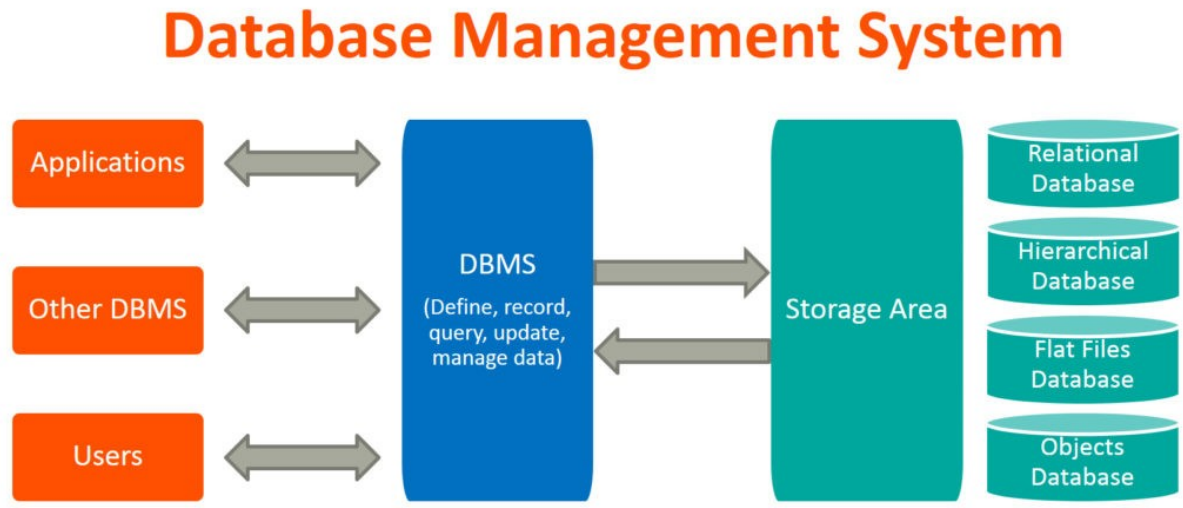
Database languages. These are components of the DBMS used to access, modify, store, and retrieve data items from databases; specify database schema.

Query processor. As a fundamental component of the DBMS, the query processor acts as an intermediary between users and the DBMS data engine in order to communicate query requests.

Runtime database manager. A centralized management component of DBMS that handles functionality associated with runtime data, which is commonly used for context-based database access.

Database manager. Unlike the runtime database manager that handles queries and data at runtime, the database manager performs DBMS functionality associated with the data within databases.

Database engine. This is the core software component within the DBMS solution that performs the core functions associated with data storage and retrieval.



Database language

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

1. DDL – Data Definition Language
2. DML – Data Manipulation Language
3. DCL – Data Control Language
4. TCL – Transaction Control Language

DDL(Data Definition Language) : DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

Examples of DDL commands:

- **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- **DROP** – is used to delete objects from the database.
- **ALTER**-is used to alter the structure of the database.
- **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT** –is used to add comments to the data dictionary.
- **RENAME** –is used to rename an object existing in the database.

DML(Data Manipulation Language) : The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

Examples of DML:

- **INSERT** – is used to insert data into a table.
- **UPDATE** – is used to update existing data within a table.
- **DELETE** – is used to delete records from a database table.

DCL(Data Control Language) : DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

Examples of DCL commands:

- **GRANT**-gives user's access privileges to database.
- **REVOKE**-withdraw user's access privileges given by using the GRANT command

TCL(transaction Control Language) : TCL commands deals with the transaction within the database.

Examples of TCL commands:

- **COMMIT**– commits a Transaction.
- **ROLLBACK**– rollbacks a transaction in case of any error occurs.
- **SAVEPOINT**–sets a savepoint within a transaction.
- **SET TRANSACTION**–specify characteristics for the transaction.

Benefits of DBMS

Introducing DBMS software to manage databases results in the following benefits:

- **Data security.** DBMS allows organizations to enforce policies that enable compliance and security.
- **Data sharing.** Fast and efficient collaboration between users.
- **Data access and auditing.** Controlled access to databases. Logging associated access activities allows organizations to audit for security and compliance.
- **Data integration.** Instead of operating island of database resources, a single interface is used to manage databases with logical and physical relationships.
- **Abstraction and independence.** Organizations can change the physical schema of database systems without necessitating changes to the logical schema that govern database relationships.
- **Uniform management and administration.** A single console interface to perform basic administrative tasks makes the job easier for database admins and IT users.

