

# INDEX

S.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO.	FACULTY SIGN
1		SQL Data Definition Language Commands on sample exercise		
2		SQL Data Manipulation Language Commands		
3		SQL Data Control Language Commands and Transaction control commands to the sample exercises		
4		Inbuilt functions in SQL on sample exercise		
5		Construct a ER Model for the application to be constructed to a Database		
6		Nested Queries on sample exercise		
7		Join Queries on sample exercise.		
8		Set Operators & Views.		
9		PL/SQL Conditional and Iterative Statements		
10		PL/SQL Procedures on sample exercise		
11		PL/SQL Functions		
12		PL/SQL Cursors		
13		PL/SQL Exception Handling		
14		PL/SQL Trigger		
15		Frame and execute the appropriate PL/SQL Cursors and Exceptional Handling for the project		

**EXNO :1**

## **DATA DEFINITION LANGUAGE COMMANDS**

**AIM:**

**PROCEDURE:**

### **Data Definition Commands:**

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 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.

### **1- i) CREATE TABLE**

It is used to create a table.

#### **Rules:**

1. Oracle reserved words cannot be used.
2. Underscore, numerals, letters are allowed but not blank space.
3. Maximum length for the table name is 30 characters.
4. Different tables should not have same name.
5. We should specify a unique column name.
6. We should specify proper data type along with width.

#### **Syntax:**

**SQL>**Create table tablename (column\_name1 data\_type constraints, column\_name2 data\_type constraints ...);

### **ii) DESC**

This is used to view the structure of the table.

**Syntax:**

**SQL>**desc tablename;

**iii) CREATING NEW TABLE FROM EXISTING TABLE:**

**Syntax:**

**CREATE TABLE** new\_table\_name **AS SELECT** column1, column2,... **FROM** existing\_table\_name **WHERE**.... ;

**2- DROP TABLE**

It will delete the table .

**Syntax:**

**SQL>**DROP TABLE <TABLENAME>;

**3- ALTER COMMAND**

Alter command is used to:

1. Add a new column.
2. Modify the existing column definition.
3. To include or drop integrity constraint.

**i) - ADD COMMAND**

Add the new column to the existing table.

**Syntax:**

**SQL>**alter table tablename add/modify (attribute datatype(size));

**ii) - MODIFY COMMAND**

Modify the existing column definition.

**Syntax :**

**SQL>**alter table <tablename> modify(columnname constraint);

**SQL>**alter table <tablename>modify(columnnamedatatype);

**4- TRUNCATE TABLE**

If there is no further use of records stored in a table and the structure has to be retained then the records alone can be deleted.

**Syntax:**

**SQL>**TRUNCATE TABLE <TABLE NAME>;

## **5- COMMENT**

Comments can be written in the following three formats:

1. Single line comments.
  2. Multi line comments
  3. In line comments
- **Single line comments:** Comments starting and ending in a single line are considered as single line comments.  
Line starting with ‘--’ is a comment and will not be executed.

### **Syntax:**

```
--single line comment  
--another comment
```

- **Multi line comments:** Comments starting in one line and ending in different line are considered as multi line comments. Line starting with ‘/\*’ is considered as starting point of comment and are terminated when ‘\*/’ is encountered.

### **Syntax:**

```
/* multi line comment  
another comment */
```

- **In line comments:** In line comments are an extension of multi line comments, comments can be stated in between the statements and are enclosed in between ‘/\*’ and ‘\*/’.

### **Syntax:**

```
SQL>SELECT * FROM /* table name; */
```

## **6- RENAME**

If you want to change the name of the table in the SQL database because they want to give a more relevant name to the table. Any database user can easily change the name by using the RENAME TABLE and ALTER TABLE statement in Structured Query Language.

### **Syntax:**

```
SQL>RENAME old_table _name To new_table_name ;
```

### **Program:**

```
SQL> connect
Enter user-name: system
Enter password: admin
Connected.
```

```
SQL> create table emp(id number(10),name varchar(10));
```

Table created.

```
SQL> desc emp;
```

Name	Null?	Type
-----		
ID		NUMBER(10)
NAME		VARCHAR2(10)

```
SQL> alter table emp add(dept varchar(10));
```

Table altered.

```
SQL> desc emp;
```

Name	Null?	Type
-----		
ID		NUMBER(10)
NAME		VARCHAR2(10)
DEPT		VARCHAR2(10)

```
SQL> alter table emp modify dept varchar(20);
```

Table altered.

```
SQL> desc emp;
```

Name	Null?	Type
-----		
ID		NUMBER(10)
NAME		VARCHAR2(10)
DEPT		VARCHAR2(20)

```
SQL> alter table emp drop column dept;
```

Table altered.

SQL> desc emp;

Name	Null?	Type
-----		
ID		NUMBER(10)
NAME		VARCHAR2(10)

SQL> alter table emp rename to emp1;

Table altered.

SQL> desc emp1;

Name	Null?	Type
-----		
ID		NUMBER(10)
NAME		VARCHAR2(10)

SQL> desc emp2;

Name	Null?	Type
-----		
ID		NUMBER(10)
NAME		VARCHAR2(10)
DEPT		VARCHAR2(10)

SQL> drop table emp2;

Table dropped.

SQL> select \* from emp2;

select \* from emp2  
\* ERROR

at line 1:

ORA-00942: table or view does not exist

SQL> select \* from emp1;

ID	NAME	DEPT
-----		
1	aaa	cse
2	aaa	cse
3	aaa	ece

4 aaa cse  
5 aaa cse

SQL> truncate table emp1;

Table truncated.

SQL> select \* from emp1;

no rows selected

SQL> desc emp1;

Name	Null?	Type
-----		
ID		NUMBER(10)
NAME		VARCHAR2(10)
DEPT		VARCHAR2(10)

SQL> drop table emp1;

Table dropped.

SQL> select \* from emp1;

select \* from emp1

\* ERROR

at line 1:

ORA-00942: table or view does not exist

SQL> desc emp1;

ERROR:

ORA-04043: object emp1 does not exist

**Result:**

**EX.NO:2**

**DATE:**

## **DATA MANIPULATION COMMANDS FOR INSERTING, DELETING, UPDATING AND RETRIEVING TABLES**

**AIM:**

**DESCRIPTION:**

**Data Manipulation Language:**

Data manipulation language (DML) statements access and manipulate data in existing tables. DML commands are the most frequently used SQL commands and is used to query and manipulate the existing database objects. Some of the commands are Insert, Select, Update, Delete.

**Examples of DML:**

- 1) **Insert Command:** This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined.
- 2) **Select Commands:** It is used to retrieve information from the table. It is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.
- 3) **Update Command:** It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.
- 4) **Delete command:** After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.

### **INSERTING VALUES INTO TABLE**

**Create table:**

```
SQL>Create table persons( pid number(5), firstnameVarChar(15),lastnameVarChar(15), address VarChar(25),city varchar(10));
```



```
SQL> Create table persons( pid number(5), firstname VarChar(15),lastname VarChar
(15), address VarChar(25),city varchar(10));
Table created.
SQL> desc persons;
```

Name	Null?	Type
PID		NUMBER(5)
FIRSTNAME		VARCHAR2(15)
LASTNAME		VARCHAR2(15)
ADDRESS		VARCHAR2(25)
CITY		VARCHAR2(10)

## **INSERT COMMAND**

Insert command is used to insert values into table.

### **Insert a single record into table.**

**Syntax:**SQL>insert into <table name> values (value list)

SQL>insert into persons values (001,'nelson','raj','no25,annai street','chennai');1  
row created.

```
SQL> insert into persons values (001,'nelson','raj','no25,annai street','chennai
');
1 row created.
SQL> /
1 row created.
```

### **Insert more than a record into persons table using a single insert command.**

SQL> insert into persons values(&pid,&firstname,&lastname,&address,&city);

```
SP2-0042: unknown command "poonamalle" - rest of line ignored.
SQL> insert into persons values(&pid,&firstname,&lastname,&address,&city
);
Enter value for pid: 002
Enter value for firstname: ram
Enter value for lastname: kumar
Enter value for address: no26,raja nagar
Enter value for city: avadi
old 1: insert into persons values(&pid,&firstname,&lastname,&address,&c
ity')
new 1: insert into persons values(002,'ram','kumar','no26,raja nagar','avadi')
1 row created.
```

```

SQL> /
Enter value for pid: 003
Enter value for firstname: divya
Enter value for lastname: shivani
Enter value for address: no27,pallavan nagar
Enter value for city: adyar
old 1: insert into persons values(&pid,'&firstname','&lastname','&address','&city')
new 1: insert into persons values(003,'divya','shivani','no27,pallavan nagar','adyar')
1 row created.

```

### Skipping the fields while inserting:

SQL> insert into persons(pid,firstname) values(500,'prabhu');

```

SQL> insert into persons(pid,firstname) values(500,'prabhu');
1 row created.

```

### SELECT COMMAND

It is used to retrieve information from the table. It is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.

#### **Syntax:**

SQL> Select \* from tablename; // This query selects all rows from the table.

#### **Example:**

SQL>Select \* from persons;

```
SQL> select *from persons;
```

PID	FIRSTNAME	LASTNAME	ADDRESS	CITY
1	nelson	raj	no25,annai street	chennai
1	nelson	raj	no25,annai street	chennai
2	ram	kumar	no26,raja nagar	avadi
3	divya	shivani	no27,pallavan nagar	adyar
500	prabhu			

### THE RETRIEVAL OF SPECIFIC COLUMNS FROM A TABLE:

It retrieves the specified columns from the table

**Syntax:** SQL>Select column\_name1, .....,column\_name N from table name;

**Example:** SQL>Select pid, firstname from persons;

```
SQL> Select pid, firstname from persons;

  PID FIRSTNAME
-----
    1  nelson
    1  nelson
    2   ram
    3  divya
   500 prabhu
```

### **Elimination of duplicates from the select clause:**

It prevents retrieving the duplicated values .Distinct keyword is to be used.

**Syntax:** SQL>Select DISTINCT col1, col2 from table name;

#### **Example:**

SQL>Select DISTINCT lastname from persons;

```
SQL> Select DISTINCT  lastname from persons;

LASTNAME
-----
kumar
raj
shivani
```

### **SELECT COMMAND WITH WHERE CLAUSE:**

To select specific rows from a table we include 'where' clause in the select command. It can appear only after the 'from' clause.

**Syntax:** SQL>Select column\_name1, .....,column\_name N from table name where condition;

**Example:** SQL>Select firstname, lastname from persons where pid>2;

```
SQL> Select firstname, lastname from persons where pid>2;

FIRSTNAME      LASTNAME
-----
divya          shivani
prabhu
```

### **Select command with order by clause:**

**Syntax:** SQL>Select column\_name1, .....,column\_namen from table name where condition  
orderbycolmnname;

#### **Example:**

SQL>Select firstname, lastname from persons order by pid;

```
SQL> Select firstname, lastname from persons order by pid;
```

FIRSTNAME	LASTNAME
nelson	raj
nelson	raj
ram	kumar
divya	shivani
prabhu	

### Select command to create a table:

#### Syntax:

```
SQL>create table tablename as select * from existing_tablename;
```

#### Example:

```
SQL>create table persons1 as select * from persons;  
Table created
```

### SELECT COMMAND TO INSERT RECORDS:

**Syntax:**SQL>insert into tablename( select columns from existing\_tablename);

**Example:**SQL>insert into persons1( select \* from persons);

PID	FIRSTNAME	LASTNAME	ADDRESS	CITY	PHONENO
001	nelson	raj	no25,annai street	Chennai	
100	niranjan	kumar	10/25 krishna street	Mumbai	999999999
102	arjun	kumar	30 sundaram street	coimbatore	
300	gugan	chand	5/10 mettu street	Coimbatore	
500	prabhu				

### SELECT COMMAND USING IN KEYWORD:

**Syntax:**SQL>Select column\_name1, .....,column\_namen from table name where colmnname IN (value1,value2);

**Example:** SQL>Select \* from persons where pid in (100,500);  
(OR)

```
SQL>Select * from persons where (pid=100 OR pid=500);
```

PID	FIRSTNAME	LASTNAME	ADDRESS	CITY	PHONENO
100	niranjan	kumar	10/25 krishna street	Mumbai	999999999
500	prabhu				

### **SELECT COMMAND USING BETWEEN KEYWORD:**

**Syntax:**SQL>Select column\_name1, .....,column\_namen from table name where colmnnameBETWEEN value1 AND value2;

**Example:** SQL>Select \* from persons where pid between 100 and 500;

PID	FIRSTNAME	LASTNAME	ADDRESS	CITY	PHONENO
100	niranjan	kumar	10/25 krishna street	Mumbai	999999999
500	prabhu				

### **SELECT COMMAND USING PATTERN:**

**Syntax:**SQL>Select column\_name1, .....,column\_namen from table name where colmnnameLIKE '% or \_';

**Example:** SQL>Select \* from persons where firstname like 'nir\_n%';

PID	FIRSTNAME	LASTNAME	ADDRESS	CITY	PHONENO
100	niranjan	kumar	10/25 krishna street	Mumbai	999999999

### **RENAMING THE FIELDNAME AT THE TIME OF DISPLAY USING SELECT STATEMENT:**

**Syntax:**SQL>Select old\_column\_namenew\_column\_namefrom table name where condition;

**Example:** SQL>Select pidpersonid from persons;

### **SELECT COMMAND TO RETRIEVE NULL VALUES:**

**Syntax:**SQL>Select column\_name from table name where column\_name is NULL ;

**Example:** SQL>Select \* from persons where lastname is null;

PID	FIRSTNAME	LASTNAME	ADDRESS	CITY	PHONENO
500	prabhu				

## **UPDATE COMMAND:**

### **Syntax:**

UPDATE table\_name SET column\_name = value [, column\_name = value]...[  
WHERE condition ];

**Example:**SQL>update persons set pid= 5 where firstname='prabhu';

Table updated.

## **DELETE COMMAND**

**Syntax:** SQL>Delete from table where conditions;

**Example:**SQL>delete from persons where pid=500;1

row deleted.

## **RESULT:**

**EX.NO:3**

**DATE:**

## **DATA CONTROL LANGUAGE COMMANDS AND TRANSACTION CONTROL COMMANDS**

**AIM:**

**DESCRIPTION:**

### **Transaction Control statements**

**Transaction Control Language (TCL)** commands are used to manage transactions in the database. These are used to manage the changes made by DML-statements. It also allows statements to be grouped together into logical transactions.

#### **Examples of TCL:**

- (i) Commit
- (ii) Rollback
- (iii) Savepoint

**(i) Commit:** Commit command saves all the work done.

**Syntax:** commit;

```
SQL> create table Regions(reg_no number(5), region_name varchar(25));
Table created.
SQL> insert into Regions values(101,'Chennai');
1 row created.
SQL> insert into Regions values(102,'Madurai');
1 row created.
SQL> insert into Regions values(103,'Covai');
1 row created.
```

```
SQL> select * from Regions;
```

REG_NO	REGION_NAME
101	Chennai
102	Madurai
103	Covai

```
SQL> commit;
```

```
Commit complete.
```

(ii) **Rollback:** Rollback Command restores database to original since the last Commit.

**Syntax:** ROLLBACK TO SAVEPOINT <savepoint\_name>;

```
SQL> select * from Regions;
```

REG_NO	REGION_NAME
101	Chennai
102	Madurai
103	Covai

```
SQL> update Regions set region_name='Aarani' where region_name='Madurai';
```

```
1 row updated.
```

```
SQL> select * from Regions;
```

REG_NO	REGION_NAME
101	Chennai
102	Aarani
103	Covai

```
SQL> rollback;
```

```
Rollback complete.
```

```
SQL> select * from Regions;
```

REG_NO	REGION_NAME
101	Chennai
102	Madurai
103	Covai



### (iii) Savepoint:

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

**Syntax:**SAVEPOINT <savepoint\_name>;

```
SQL> INSERT INTO Regions VALUES(150, 'Nellai');
1 row created.

SQL> savepoint AA;
Savepoint created.

SQL> INSERT INTO Regions VALUES(151, 'Pondy');
1 row created.

SQL> savepoint BB;
Savepoint created.

SQL> INSERT INTO Regions VALUES(152, 'Ponneri');
1 row created.

SQL> savepoint CC;
Savepoint created.

SQL> select * from Regions order by reg_no;

  REG_NO REGION_NAME
-----
    101 Chennai
    103 Covai
    105 Bang
    106 Tanjore
    108 Covai
    120 Madurai
    150 Nellai
    151 Pondy
    152 Ponneri

9 rows selected.

SQL> rollback to BB;
Rollback complete.
```

```
SQL> rollback to BB;

Rollback complete.

SQL> select * from Regions order by reg_no;

  REG_NO REGION_NAME
-----
    101 Chennai
    103 Covai
    105 Bang
    106 Tanjore
    108 Covai
    120 Madurai
    150 Nellai
    151 Pondy

8 rows selected.
```

## Data Control Language

Data Control Language (DCL) is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges. Privileges are of two types,

- **System:** This includes permissions for creating session, table, etc and all types of other system privileges.
- **Object:** This includes permissions for any command or query to perform any operation on the database tables.

In DCL we have two commands,

- **GRANT:** Used to provide any user access privileges or other privileges for the database.
- **REVOKE:** Used to take back permissions from any user.

```
SQL> GRANT CREATE table TO system;
Grant succeeded.

SQL> REVOKE CREATE TABLE FROM system;
Revoke succeeded.

SQL> GRANT CREATE SESSION TO system;
Grant succeeded.

SQL> GRANT CREATE table TO system;
Grant succeeded.

SQL> ALTER USER system QUOTA UNLIMITED ON users;
User altered.
```

```
SQL> REVOKE CREATE TABLE FROM system;
Revoke succeeded.

SQL>
```

**RESULT:**

**EX.NO:4**

## **SQL FUNCTIONS**

**AIM**

### **DESCRIPTION:**

#### **What are functions?**

Functions are methods used to perform data operations. SQL has many in-built functions used to perform string concatenations, mathematical calculations etc.

SQL functions are categorized into the following two categories:

1. Aggregate Functions
2. Scalar Functions

Let us look into each one of them, one by one.

### **AGGREGATE SQL FUNCTIONS**

The Aggregate Functions in SQL perform calculations on a group of values and then return a single value. Following are a few of the most commonly used Aggregate Functions:

<b>Function</b>	<b>Description</b>
SUM()	Used to return the sum of a group of values.
COUNT()	Returns the number of rows either based on a condition, or without a condition.
AVG()	Used to calculate the average value of a numeric column.
MIN()	This function returns the minimum value of a column.
MAX()	Returns a maximum value of a column.
FIRST()	Used to return the first value of the column.
LAST()	This function returns the last value of the column.