Experiment NO – 2

Aim: To implement Data Definition language - Create, alter, drop, truncate - To implement Constraints. (a). Primary key, (b).Foreign Key, (c). Check, (d). Unique, (e). Null, (f). Not null , (g) . Default, (h). Enable Constraints, (i). Disable Constraints (j). Drop Constraints

Implementation

Creating Table

```
CREATE TABLE students (
    studentID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

esult Grid | Filter Rows: | Export: | Wr

Field	Type	Null	Key	Default	Extra
StudentID	int	YES		NULL	
LastName	varchar(255)	YES		NULL	
FirstName	varchar(255)	YES		NULL	
Address	varchar(255)	YES		NULL	
City	varchar(255)	YES		NULL	

Alter table

```
ALTER TABLE students

ADD Email varchar(255);
```

Field	Type	Null	Key	Default	Extra
StudentID	int	YES		NULL	
LastName	varchar(255)	YES		NULL	
FirstName	varchar(255)	YES		NULL	
Address	varchar(255)	YES		NULL	
City	varchar(255)	YES		NULL	
Email	varchar(255)	YES		NULL	

Inserting into table

```
INSERT INTO students (StudentID, LastName, FirstName, Address, City, Email)

VALUES (1123, 'Tom','Erichsen', '11/2 Delhi', 'Delhi', 'tom@gmail.com');
```

	StudentID	LastName	FirstName	Address	City	Email
▶	1123	Tom	Erichsen	11/2 Delhi	Delhi	tom@gmail.com

Truncating Table

TRUNCATE TABLE students;

	StudentID	LastName	FirstName	Address	City	Email

Dropping table

DROP Table students;

	Tables_in_mydb

Primary key,null,not null,default,check,unique constraints

CREATE TABLE People(Id INTEGER, LastName TEXT NOT NULL,

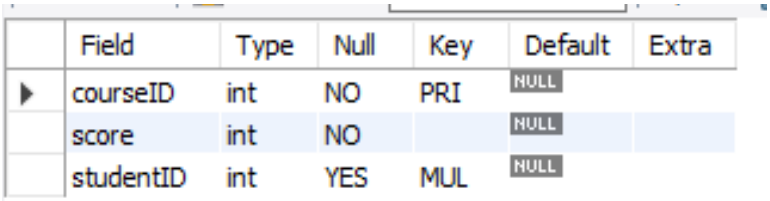
FirstName TEXT NOT NULL, City VARCHAR(55));

```
CREATE TABLE students (
    studentID int primary key,
    LastName varchar(255) NULL,
    FirstName varchar(255) NOT NULL,
    Address varchar(255) ,
    City varchar(255) default 'Delhi',
    emailid varchar(255) NOT NULL UNIQUE,
    Age int CHECK (Age>=18)
);
```

Field	Type	Null	Key	Default	Extra
studentID	int	NO	PRI	NULL	
LastName	varchar(255)	YES		NULL	
FirstName	varchar(255)	NO		NULL	
Address	varchar(255)	YES		NULL	
City	varchar(255)	YES		Delhi	
emailid	varchar(255)	NO	UNI	NULL	
Age	int	YES		NULL	

Foreign key

```
CREATE TABLE marks (
    courseID int NOT NULL,
    score int NOT NULL,
    studentID int,
    PRIMARY KEY (courseID),
    FOREIGN KEY (studentID) REFERENCES students(studentID)
);
```



Field	Type	Null	Key	Default	Extra
courseID	int	NO	PRI	NULL	
score	int	NO		NULL	
studentID	int	YES	MUL	NULL	

Experiment NO – 3

Aim: To implementation on DML, TCL and DRL (a) Insert, (b) Select, (c) Update, (d) Delete, (e) commit, (f) rollback, (g) save point, (i) Like'%', (j)Relational Operator.

Implementation

Inserting into table

```
INSERT INTO students (studentID, FirstName,LastName, Address,City,emailid,age)
VALUES ('2213', 'John','Doe','23/1 New Delhi', 'Delhi', 'John@gmail.com',20),
 ('2214','Raj','Kumar','11/2 New Delhi','Delhi','raj@gmail.com',21),
 ('2215','Ravi','Kumar','13/2 Borivli','Mumbai','ravi@gmail.com',21),
 ('2216','Ajay','aditya','122 South Mumbai','Mumbai','aditya@gmail.com',22);
```

Select statement

```
select * from students;
```

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21
2216	Ajay	aditya	122 South Mumbai	Mumbai	aditya@gmail.com	22

Updating table

```
UPDATE students
SET FirstName = 'aditya', LastName = ‘Singh’
WHERE studentID = 2216;
select * from students where studentID = 2216;
```

studentID	FirstName	LastName	Address	City	emailid	Age
2216	aditya	Singh	122 South Mumbai	Mumbai	aditya@gmail.com	22

Deleting table