### **SCALAR SQL FUNCTIONS**

The Scalar Functions in SQL are used to return a single value from the given input value. Following are a few of the most commonly used Aggregate Functions:

<b>Function</b>	<b>Description</b>
LCASE()	Used to convert string column values to lowercase
UCASE()	This function is used to convert a string column values to Uppercase.
LEN()	Returns the length of the text values in the column.

MID()	Extracts substrings in SQL from column values having String data type.
ROUND()	Rounds off a numeric value to the nearest integer.
NOW()	This function is used to return the current system date and time.
FORMAT()	Used to format how a field must be displayed.

### **EXAMPLE:**

### **CHARACTER/STRING FUNCTION:**

SQL> select upper('welcome') from dual;

SQL> select upper('hai') from dual; SQL>

select lower('HAI') from dual;

SQL> select initcap('hello world') from dual; SQL>

select ltrim('hello world','hell') from dual; SQL>

select rtrim('hello world','ld')from dual; SQL>

select concat('SRM',' University')from dual;SQL>

select length('Welcome') from dual;

SQL> select replace('SRM University', 'University','IST')from dual;SQL>

select lpad('SRM University',20,'\*')from dual;

SQL> select rpad('SRM University',15,'\$')from dual;

SQL> select substr('Welcome to SRM University', 4,7)from dual;

SQL> select replace('COMPUTER','O','AB')from dual;

SQL> select replace('University','city','Inter')from dual;

SQL> select translate('cold','ld','ol')from dual

## OUTPUT:

```
SQL> select upper('welcome') from dual;

UPPER('
-----
WELCOME

SQL> select upper('hai') from dual;

UPP
---
HAI

SQL> select lower('HAI') from dual;

LOW
---
hai

SQL> select initcap('hello world') from dual;

INITCAP('HE
-----
Hello World

SQL> select ltrim('hello world','hell') from dual;

LTRIM('
-----
o world

SQL> select rtrim('helloworld','ld') from dual;

RTRIM('H
-----
hellowor

SQL> select concat('SRM','University') from dual;

CONCAT('SRM',
-----
SRMUniversity

SQL> select length('Welcome')from dual;

LENGTH('WELCOME')
-----
7
```

```
SQL> select replace('SRM University','University','IST')from dual;
```

```
REPLACE
```

```
-----
```

```
SRM IST
```

```
SQL> select lpad('SRMUniversity',20,'*')from dual;
```

```
LPAD('SRMUNIVERSITY'
```

```
-----
```

```
*****SRMUniversity
```

```
SQL> select rpad('SRMUniversity',15,'$')from dual;
```

```
RPAD('SRMUNIVER
```

```
-----
```

```
SRMUniversity$$
```

```
SQL> select substr('Welcome to SRM University',4,7)from dual;
```

```
SUBSTR(
```

```
-----
```

```
come to
```

```
SQL> SELECT REPLACE('COMPUTER','O','AB') from dual;
```

```
REPLACE('
```

```
-----
```

```
CABMPUTER
```

```
SQL> SELECT REPLACE('University','Univer','Inter') from dual;
```

```
REPLACE('
```

```
-----
```

```
Intersity
```

```
SQL> SELECT REPLACE('University','city','Inter') from dual;
```

```
REPLACE('U
```

```
-----
```

```
University
```

```
SQL> select translate('cold','ld','ol')from dual;
```

```
TRAN
```

```
----
```

```
cool
```

```
SQL> select translate('Welcome','eo','ag')from dual;
```

```
TRANSLA
```

```
-----
```

```
Walcgma
```

## DATE & TIME FUNCTION

```
SQL> select sysdate from dual;
SQL> select round(sysdate)from dual;
SQL> select add_months(sysdate,3)from dual;
SQL> select last_day(sysdate)from dual; SQL>
select sysdate+20 from dual;
SQL> select next_day(sysdate,'tuesday')from dual;
```

### OUTPUT:

```
SQL> select sysdate from dual;
SYSDATE
-----
22-FEB-22

SQL> select round(sysdate) from dual;
ROUND(SYS
-----
23-FEB-22

SQL> select add_months(sysdate,5) from dual;
ADD_MONTH
-----
22-JUL-22

SQL> select last_day(sysdate) from dual;
LAST_DAY(
-----
28-FEB-22

SQL> select sysdate+15 from dual;
SYSDATE+1
-----
09-MAR-22

SQL> select next_day(sysdate,'Monday') from dual;
NEXT_DAY(
-----
28-FEB-22
```



## NUMERIC FUNCTION

```
SQL> select round(15.6789)from dual;  
SQL> select ceil(23.20)from dual; SQL>  
select floor(34.56)from dual; SQL>  
select trunc(15.56743)from dual;SQL>  
select sign(-345)from dual;  
SQL> select abs(-70)from dual;
```

### OUTPUT:

```
SQL> select round(15.6789) from dual;  
ROUND(15.6789)  
-----  
16  
  
SQL> select ceil(23.20) from dual;  
CEIL(23.20)  
-----  
24  
  
SQL> select floor(34.56) from dual;  
FLOOR(34.56)  
-----  
34  
  
SQL> select trunc(15.56743) from dual;  
TRUNC(15.56743)  
-----  
15  
  
SQL> select sign(-345) from dual;  
SIGN(-345)  
-----  
-1  
  
SQL> select abs(-70) from dual;  
ABS(-70)  
-----  
70
```

### **MATHFUNCTION:**

SQL> select power(10,12) from dual;

SQL> select power(5,6) from dual;

SQL> select mod(11,5) from dual;

SQL> select exp(10) from dual; SQL>

select sqrt(225) from dual;

```
SQL> select power(10,12) from dual;
```

```
POWER(10,12)
```

```
-----  
1.0000E+12
```

```
SQL> select power(5,6) from dual;
```

```
POWER(5,6)
```

```
-----  
15625
```

```
SQL> select mod(11,5) from dual;
```

```
MOD(11,5)
```

```
-----  
1
```

```
SQL> select exp(10) from dual;
```

```
EXP(10)
```

```
-----  
22026.4658
```

```
SQL> select sqrt(225) from dual;
```

```
SQRT(225)
```

```
-----  
15
```

### **RESULT:**

**Ex.No:5**

**CONSTRUCT A ER MODEL FOR THE APPLICATION TO BE CONSTRUCTED TO A DATABASE**  
**AIM:**

**HOW TO DRAW AN ENTITY RELATIONSHIP DIAGRAM?:**

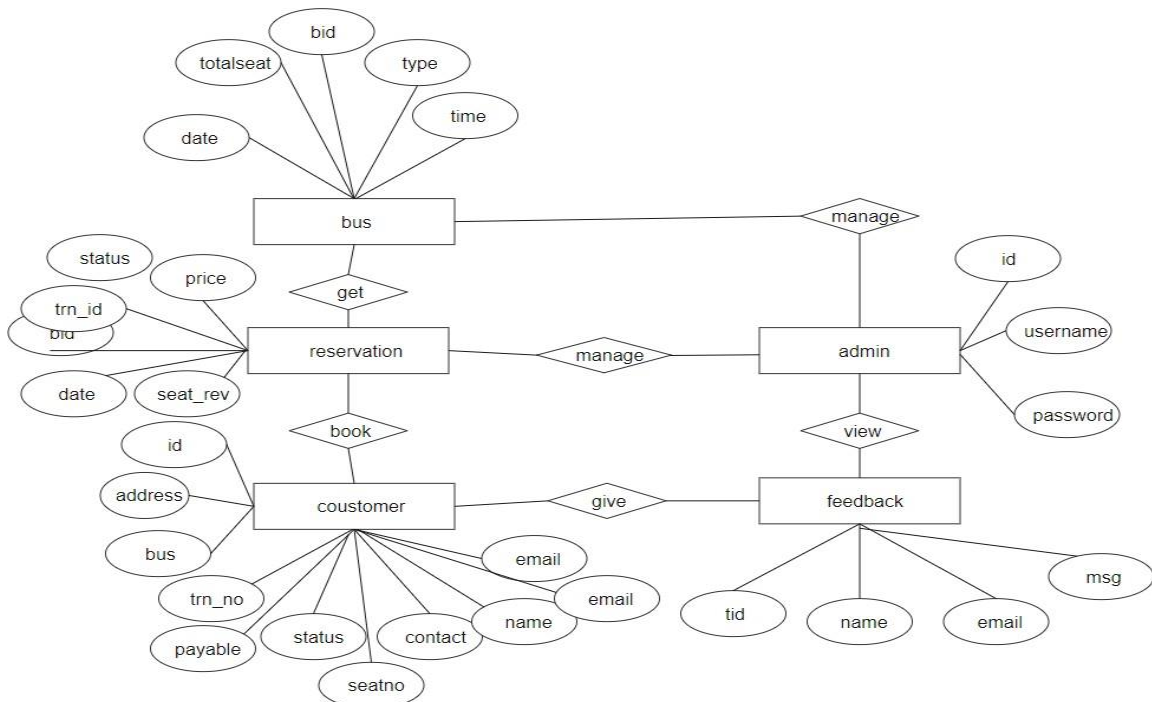
1. Determine the Entities in Your ERD.
2. Add Attributes to Each Entity.
3. Define the Relationships Between Entities.
4. Add Cardinality to Every Relationship in your ER Diagram.
5. Finish and Save Your ERD.

**Few Online Entity Relationship Diagram Tools**

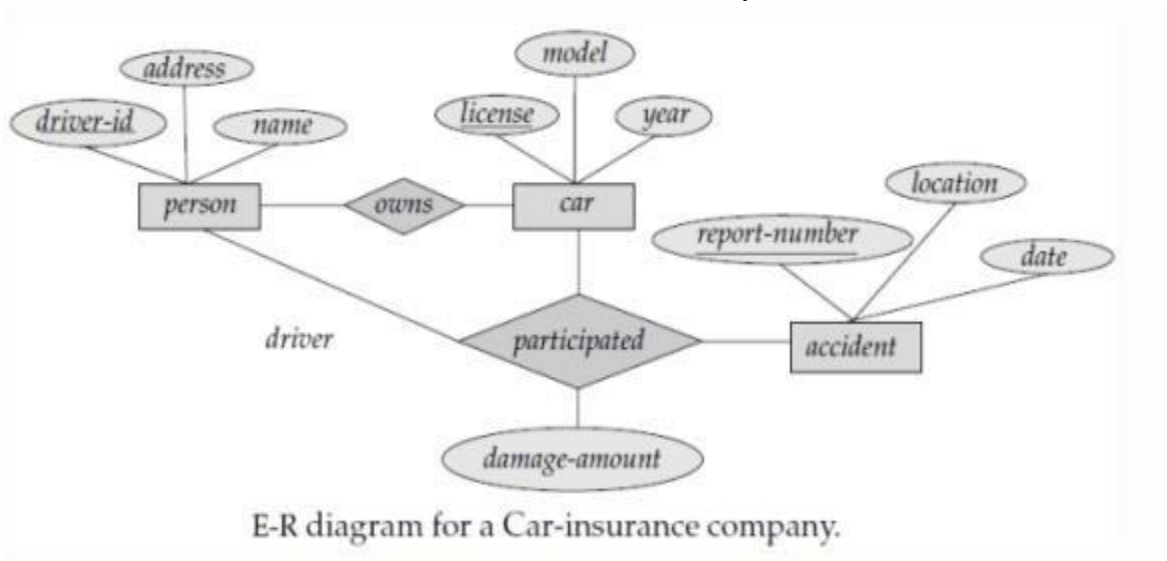
1. Vertabelo
2. Creatly
3. ERDPlus
4. Lucidchart
5. Visual Paradigm Online
6. Draw.io
7. Microsoft Visio
8. Gliffy
9. SqlDBM ER Diagram Online Tool
10. ER Draw Max

S.No.	Problem Statement
1	<p>(a) Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.</p> <p><b>ER Diagram:</b>  <b>Answer :</b></p> <p style="text-align: center;">E-R diagram for a hospital.</p> <p>(b) Construct appropriate tables for the above ER Diagram :</p> <p><b>Hospital tables:</b>  patients (patient-id, name, insurance, date-admitted, date-checked-out)  doctors (doctor-id, name, specialization)  test (testid, testname, date, time, result)  doctor-patient (patient-id, doctor-id)  test-log (testid, patient-id)  performed-by (testid, doctor-id)</p>
2	<p>Design an E-R diagram for keeping track of the exploits of your favourite sports team. You should store the matches played, the scores in each match, the players in each match and individual player statistics for each match. Summary statistics should be modeled as derived attributes</p> <p><b>ER Diagram:</b></p> <p style="text-align: center;">E-R diagram for favourite team statistics.</p>
3	Design an E-R diagram for bus reservation system.

# BUS RESERVATION



- (a) Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.



- (b) Construct appropriate tables for the above ER Diagram?

**Car insurance tables:**

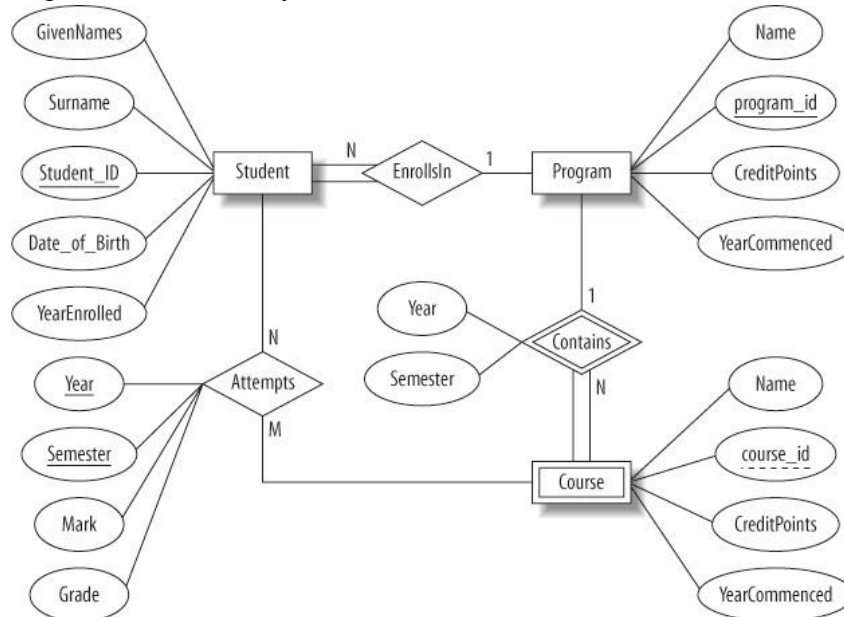
person (driver-id, name,  
address)car (license, year,  
model)

accident (report-number, date, location)

participated(driver-id, license, report-number, damage-amount)

Design an E-R diagram for University Database

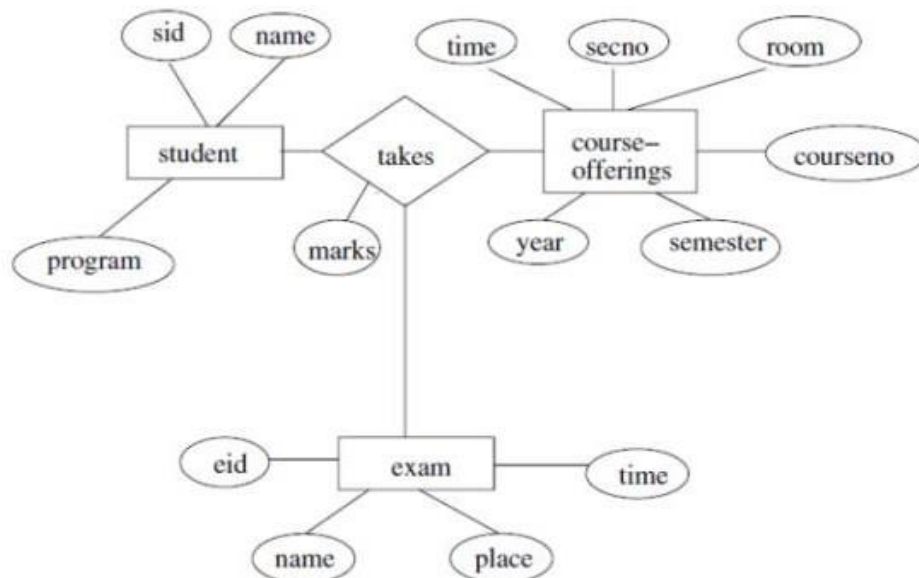
5



Consider a database used to record the marks that students get in different exams of different course offerings.

a) Construct an E-R diagram that models exams as entities, and uses a ternary relationship, for the above database.

6



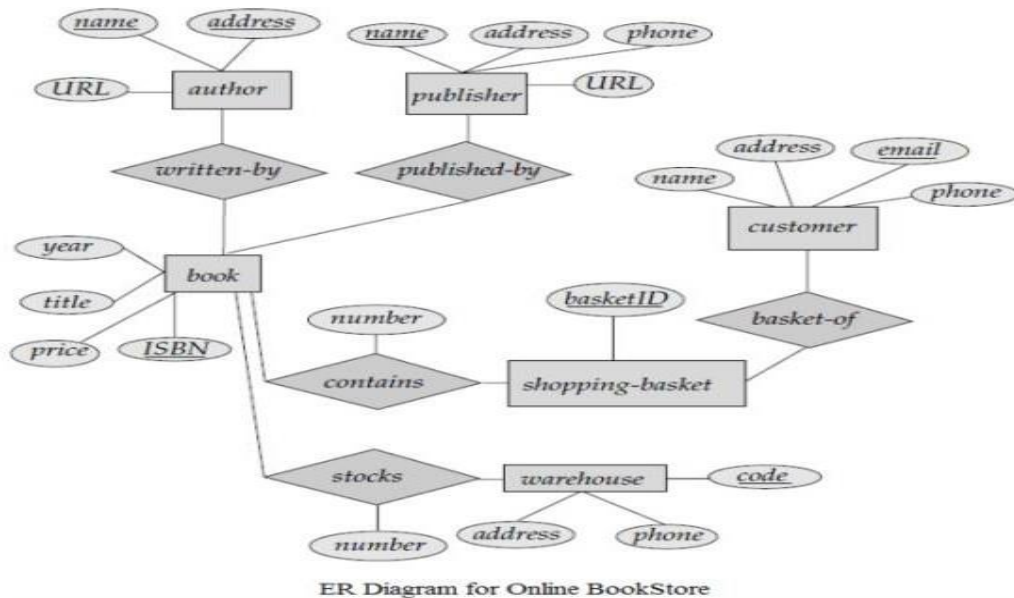
E-R diagram for marks database.

- b) Construct an alternative E-R diagram that uses only a binary relationship between students and course-offerings. Make sure that only one relationship exists between a particular student and course-offering pair, yet you can represent the marks that a student gets in different exams of a course offering.