```
DELETE FROM students WHERE studentID=2216;
```

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21

Commit

set autocommit = 0;

INSERT INTO students (studentID, FirstName,LastName, Address,City,emailid,age)

VALUES ('2216', 'Aditya','Singh','23/1 New Delhi', 'Delhi', 'aditya@gmail.com',20);

In new connection

Select * from students;

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21

In old connection

Commit;

In new connection

Select * from students;

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21
2216	Aditya	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20

Rollback

set autocommit = 0;

INSERT INTO students (studentID, FirstName,LastName, Address,City,emailid,age)

VALUES ('2217', 'Jai','Singh','23/1 New Delhi', 'Delhi', 'jai@gmail.com',20);

Select * from students;

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21
2216	Aditya	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20
2217	Jai	Singh	23/1 New Delhi	Delhi	jai@gmail.com	20

rollback;

select * from students;

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21
2216	Aditya	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20

Savepoint

START TRANSACTION;

SAVEPOINT initial_save;

UPDATE students

SET firstName = 'Harry'

where studentID = 2216;

select * from students;

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	Kumar ivli	Mumbai	ravi@gmail.com	21
2216	Harry	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20

ROLLBACK TO initial_save;

select * from students;

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21
2216	Aditya	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20

Like operator

SELECT * FROM students

WHERE FirstName LIKE 'a%';

studentID	FirstName	LastName	Address	City	emailid	Age
2216	Aditya	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20

SELECT * FROM students

WHERE FirstName LIKE '%i';

studentID	FirstName	LastName	Address	City	emailid	Age
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21

Relational operator

SELECT * FROM students

WHERE age > 20;

studentID	FirstName	LastName	Address	City	emailid	Age
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21

SELECT * FROM students

WHERE age <> 21;

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2216	Aditya	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20

SELECT * FROM students

WHERE FirstName = 'Aditya';

studentID	FirstName	LastName	Address	City	emailid	Age
2216	Aditya	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20

SELECT * FROM students

WHERE age <= 20;

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2216	Aditya	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20

Experiment NO – 4

Aim: To implement Nested Queries & Join Queries - To implementation of Nested Queries - (a) Inner join, (b) Left join, (c) Right join (d) Full join

Implementation

Nested Queries

```
SELECT * FROM students
  WHERE studentID IN (SELECT studentId
      FROM students
      WHERE age > 20) ;
```

studentID	FirstName	LastName	Address	City	emailid	Age
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21

```
INSERT INTO students2
  SELECT * FROM students
  WHERE studentID IN (SELECT studentID
  FROM students) ;
select * from students;
```

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	21
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	21
2216	Aditya	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20

```
UPDATE students
SET age = age + 1
WHERE age IN (SELECT age FROM students2
WHERE age >= 20 );
select * from students;
```

studentID	FirstName	LastName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	21
2214	Raj	Kumar	11/2 New Delhi	Delhi	raj@gmail.com	22
2215	Ravi	Kumar	13/2 Borivli	Mumbai	ravi@gmail.com	22
2216	Aditya	Singh	23/1 New Delhi	Delhi	ravi@gmail.com	21

```
DELETE FROM students2
   WHERE age IN (SELECT age FROM students
      WHERE age = 21 );
select * from students2;
```

studentID	LastName	FirstName	Address	City	emailid	Age
2213	John	Doe	23/1 New Delhi	Delhi	John@gmail.com	20
2216	Aditya	Singh	23/1 New Delhi	Delhi	aditya@gmail.com	20

Inner Join

```
SELECT students.FirstName, students.address, marks.courseID,marks.score
FROM students
INNER JOIN marks
ON students.studentID = marks.studentID;
```

FirstName	address	courseID	score
John	23/1 New Delhi	100	80
John	23/1 New Delhi	102	60
Raj	11/2 New Delhi	100	70
Raj	11/2 New Delhi	102	65

Left join

```
SELECT students.FirstName, students.address, marks.courseID,marks.score
FROM students
LEFT JOIN marks
ON students.studentID = marks.studentID;
```

FirstName	address	courseID	score
John	23/1 New Delhi	100	80
John	23/1 New Delhi	102	60
Raj	11/2 New Delhi	100	70
Raj	11/2 New Delhi	102	65
Ravi	13/2 Borivli	NULL	NULL
Aditya	23/1 New Delhi	NULL	NULL

Right Join

```
SELECT students.FirstName, students.address, marks.courseID,marks.score
FROM students
RIGHT JOIN marks
ON students.studentID = marks.studentID;
```

FirstName	address	courseID	score
John	23/1 New Delhi	100	80
John	23/1 New Delhi	102	60
Raj	11/2 New Delhi	100	70
Raj	11/2 New Delhi	102	65

Full Join

```
LEFT JOIN marks
ON students.studentID = marks.studentID
union all
SELECT students.FirstName , students.address, marks.courseID,marks.score
FROM students
RIGHT JOIN marks
ON students.studentID = marks.studentID;
```

FirstName	address	courseID	score
John	23/1 New Delhi	100	80
John	23/1 New Delhi	102	60
Raj	11/2 New Delhi	100	70
Raj	11/2 New Delhi	102	65
Ravi	13/2 Borivli	NULL	NULL
Aditya	23/1 New Delhi	NULL	NULL
John	23/1 New Delhi	100	80
John	23/1 New Delhi	102	60
Raj	11/2 New Delhi	100	70
Raj	11/2 New Delhi	102	65

Experiment NO – 5

Aim: To implement Views - (a) View, (b) joint view, (c) force view, (d) View with check option

Implementation

Creating View

CREATE VIEW student_address AS SELECT FirstName,address FROM students;

select * from student_address;

FirstName	address
John	23/1 New Delhi
Raj	11/2 New Delhi
Ravi	13/2 Borivli
Aditya	23/1 New Delhi

Creating view with Join

CREATE VIEW studentmarks AS SELECT students.FirstName,marks.score

FROM students

INNER JOIN marks ON

students.studentID = marks.studentID;

select * from studentmarks;

FirstName	score
John	80
John	60
Raj	70
Raj	65

Creating Force view

//table courses does not exists but view created

CREATE FORCE VIEW course AS SELECT FirstName,address FROM courses;

Creating view with check option

CREATE VIEW studentnm AS

```
SELECT   studentID,FirstName,LastName,Address,City,emailid,Age
FROM students WHERE
FirstName LIKE '%i'
with check option;
INSERT INTO studentnm(studentID,FirstName,LastName,Address,City,emailid,Age)
VALUES(1227,'John','Smith','New Delhi','Delhi','johnsmith@classicmodelcars.com',23);
```

//Table will not be updated from view with check option

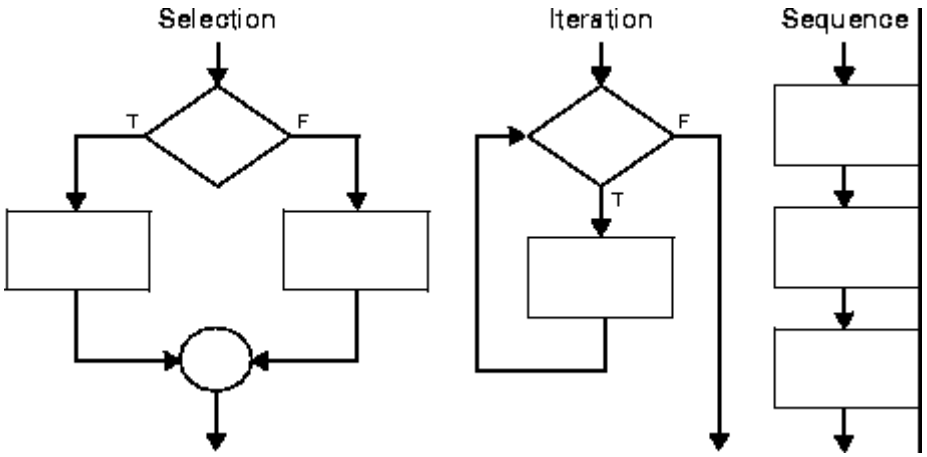
```
11  12:35:44  INSERT INTO studentnm(studentID,FirstName,LastName,Address,City,emailid,Age) VA...  Error Code: 1369. CHECK OPTION failed 'mydb.studentnm'
```

Experiment NO – 6

Aim: Describe about Control Structure

Theory

According to the *structure theorem*, any computer program can be written using the basic control structures shown in Figure 3-1. They can be combined in any way necessary to deal with a given problem.



The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is true or false. A *condition* is any variable or expression that returns a Boolean value (TRUE or FALSE). The iteration structure executes a sequence of statements repeatedly as long as a condition holds true. The sequence structure simply executes a sequence of statements in the order in which they occur.

Conditional Control: IF Statements

Often, it is necessary to take alternative actions depending on circumstances.
The IF statement lets you execute a sequence of statements conditionally. That is, whether the sequence is executed or not depends on the value of a condition. There are three forms of IF statements: IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSIF.

IF-THEN

The simplest form of IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF (not ENDIF), as follows:

```
IF condition THEN
   sequence_of_statements;
END IF;
```

The sequence of statements is executed only if the condition yields TRUE. If the condition yields FALSE or NULL, the IF statement does nothing. In either case, control passes to the next statement. An example follows:

```
IF sales > quota THEN
  compute_bonus(empid);
  UPDATE payroll SET pay = pay + bonus WHERE empno = emp_id;
END IF;
```

You might want to place brief IF statements on a single line, as in

```
IF x > y THEN high := x; END IF;
```

IF-THEN-ELSE

The second form of IF statement adds the keyword ELSE followed by an alternative sequence of statements, as follows:

```
IF condition THEN
  sequence_of_statements1;
ELSE
  sequence_of_statements2;
END IF;
```

The sequence of statements in the ELSE clause is executed only if the condition yields FALSE or NULL. Thus, the ELSE clause ensures that a sequence of statements is executed. In the following example, the first or second UPDATE statement is executed when the condition is true or false, respectively:

```
IF trans_type = 'CR' THEN
  UPDATE accounts SET balance = balance + credit WHERE ...
ELSE
  UPDATE accounts SET balance = balance - debit WHERE ...
END IF;
```

The THEN and ELSE clauses can include IF statements. That is, IF statements can be nested, as the following example shows:

```
IF trans_type = 'CR' THEN
  UPDATE accounts SET balance = balance + credit WHERE ...
ELSE
  IF new_balance >= minimum_balance THEN
    UPDATE accounts SET balance = balance - debit WHERE ...
  ELSE
```

```
    RAISE insufficient_funds;
  END IF;
END IF;
```

IF-THEN-ELSIF

Sometimes you want to select an action from several mutually exclusive alternatives. The third form of IF statement uses the keyword ELSIF (not ELSEIF) to introduce additional conditions, as follows:

```
IF condition1 THEN
  sequence_of_statements1;
ELSIF condition2 THEN
  sequence_of_statements2;
ELSE
  sequence_of_statements3;
END IF;
```

If the first condition yields FALSE or NULL, the ELSIF clause tests another condition. An IF statement can have any number of ELSIF clauses; the final ELSE clause is optional. Conditions are evaluated one by one from top to bottom. If any condition yields TRUE, its associated sequence of statements is executed and control passes to the next statement. If all conditions yield FALSE or NULL, the sequence in the ELSE clause is executed. Consider the following example:

```
BEGIN
  ...
  IF sales > 50000 THEN
    bonus := 1500;
  ELSIF sales > 35000 THEN
    bonus := 500;
  ELSE
    bonus := 100;
  END IF;
  INSERT INTO payroll VALUES (emp_id, bonus, ...);
END;
```

Experiment NO – 7

Aim: To write a PL/SQL block for Addition of Two Numbers

Implementation

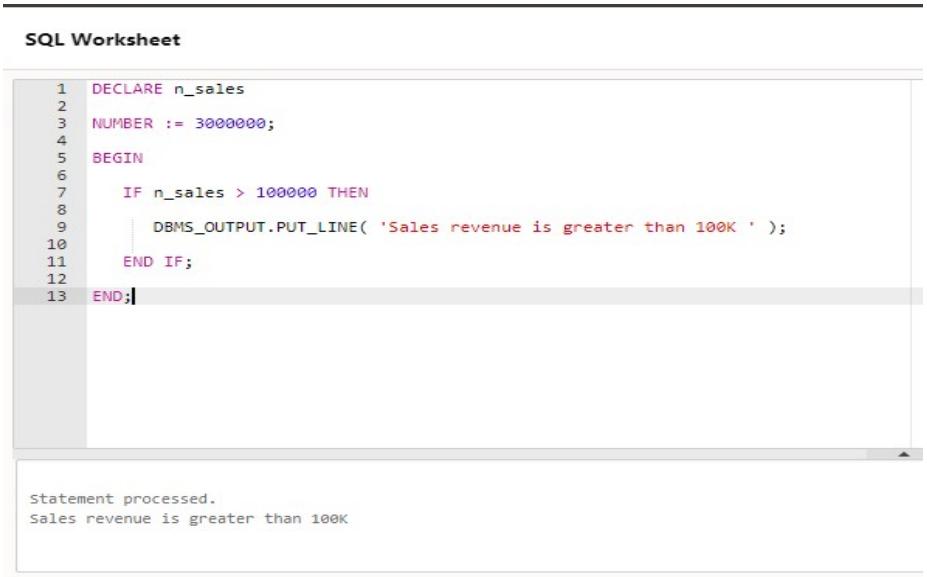
```
declare
x number(5);
y number(5);
z number(7);
begin
x:=13;
y:=27;
z:=x+y;
dbms_output.put_line('Sum is '||z);
end;
```

Experiment NO – 8

Aim: To write a PL/SQL block for IF Condition

Implementation

```
DECLARE n_sales
NUMBER := 3000000;
BEGIN
  IF n_sales > 100000 THEN
    DBMS_OUTPUT.PUT_LINE( 'Sales revenue is greater than 100K ' );
  END IF;
END;
```



Experiment NO – 9

Aim: To write a PL/SQL block for IF and else condition

Implementation

```
DECLARE
a NUMBER:=14;
BEGIN
dbms_output.put_line ('Program started');
IF( mod(a,2)=0) THEN
dbms_output.put_line('a is even number' );
ELSE
dbms_output.put_line('a is odd number1');
END IF;
dbms_output.put_line ('Program completed');
END;
```

SQL Worksheet

```
1  DECLARE
2
3  a NUMBER:=14;
4
5  BEGIN
6
7  dbms_output.put_line ('Program started');
8
9  IF( mod(a,2)=0) THEN
10
11 dbms_output.put_line('a is even number' );
12
13 ELSE
14
15 dbms_output.put_line('a is odd number1');
16
17 END IF;
18
19 dbms_output.put_line ('Program completed');
20
21 END;
```

```
Statement processed.
Program started
a is even number
Program completed
```

Experiment NO – 10

Aim: To write a PL/SQL block for greatest of three numbers using IF AND ELSEIF

Implementation