**ER Diagram:**



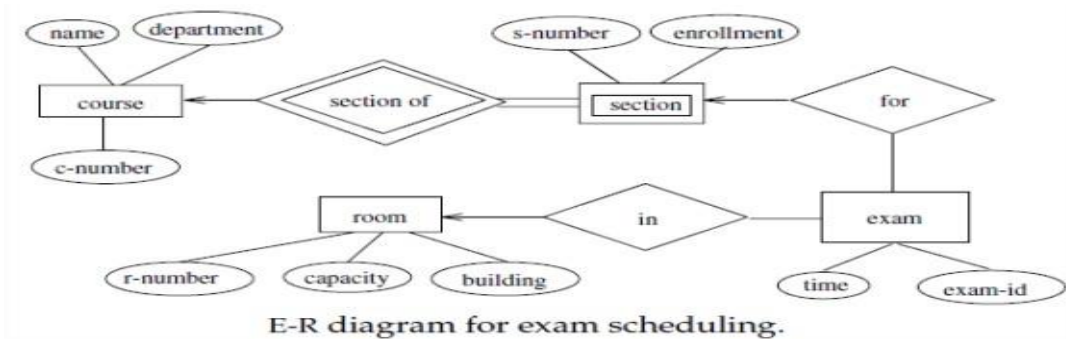
- 7 Draw the E-R diagram which model

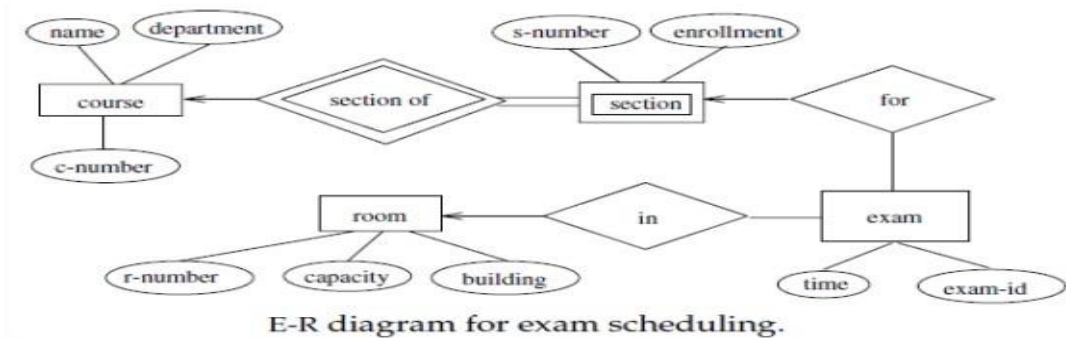


online bookstore.

s an

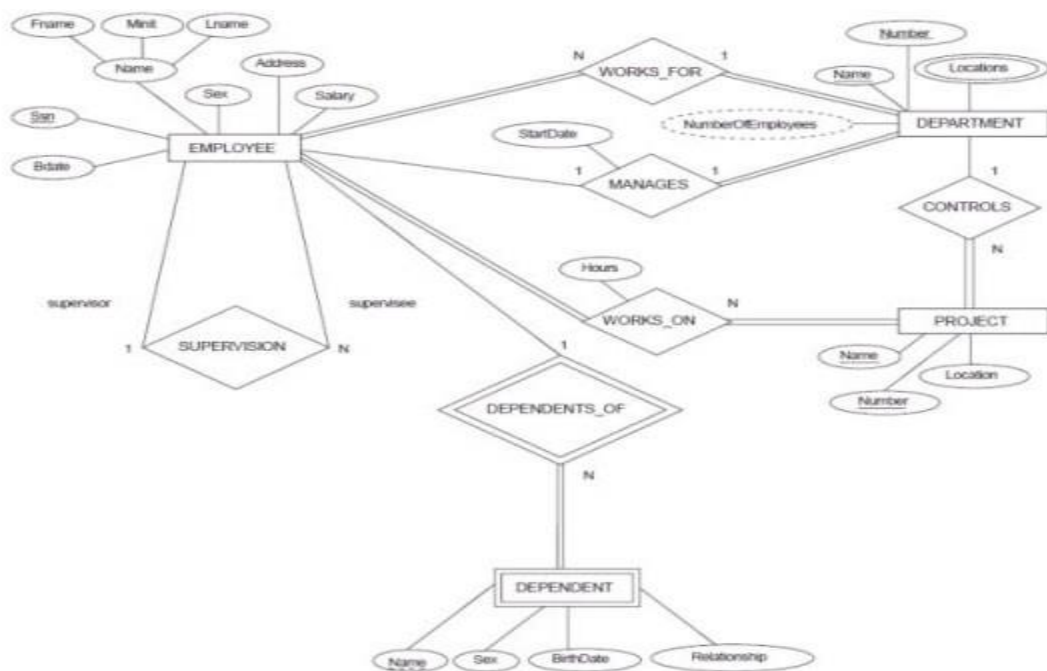
8	<p>Consider a university database for the scheduling of classrooms for -final exams. This database could be modeled as the single entity set exam, with attributes course-name, sectionnumber, room-number, and time. Alternatively, one or more additional entity sets could be defined, alongwith relationship sets to replace some of the attributes of the exam entity set, as</p> <ul style="list-style-type: none"> <li>• course with attributes name, department, and c-number</li> <li>• section with attributes s-number and enrollment, and dependent as a weak entity set on course</li> <li>• room with attributes r-number, capacity, and building</li> </ul> <p>Show an E-R diagram illustrating the use of all three additional entity sets listed.</p>



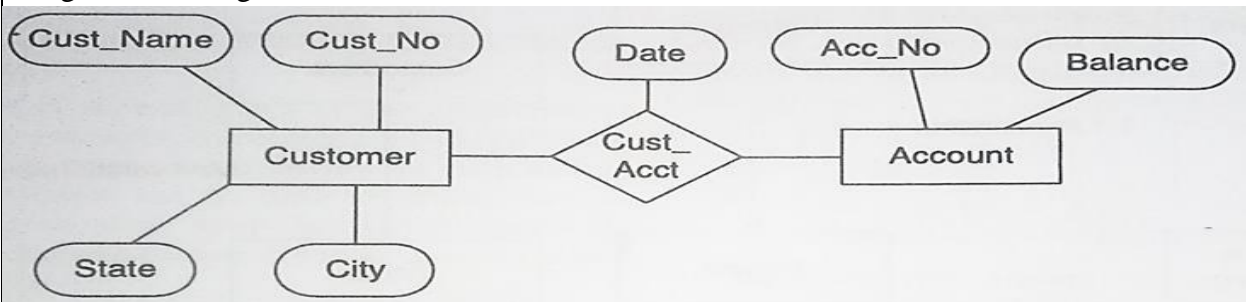
	 <p>E-R diagram for exam scheduling.</p>
9	<p>Construct an ER Diagram for Company having following details :</p> <ul style="list-style-type: none"> <li>• Company organized into DEPARTMENT. Each department has unique name and a particular employee who manages the department. Start date for the manager is recorded. Department may have several locations.</li> <li>• A department controls a number of PROJECT. Projects have a unique name, number and a single location.</li> <li>• Company's EMPLOYEE name, ssno, address, salary, sex and birth date are recorded. An employee is assigned to one department, but may work for several projects (not necessarily controlled by her dept). Number of hours/week an employee works on each project is recorded; The immediate supervisor for the employee.</li> <li>• Employee's DEPENDENT are tracked for health insurance purposes (dependent name,</li> </ul>



birthdate, relationship to employee).



Design an E-R diagram for Customer Account.



10

**EX.NO:6**

## **NESTED QUERIES**

**AIM:**

**DESCRIPTION:**

Nested query is one of the most useful functionalities of SQL. Nested queries are useful when we want to write complex queries where one query uses the result from another query. Nested queries will have multiple **SELECT** statements nested together. A **SELECT** statement nested within another **SELECT** statement is called a subquery.

### **What is a Nested Query in SQL?**

A nested query in SQL contains a query inside another query. The result of the inner query will be used by the outer query. For instance, a nested query can have two **SELECT** statements, one on the inner query and the other on the outer query.

### **What are the Types of Nested Queries in SQL?**

Nested queries in SQL can be classified into two different types:

1. Independent Nested Queries
2. Co-related Nested Queries

#### **1. Independent Nested Queries**

In independent nested queries, the execution order is from the innermost query to the outer query. An outer query won't be executed until its inner query completes its execution. The result of the inner query is used by the outer query. Operators such as **IN**, **NOT IN**, **ALL**, and **ANY** are used to write independent nested queries.

The **IN** operator checks if a column value in the outer query's result is **present** in the inner query's result. The final result will have rows that satisfy the **IN** condition.

The **NOT IN** operator checks if a column value in the outer query's result is **not present** in the inner query's result. The final result will have rows that satisfy the **NOT IN** condition.

The **ALL** operator compares a value of the outer query's result with **all the values** of the inner query's result and returns the row if it matches all the values.

The **ANY** operator compares a value of the outer query's result with all the inner query's result values and returns the row if there is a match with **any value**.

## 2. Co-related Nested Queries

In co-related nested queries, the inner query uses the values from the outer query so that the inner query is executed for every row processed by the outer query. The co-related nested queries run slowly because the inner query is executed for every row of the outer query's result.

### How to Write Nested Query in SQL?

We can write a nested query in SQL by nesting a **SELECT** statement within another **SELECT** statement. The outer **SELECT** statement uses the result of the inner **SELECT** statement for processing.

The general syntax of nested queries will be:

```
SELECT column_name [, column_name ]
FROM table1 [, table2 ]
WHERE column_name OPERATOR
( SELECT column_name [, column_name ]
  FROM table1 [, table2 ]
  [WHERE]
)
```

The **SELECT** query inside the brackets (()) is the inner query, and the **SELECT** query outside the brackets is the outer query. The result of the inner query is used by the outer query.