```
DECLARE
        a NUMBER := 26;
        b NUMBER := 18;
        c NUMBER := 11;
BEGIN
        IF a > b
        AND a > c THEN
        dbms_output.Put_line('Greatest number is '||a);
        ELSIF b > a  AND  b > c THEN
        dbms_output.Put_line('Greatest number is '||b);
        ELSE
        dbms_output.Put_line('Greatest number is '||c);
        END IF;
END;
```

SQL Worksheet

```
1
2   DECLARE
3
4       a NUMBER := 26;
5
6       b NUMBER := 18;
7
8       c NUMBER := 11;
9   BEGIN
10
11      IF a > b
12      AND a > c THEN
13
14      dbms_output.Put_line('Greatest number is '
15                           ||a);
16      ELSIF b > a
17          AND b > c THEN
18
19      dbms_output.Put_line('Greatest number is '
20                           ||b);
21      ELSE
```

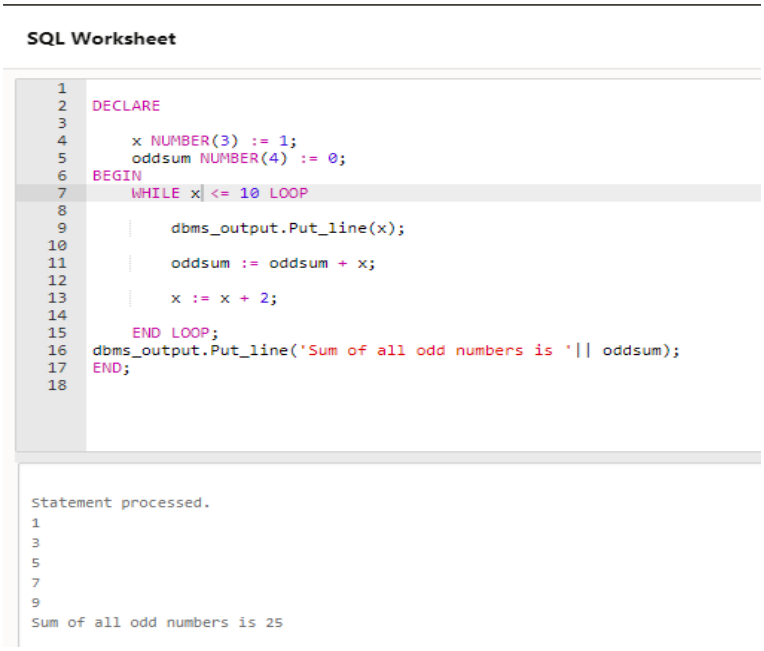
```
Statement processed.
Greatest number is 26
```

Experiment NO – 11

Aim: To write a PL/SQL block for summation of odd numbers using for LOOP

Implementation

```
DECLARE
        x NUMBER(3) := 1;
        oddsum NUMBER(4) := 0;
BEGIN
        WHILE x <= 10 LOOP
                dbms_output.Put_line(x);
                oddsum := oddsum + x;
                x := x + 2;
        END LOOP;
dbms_output.Put_line('Sum of all odd numbers is '|| oddsum);
END;
```



Experiment NO – 12

Aim: Describe about Procedures

Procedure is a subprogram unit that consists of a group of PL/SQL statements. Each procedure in Oracle has its own unique name by which it can be referred. This subprogram unit is stored as a database object. Below are the characteristics of this subprogram unit.

Note: Subprogram is nothing but a procedure, and it needs to be created manually as per the requirement. Once created they will be stored as database objects.

- Procedures are standalone blocks of a program that can be stored in the database.
- Call to these procedures can be made by referring to their name, to execute the PL/SQL statements.
- It is mainly used to execute a process in PL/SQL.
- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into the procedure or fetched from the procedure through parameters.
- These parameters should be included in the calling statement.
- Procedure can have a RETURN statement to return the control to the calling block, but it cannot return any values through the RETURN statement.
- Procedures cannot be called directly from SELECT statements. They can be called from another block or through EXEC keyword.

Syntax

```
CREATE OR REPLACE PROCEDURE
<procedure_name>
        (
        <parameterl IN/OUT <datatype>
        ..
        .
        )
[ IS | AS ]
        <declaration_part>
BEGIN
        <execution part>
EXCEPTION
        <exception handling part>
END;
```

- CREATE PROCEDURE instructs the compiler to create new procedure. Keyword 'OR REPLACE' instructs the compile to replace the existing procedure (if any) with the current one.
- Procedure name should be unique.
- Keyword 'IS' will be used, when the procedure is nested into some other blocks. If the procedure is standalone then 'AS' will be used. Other than this coding standard, both have the same meaning.

Example;

```
CREATE OR REPLACE PROCEDURE welcome_msg (p_name IN VARCHAR2)
IS
BEGIN
dbms_output.put_line (‘Welcome '|| p_name);
END;
/
EXEC welcome_msg (‘Guru99’);
```

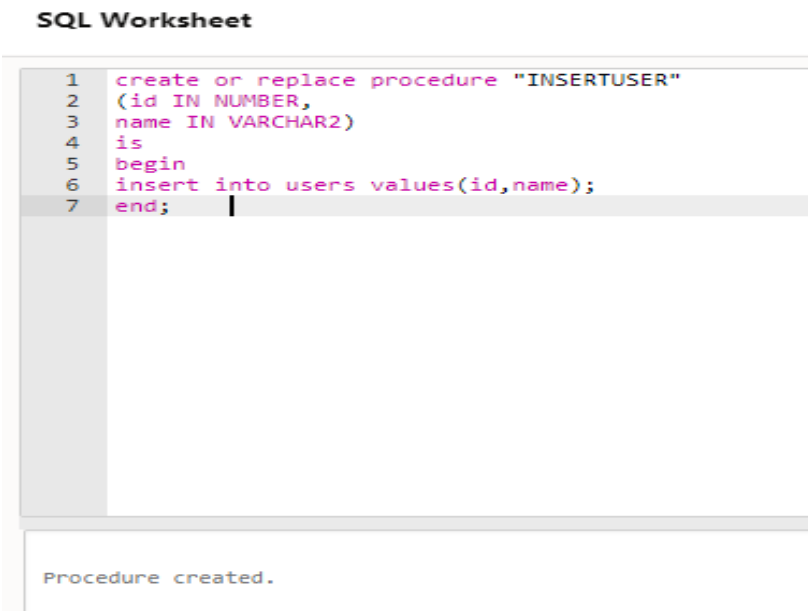
Experiment NO – 13

Aim: To write a PL/SQL Procedure using Positional Parameters

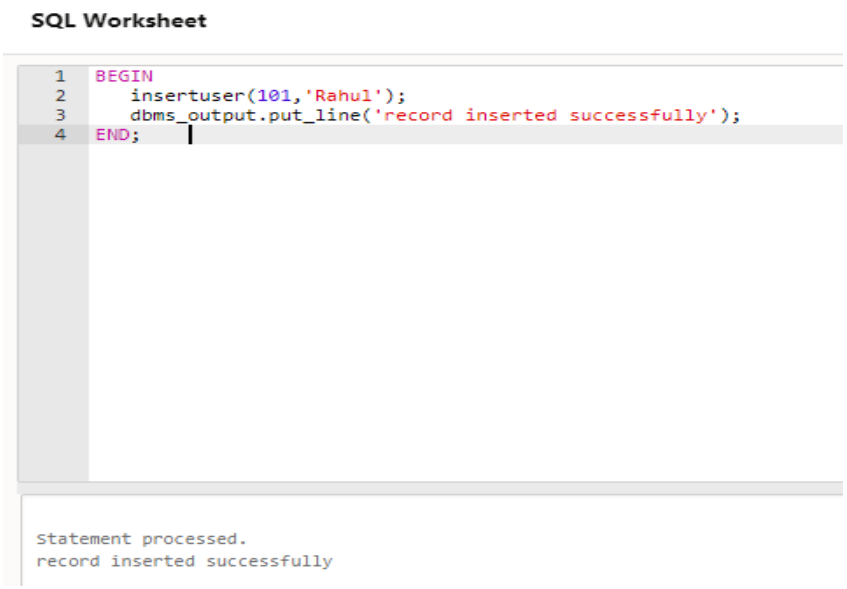
Implementation

```
CREATE OR REPLACE PROCEDURE CallMe(pA VARCHAR2,pB NUMBER,pC
BOOLEAN,pD DATE) AS
   BEGIN
     NULL;
  END CallMe;
```

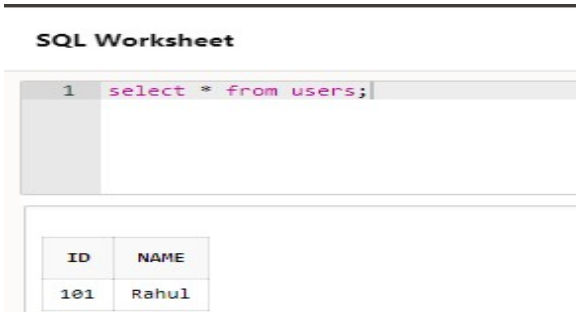
```
create table users (id number(10) primary key,name varchar2(100));
create or replace procedure "INSERTUSER"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into users values(id,name);
end;
```