### EXAMPLE:

#### TABLE #1 - employeedata

```
SQL> CREATE TABLE employeedata(id NUMBER PRIMARY KEY, name VARCHAR2(25) NOT NULL, salary NUMBER NOT NULL, role VARCHAR2(15) NOT NULL);
```

Table created.

```
SQL> INSERT INTO employeedata VALUES (1, 'Augustine Hammond', 10000, 'Developer');
```

1 row created.

```
SQL> INSERT INTO employeedata VALUES (2, 'Perice John', 10000, 'Manager');
```

1 row created.

```
SQL> INSERT INTO employeedata VALUES (3, 'Ragu Delafoy', 30000, 'Developer');
```

1 row created.

```
SQL> INSERT INTO employeeedata VALUES (4, 'Teakwood Saffen', 40000, 'Manager');
```

1 row created.

```
SQL> INSERT INTO employeeedata VALUES (5, 'Freddy Malcom', 50000, 'Developer');
```

1 row created.

```
SQL> select * from employeeedata;
```

### OUTPUT:

ID	NAME	SALARY	ROLE
1	Augustine Hammond	10000	Developer
2	Perice John	10000	Manager
3	Ragu Delafoy	30000	Developer
4	Teakwood Saffen	40000	Manager
5	Freddy Malcom	50000	Developer

### TABLE #2 - awards

```
SQL>CREATE TABLE awards( id NUMBER PRIMARY KEY, employee_id NUMBER NOT  
NULL, award_date DATE NOT NULL );
```

Table created.

```
SQL> INSERT INTO awards VALUES(1, 1, TO_DATE('2022-04-01', 'YYYY-MM-DD'));
```

1 row created.

```
SQL> INSERT INTO awards VALUES(2, 3, TO_DATE('2022-05-01', 'YYYY-MM-DD'));
```

1 row created.

```
SQL> select * from awards;
```

### OUTPUT:

ID	EMPLOYEE_ID	AWARD_DAT
1	1	01-APR-22

## Independent Nested Queries

### Example 1: IN

- Select all employees who won an award.

```
SQL> SELECT id, name FROM employee_data WHERE id IN (SELECT employee_id FROM awards);
```

#### OUTPUT:

ID	NAME
1	Augustine Hammond
3	Ragu Delafoey

### Example 2: NOT IN

- Select all employees who never won an award.

```
SQL> SELECT id, name FROM employee_data WHERE id NOT IN (SELECT employee_id FROM awards);
```

#### OUTPUT:

ID	NAME
2	Perice John
4	Teakwood Saffen
5	Freddy Malcom

### Example 3: ALL

- Select all Developers who earn more than all the Managers

```
SQL> SELECT * FROM employee_data WHERE role = 'Developer' AND salary > ALL (SELECT salary FROM employee_data WHERE role = 'Manager');
```

#### OUTPUT:

ID	NAME	SALARY	ROLE
----	------	--------	------

5 Freddy Malcom	50000	Developer
-----------------	-------	-----------

#### Example 4: ANY

- Select all Developers who earn more than any Manager

*SQL> SELECT \* FROM employeeedata WHERE role = 'Developer' AND salary > ANY (SELECT salary FROM employeeedata WHERE role = 'Manager');*

#### OUTPUT:

ID	NAME	SALARY	ROLE
3	Ragu Delafoy	30000	Developer
5	Freddy Malcom	50000	Developer

#### Co-related Nested Queries

- Select all employees whose salary is above the average salary of employees in their role.

#### Example:

*SQL> SELECT \* FROM employeeedata emp1 WHERE salary > (SELECT AVG(salary) FROM employeeedata emp2 WHERE emp1.role = emp2.role);*

#### OUTPUT:

ID	NAME	SALARY	ROLE
4	Teakwood Saffen	40000	Manager
5	Freddy Malcom	50000	Developer

#### Explanation

*The manager with id 4 earns more than the average salary of all managers (25000), and the developer with id 5 earns more than the average salary of all developers (30000). The inner*

*query is executed for all rows fetched by the outer query. The role value (emp1.role) of every outer query's row is used by the inner query (emp1.role = emp2.role).*

- We can find the average salary of managers and developers using the below query:

*SQL> SELECT role, AVG(salary) FROM employeedata GROUP BY role;*

**OUTPUT:**

ROLE	AVG(SALARY)
------	-------------

Developer	30000
Manager	25000

**RESULT:**

**AIM****PROCEDURE:**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

**Different Types of SQL JOINS**

Here are the different types of the JOINS in SQL:

1. **(INNER) JOIN: Returns records that have matching values in both tables**

*Syntax:*

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

2. **LEFT JOIN: Returns all records from the 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;
```

**EXAMPLE:**

**CREATE A TABLE PRODUCTORDERS**

```
SQL> CREATE table productorders(Order_Id number(5),Orderno number(5),P_Id number(3));
```

Table created.

```
SQL> desc productorders;
```

Name	Null?	Type
-----		
ORDER_ID		NUMBER(5)
ORDERNO		NUMBER(5)
P_ID		NUMBER(3)

**INSERTING VALUES INTO THE TABLE PRODUCTORDERS**

```
SQL> INSERT into productorders values(&Order_Id,&Orderno,&P_Id);
```

Enter value for order\_id: 1 Enter

value for orderno: 77895Enter

value for p\_id: 3

old 1: INSERT into productorders values(&*Order\_Id*,&*Orderno*,&*P\_Id*)new

1: INSERT into productorders values(1,77895,3)

1 row created.

```
SQL> INSERT into productorders values(&Order_Id,&Orderno,&P_Id);
```

Enter value for order\_id: 2

Enter value for orderno: 44678

Enter value for p\_id: 3

old 1: INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id)new

1: INSERT into productorders values(2,44678,3)

1 row created.

*SQL> INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id);*

Enter value for order\_id: 3 Enter

value for orderno: 22456Enter

value for p\_id: 1

old 1: INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id)new

1: INSERT into productorders values(3,22456,1)

*1 row created.*

*SQL> INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id);*

Enter value for order\_id: 4 Enter

value for orderno: 24562Enter

value for p\_id: 1

old 1: INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id)new

1: INSERT into productorders values(4,24562,1)

*1 row created.*

*SQL> INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id);*

Enter value for order\_id: 5 Enter

value for orderno: 34764Enter

value for p\_id: 15

old 1: INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id)new

1: INSERT into productorders values(5,34764,15)

*1 row created.*

## DISPLAYING DATA FROM TABLE PRODUCTORDERS

SQL> select \* from productorders;

ORDER_ID	ORDERNO	P_ID
-----	-----	-----
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

## CREATE A SECOND TABLE PERSON

SQL> CREATE table person(p\_Id number(5),LASTNAME varchar2(10),Firstname varchar2(15),  
Address varchar2(20),city varchar2(10));

Table created.

## INSERTING VALUES INTO THE TABLE PERSON

SQL> INSERT into person values(&p\_Id,&Lastname','&firstname','&Address','&city');

Enter value for p\_id: 1

Enter value for lastname: Smith Enter

value for firstname: Jadon Enter value

for address: RamapuramEnter value

for city: Chennai

old 1: INSERT into person values(&p\_Id,&Lastname','&firstname','&Address','&city')new

1: INSERT into person values(1,'Smith','Jadon','Ramapuram','Chennai')

1 row created.

SQL> INSERT into person values(&p\_Id,&Lastname','&firstname','&Address','&city');

Enter value for p\_id: 2

Enter value for lastname: Hemal Enter

value for firstname: Elango Enter

value for address: Anna NagarEnter

value for city: Tamilnadu

old 1: INSERT into person values(&p\_Id,'&Lastname','&firstname','&Address','&city')new

1: INSERT into person values(2,'Hemal','Elango','Anna Nagar','Tamilnadu')

*1 row created.*

*SQL> INSERT into person values(&p\_Id,'&Lastname','&firstname','&Address','&city');*

Enter value for p\_id: 3

Enter value for lastname: Lanser

Enter value for firstname: Kim Enter

value for address: HyderabadEnter

value for city: AP

old 1: INSERT into person values(&p\_Id,'&Lastname','&firstname','&Address','&city')new

1: INSERT into person values(3,'Lanser','Kim','Hyderabad','AP')

*1 row created.*

## **DISPLAYING DATA FROM TABLE PERSON**

*SQL> select \*from person;*

P_ID	LASTNAME	FIRSTNAME	ADDRESS	CITY
1	Smith	Jadon	Ramapuram	Chennai
2	Hemal	Elango	Anna Nagar	Tamilnadu
3	Lanser	Kim	Hyderabad	AP

## #1 INNER JOIN

### OUTPUT

```
SQL> SELECT person.firstname, person.city FROM person INNER JOIN productorders ON person.p_Id = productorders.p_Id;
```

FIRSTNAME	CITY
-----------	------

-----	-----
-------	-------

Kim	AP
-----	----

Kim	AP
-----	----

Jadon	Chennai
-------	---------

Jadon	Chennai
-------	---------

## #2 LEFT JOIN

### OUTPUT