```
BEGIN
   insertuser(101,'Rahul');
   dbms_output.put_line('record inserted successfully');
END;
```



Select * from users;

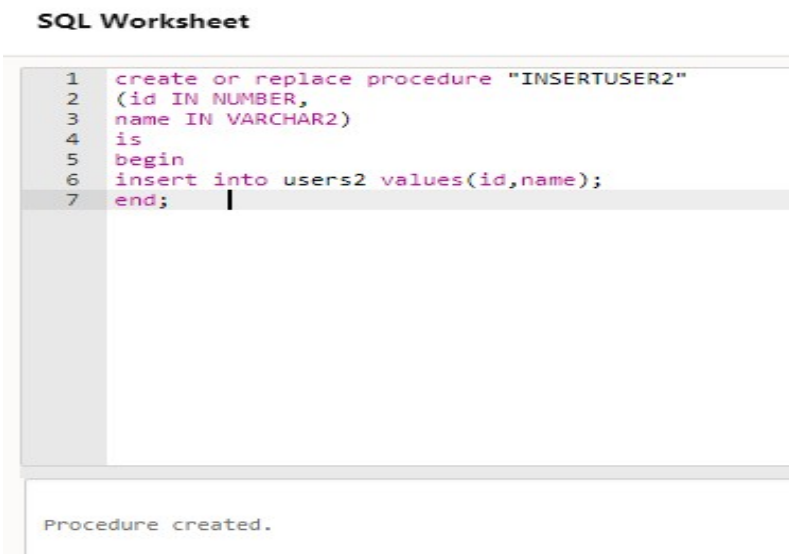


Experiment NO – 14

Aim: To write a PL/SQL Procedure using notational parameters

Implementation

```
create table users2 (id number(10) primary key,name varchar2(100));
create or replace procedure "INSERTUSER2"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into users2 values(id,name);
end;
```




```
BEGIN
   insertuser(id => 104,'Virat');
   dbms_output.put_line('record inserted successfully');
END;
```

SQL Worksheet

```
1  BEGIN
2     insertuser2(id => 104,name => 'Virat');
3     dbms_output.put_line('record inserted successfully');
4  END;
```

```
Statement processed.
record inserted successfully
```

Select * from users2;

SQL Worksheet

```
1  select * from users2;
```

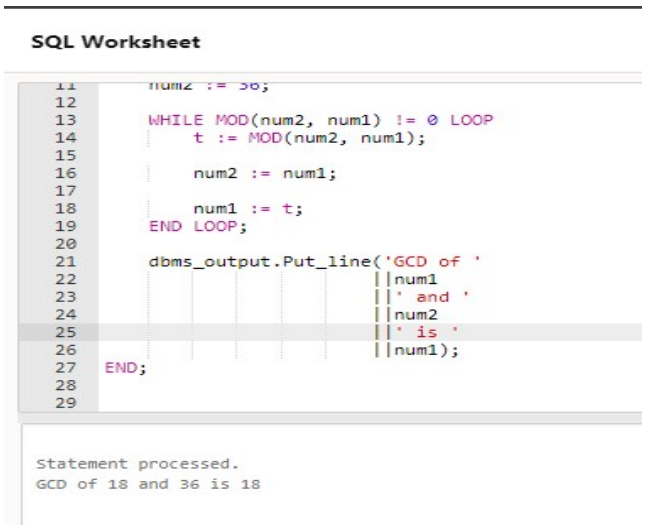
ID	NAME
104	Virat

Experiment NO – 15

Aim: To write a PL/SQL Procedure for GCD Numbers

Implementation

```
DECLARE
        num1 INTEGER;
        num2 INTEGER;
        t INTEGER;
BEGIN
        num1 := 18;
        num2 := 36;
        WHILE MOD(num2, num1) != 0 LOOP
                t := MOD(num2, num1);
                num2 := num1;
                num1 := t;
        END LOOP;
        dbms_output.Put_line('GCD of '||num1 ||' and '||num2 ||' is '||num1);
END;
```



Experiment NO – 16

Aim: To write a PL/SQL Procedure for cursor implementation

Implementation

```
CREATE TABLE students (
   studentID int primary key,
   LastName varchar(255) NULL,
   FirstName varchar(255) NOT NULL,
   Address varchar(255) ,
   City varchar(255) default 'Delhi',
   emailid varchar(255) NOT NULL UNIQUE,
   Age int CHECK (Age>=18)
);
INSERT All

INTO students (studentID, FirstName,LastName, Address,City,emailid,age)
VALUES
('2213', 'John','Doe','23/1 New Delhi', 'Delhi', 'John@gmail.com',20)
INTO students (studentID, FirstName,LastName, Address,City,emailid,age)
VALUES
('2214','Raj','Kumar','11/2 New Delhi','Delhi','raj@gmail.com',21)
INTO students (studentID, FirstName,LastName, Address,City,emailid,age)
VALUES
('2215','Ravi','Kumar','13/2 Borivli','Mumbai','ravi@gmail.com',21)
INTO students (studentID, FirstName,LastName, Address,City,emailid,age)
VALUES
('2216','Ajay','aditya','122 South Mumbai','Mumbai','aditya@gmail.com',22)
select * from dual;
```

```
DECLARE
  c_studentid students.studentid%type;
  c_FirstName students.FirstName%type;
  c_addr students.address%type;
  CURSOR c_students is
    SELECT studentid, FirstName, address FROM students;
BEGIN
  OPEN c_students;
  LOOP
  FETCH c_students into c_studentid, c_FirstName, c_addr;
    EXIT WHEN c_students%notfound;
    dbms_output.put_line(c_studentid || ' ' || c_FirstName || ' ' || c_addr);
  END LOOP;
  CLOSE c_students;
END;
```