```
SQL> SELECT person.lastname, person.firstname, productorders.orderno FROM person LEFT JOIN productorders ON person.p_Id = productorders.p_Id ORDER BY person.lastname;
```

LASTNAME	FIRSTNAME	ORDERNO
----------	-----------	---------

-----	-----	-----
-------	-------	-------

Hemal	Elango	
-------	--------	--

Lanser	Kim	77895
--------	-----	-------

Lanser	Kim	44678
--------	-----	-------

Smith	Jadon	24562
-------	-------	-------

Smith	Jadon	22456
-------	-------	-------

### #3 RIGHT (OUTER) JOIN

#### OUTPUT

*SQL> SELECT person.lastname, person.firstname, productorders.orderno FROM person RIGHT OUTER JOIN productorders ON person.p\_Id = productorders.p\_Id ORDER BY person.firstname;*

LASTNAME	FIRSTNAME	ORDERNO
-----	-----	-----
Smith	Jadon	22456
Smith	Jadon	24562
Lanser	Kim	77895
Lanser	Kim	44678
		34764

### #4 FULL OUTER JOIN

#### OUTPUT

*SQL> SELECT person.lastname, person.address FROM person FULL OUTER JOIN productorders ON person.p\_Id = productorders.p\_Id ORDER BY person.firstname;*

LASTNAME	ADDRESS
-----	-----
Hemal	Anna Nagar
Smith	Ramapuram
Smith	Ramapuram
Lanser	Hyderabad
Lanser	Hyderabad

*6 rows selected*

**RESULT:**

**Ex.No:8**

**Date:**

## **SET OPERATORS AND VIEWS**

**AIM:**

**PROCEDURE:**

**SQL SET OPERATION:**

The SQL Set operation is used to combine the two or more SQL SELECT statements.

### **Types of Set Operation**

1. Union
2. UnionAll
3. Intersect
4. Minus

#### **1. Union**

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

**Syntax:**

*SELECT column\_name FROM table1 UNION SELECT column\_name FROM table2;*

#### **2. Union All**

- Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

**Syntax:**

*SELECT column\_name FROM table1 UNION ALL SELECT column\_name FROM table2;*

### 3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

#### Syntax:

*SELECT column\_name FROM table1 INTERSECT SELECT column\_name FROM table2;*

### 4. Minus

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

#### Syntax:

*SELECT column\_name FROM table1 MINUS SELECT column\_name FROM table2;*

#### Output:

Creating two tables namely stud1 and stud2.

```
SQL> select * from stud1;
```

SID	SNAME	SAGE
1001	bavana	21
1002	Levin	19
1003	prabu	29
1004	kumar	22
1001	Abi	12

```
SQL> select * from stud2;
```

SID	SNAME	SAGE
1001	Abi	12
1002	Fahad	23
1005	Kavin	24
1006	Dinesh	28



```
SQL> select * from stud1 union select * from stud2;
```

SID	SNAME	SAGE
1001	Abi	12
1001	bavana	21
1002	Fahad	23
1002	Levin	19
1003	prabu	29
1004	kumar	22
1005	Kavin	24
1006	Dinesh	28

```
8 rows selected.
```

```
SQL> select * from stud1 union all select * from stud2;
```

SID	SNAME	SAGE
1001	bavana	21
1002	Levin	19
1003	prabu	29
1004	kumar	22
1001	Abi	12
1001	Abi	12
1002	Fahad	23
1005	Kavin	24
1006	Dinesh	28

```
9 rows selected.
```

```
SQL> select * from stud1 intersect select * from stud2;
```

SID	SNAME	SAGE
1001	Abi	12

```
SQL> select * from stud1 minus select * from stud2;
```

SID	SNAME	SAGE
1001	bavana	21
1002	Levin	19
1003	prabu	29
1004	kumar	22

## SQL VIEW

SQL provides the concept of VIEW, which hides the complexity of the data and restricts unnecessary access to the database.

It permits the users to access only a particular column rather than the whole data of the table.

The **View** in the Structured Query Language is considered as the virtual table, which depends on the result-set of the predefined SQL statement.

### Create a SQL View

To create a View in Structured Query Language by using the CREATE VIEW statement. Creation of View from a *single table or multiple tables*.

### Syntax to Create View from Single Table

```
CREATE VIEW View_Name AS  
SELECT Column_Name1, Column_Name2,....., Column_NameN  
FROM Table_Name  
WHERE condition;
```

### Example to Create a View from Single table

```
SQL> create view Student_View as select sid,sname,sage from stud1 where sage>20;  
View created.  
SQL> select * from Student_View;
```

SID	SNAME	SAGE
1001	bavana	21
1003	prabu	29
1004	kumar	22

### Syntax to Create View from Multiple Tables

To create a View from multiple tables by including the tables in the SELECT statement.

```

CREATE VIEW View_Name AS
SELECT Table_Name1.Column_Name1, Table_Name1.Column_Name2,
        Table_Name2.Column_Name2, ....., Table_NameN.Column_NameN
FROM Table_Name1, Table_Name2, ....., Table_NameN
WHERE condition;

```

### Example to Create a View from Multiple tables

```
SQL> select * from stud1;
```

SID	SNAME	SAGE	STUD_SUBJECT
1001	bavana	21	
1002	Levin	19	
1003	prabu	29	
1004	kumar	22	
1001	Abi	12	
1005	Mani	18	Physics
1006	Hari	21	English
1007	Lawrence	20	Computer

8 rows selected.

```
SQL> select * from teacher;
```

TID	TNAME	TSUBJECT	TCITY
2001	Thiya	Maths	Chennai
2002	Tharun	Chemistry	Mumbai
2003	Guru	CSE	Delhi
2004	Andrew	Physics	Noida

```
SQL> create view Student_Teacher_View as select stud1.sid,stud1.sname,teacher.tid,teacher.tcity
from stud1,teacher where stud1.stud_subject=teacher.tsubject;
```

View created.

```
SQL> select * from Student_Teacher_View;
```

SID	SNAME	TID	TCITY
1005	Mani	2004	Noida

### Update an SQL View

*A view in SQL can only be modified if the view follows the following conditions:*

1. You can update that view which depends on only one table. SQL will not allow updating the view which is created more than one table.
2. The fields of view should not contain NULL values.
3. The view does not contain any subquery and DISTINCT keyword in its definition.
4. The views cannot be updatable if the SELECT statement used to create a View contains JOIN or HAVING or GROUP BY clause.
5. If any field of view contains any SQL aggregate function, you cannot modify the view.

### Syntax to Update a View

```
CREATE OR REPLACE VIEW View_Name AS
SELECT Column_Name1, Column_Name2,....., Column_NameN
FROM Table_Name
WHERE condition;
```

```
SQL> create or replace view teacher_view as select tid,tname,tsubject from teacher where tsubject='CSE';
View created.

SQL> select * from teacher_view;
```

TID	TNAME	TSUBJECT
2003	Guru	CSE

### Insert the new row into the existing view

Just like the insertion process of database tables, we can also insert the record in the views. The following SQL INSERT statement is used to insert the new row or record in the view:

#### Syntax:

```
INSERT INTO View_Name(Column_Name1, Column_Name2 , Column_Name3, ....., Column_NameN) VALUES(value1, value2, value3,..... , valueN);
```

```
SQL> select * from student_view;
```

SID	SNAME	SAGE
1001	bavana	21
1003	prabu	29
1004	kumar	22
1006	Hari	21

```
SQL> insert into student_view(sid,sname,sage)values(1007,'Jack',21);
```

```
1 row created.
```

```
SQL> select * from student_view;
```

SID	SNAME	SAGE
1001	bavana	21
1003	prabu	29
1004	kumar	22
1006	Hari	21
1007	Jack	21

### Delete the existing row from the view

Just like the deletion process of database tables, we can also delete the record from the views. The following SQL DELETE statement is used to delete the existing row or record from the view:

#### Syntax

***DELETE FROM View\_Name WHERE Condition;***

```
SQL> delete student_view where sage=29;
```

```
1 row deleted.
```

```
SQL> select * from student_view;
```

SID	SNAME	SAGE
1001	bavana	21
1004	kumar	22
1006	Hari	21
1007	Jack	21

### Drop a View

To delete the existing view from the database if it is no longer needed the following SQL DROP statement is used to delete the view:

#### Syntax:

***DROP VIEW View\_Name;***

```
SQL> drop view teacher_view;

View dropped.

SQL> select * from teacher_view;
select * from teacher_view
      *
ERROR at line 1:
ORA-00942: table or view does not exist
```

#### RESULT:

## **Experiment: 9**

### **PL/SQL Conditional and Iterative**

#### **AIM:**

**PROCEDURE:** To write & execute PL/SQL

\* As we want output of PL/SQL on screen, before starting anything type-:

SQL> SET serveroutput ON.

\* Write program save, execute & exit.

#### Basic Syntax of PL/SQL

DECLARE

/\* Variables can be declared here \*/

BEGIN

/\* Executable statements can be written here \*/

EXCEPTION;

/\* Error Handles can be written here \*/

END;

#### Decision Making with If Statements:

Syntax:

If (test\_condition) then

Statements ..... (set of)

Else

Statements ..... (set of)

End if;

For nested if-else statement we can use if -else if -else as follows:

If (test\_condition) then

Set of Statements

Elseif (condition)

Set of Statements

End if;

### LOOPING STATEMENTS-

\* For executing the set of statements repeatedly we use loops

\* SQL supports looping statements Like GOTO, FOR, WHILE & LOOP.

GOTO- << LABEL>>

Set of Statements

GOTO Label;

FOR LOOP- For <var> in [Reverse] <Int value>...<End-value>

Set of Statements

End Loop;

WHILE LOOP- While (Condition) Loop;

Set of Statements;

End Loop;



**Program:**

SQL> Set serveroutput on

SQL> declare

2 a integer:=10;

3 begin

4 dbms\_output.put\_line(a);

5 end;

6 /

10

PL/SQL procedure successfully completed.

```
SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> Set serveroutput on
SQL> declare
  2 a integer:=10;
  3 begin
  4 dbms_output.put_line(a);
  5 end;
  6 /
10

PL/SQL procedure successfully completed.
```

SQL> declare

2 a varchar(10):='hello';

3 begin

4 dbms\_output.put\_line(a);

5 end;

6 /

hello

PL/SQL procedure successfully completed.

```
SQL> declare
  2  a varchar(10):='hello';
  3  begin
  4  dbms_output.put_line(a);
  5  end;
  6  /
hello

PL/SQL procedure successfully completed.
```

SQL> declare

2 a integer:=10;

3 begin

4 dbms\_output.put\_line('value of a'||a);

5 end;

6 /

value of a:10

PL/SQL procedure successfully completed.

```
SQL> declare
  2  a integer:=10;
  3  begin
  4  dbms_output.put_line('value of a:'||a);
  5  end;
  6  /
value of a:10

PL/SQL procedure successfully completed.
```

SQL> declare

```
  2  pi constant number:=3.14;
  3  begin
  4  dbms_output.put_line(pi);
  5  end;
  6  /
3.14
```

PL/SQL procedure successfully completed.

```
SQL> declare
  2  pi constant number:=3.14;
  3  begin
  4  dbms_output.put_line(pi);
  5  end;
  6  /
3.14

PL/SQL procedure successfully completed.
```

SQL> declare

```
  2  a integer:=10;
  3  b integer:=20;
```

```
4 c integer;
5 begin
6 c:=a+b;
7 dbms_output.put_line(c);
8 end;
9 /
30
```

PL/SQL procedure successfully completed.

```
SQL> declare
  2 a integer:=10;
  3 b integer:=20;
  4 c integer;
  5 begin
  6 c:=a+b;
  7 dbms_output.put_line(c);
  8 end;
  9 /
30
```

PL/SQL procedure successfully completed.

## IF-ELSE

```
SQL> declare
  2 a integer:=30;
  3 begin
  4 if(a<20) then
  5 dbms_output.put_line('a is less than 20');
  6 else
  7 dbms_output.put_line('a is not less than 20');
```

```
8 end if;
9 end;
10 /
a is not less than 20
```

PL/SQL procedure successfully completed.

```
SQL> declare
  2  a integer:=30;
  3  begin
  4  if(a<20) then
  5  dbms_output.put_line('a is less than 20');
  6  else
  7  dbms_output.put_line('a is not less than 20');
  8  end if;
  9  end;
 10  /
a is not less than 20

PL/SQL procedure successfully completed.
```

## CASE STATEMENT

SQL> DECLARE

```
2  grade char(1) := 'A';
3  BEGIN
4  CASE grade
5  when 'A' then dbms_output.put_line('Excellent');
6  when 'B' then dbms_output.put_line('Very good');
7  when 'C' then dbms_output.put_line('Good');
8  when 'D' then dbms_output.put_line('Average');
9  when 'F' then dbms_output.put_line('Passed with Grace');
10  else dbms_output.put_line('Failed');
```

11 END CASE;

12 END;

13 /

Excellent

PL/SQL procedure successfully completed.

```
SQL> DECLARE
  2 grade char(1) := 'A';
  3 begin
  4 CASE grade
  5   when 'A' then dbms_output.put_line('Excellent');
  6   when 'B' then dbms_output.put_line('Very good');
  7   when 'C' then dbms_output.put_line('Good');
  8   when 'D' then dbms_output.put_line('Average');
  9   when 'F' then dbms_output.put_line('Passed with Grace');
 10   else dbms_output.put_line('Failed');
 11 END CASE;
 12 END;
 13 /
Excellent

PL/SQL procedure successfully completed.
```

## LOOP

SQL> DECLARE

2 i NUMBER := 1;

3 BEGIN

4 LOOP

5 EXIT WHEN i>10;

6 DBMS\_OUTPUT.PUT\_LINE(i);

7 i := i+1;

8 END LOOP;

9 END;

10 /

1

2

3

4

5

6

7

8

9

10

PL/SQL procedure successfully completed.

```
SQL> DECLARE
  2  i NUMBER := 1;
  3  BEGIN
  4  LOOP
  5  EXIT WHEN i>10;
  6  DBMS_OUTPUT.PUT_LINE(i);
  7  i := i+1;
  8  END LOOP;
  9  END;
10  /
```

```
1
2
3
4
5
6
7
8
9
10
```

PL/SQL procedure successfully completed.

## **WHILE LOOP**

SQL> DECLARE

2 i INTEGER := 1;

3 BEGIN

4 WHILE i <= 10 LOOP

5 DBMS\_OUTPUT.PUT\_LINE(i);

6 i := i+1;

7 END LOOP;

8 END;

9 /

1

2

3

4

5

6

7

8

9

10

PL/SQL procedure successfully completed.



```
SQL> DECLARE
  2  i INTEGER := 1;
  3  BEGIN
  4  WHILE i <= 10 LOOP
  5  DBMS_OUTPUT.PUT_LINE(i);
  6  i := i+1;
  7  END LOOP;
  8  END;
  9  /
1
2
3
4
5
6
7
8
9
10
```

## FOR LOOP

```
SQL> DECLARE
  2  VAR1 NUMBER;
  3  BEGIN
  4  VAR1:=10;
  5  FOR VAR2 IN 1..10
  6  LOOP
  7  DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
  8  END LOOP;
  9  END;
10 /
10
20
30
```

40  
50  
60  
70  
80  
90  
100

PL/SQL procedure successfully completed.

```
SQL> DECLARE
  2  VAR1 NUMBER;
  3  BEGIN
  4  VAR1:=10;
  5  FOR VAR2 IN 1..10
  6  LOOP
  7  DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
  8  END LOOP;
  9  END;
 10  /
10
20
30
40
50
60
70
80
90
100

PL/SQL procedure successfully completed.
```

**Result:**

## **Experiment: 10**

### **PL/SQL Procedures on sample exercise**

#### **AIM:**

#### **Procedure: (PL/SQL)**

```
DECLARE

/* Variables can be declared here */

BEGIN

/* Executable statments can the written here*/

EXCEPTION

/* Error handlers can be written here"*/

END;
```

#### **SYNTAX:**

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
    < procedure_body >
END procedure_name;
```

#### **Program:**

```
SQL> create table proc(id number(10),name varchar(10));
```

Table created.

```
SQL> create or replace procedure "INSERTUSER"
```

```
(id IN NUMBER,
```

```
name IN VARCHAR2)
```

```
is
```

```
begin
```

```
insert into proc values(id,name);
```

```
end;
```

```
/
```

```
Procedure created.
```

```
SQL> BEGIN
```

```
insertuser(101,'Rahul');
```

```
dbms_output.put_line('record inserted successfully');
```

```
END;
```

```
/
```

```
record inserted successfully
```

```
PL/SQL procedure successfully completed.
```

```
SQL> select * from proc;
```

```
ID NAME
```

```
-----
```

```
101 Rahul
```

**OUTPUT:**

**Result:**

## **Experiment: 11**

### **PL/SQL Functions**

#### **AIM:**

#### **PROCEDURE:**

##### Syntax of PL/SQL:-

```
DECLARE /* Variables can be declared here*/  
BEGIN /*Executable statements can be written here*/  
EXCEPTION /*Error handlers can be written her*/  
END;
```

##### Syntax for functions:-

```
CREATE [OR REPLACE] PROCEDURE procedure_name  
(parameter,parameter)  
IS/AS  
[declaration section]  
BEGIN  
[executable section]  
EXCEPTION  
[exception section]  
END[function_name];
```

#### **Program:**

```
SQL> create or replace function adder(n1 in number, n2 in number)  
2 return number  
3 is
```

```
4 n3 number(8);
5 begin
6 n3 :=n1+n2;
7 return n3;
8 end;
9 /
```

Function created.

SQL> DECLARE

```
2 n3 number(2);
3 BEGIN
4 n3 := adder(11,22);
5 dbms_output.put_line('Addition is: ' || n3);
6 END;
7 /
```

Addition is: 33

PL/SQL procedure successfully completed.

SQL> 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
16  RETURN z;
17 END;
18 BEGIN
19   a:= 23;
20   b:= 45;
21
22   c := findMax(a, b);
23   dbms_output.put_line(' Maximum of (23,45): ' || c);
24 END;
25 /

```

Maximum of (23,45): 45

PL/SQL procedure successfully completed.

SQL> select \* from emp;

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------



```
-----
      30 sai      26 mumbai      150000
40 sam      27 hyderabad      200000
      50 tom      29 pune      40630
```

SQL> create or replace function totalemp

2 return number is

3 total number(10):=0;

4 begin

5 select count(\*) into total from emp;

6 return total;

7 end;

8 /

Function created.

SQL> declare

2 c number(20);

3 begin

4 c:=totalemp();

5 dbms\_output.put\_line('Total no of emp:'||c);

6 end;

7 /

Total no of emp:3

PL/SQL procedure successfully completed.

SQL> DECLARE

2 num number;

3 factorial number;

4

5 FUNCTION fact(x number)

6 RETURN number

7 IS

8 f number;

9 BEGIN

10 IF x=0 THEN

11 f := 1;

12 ELSE

13 f := x \* fact(x-1);

14 END IF;

15 RETURN f;

16 END;

17

18 BEGIN

19 num:= 6;

20 factorial := fact(num);

21 dbms\_output.put\_line(' Factorial ' || num || ' is ' || factorial);

22 END;

23 /

Factorial 6 is 720

PL/SQL procedure successfully completed.

**OUTPUT:**

```
Function created.
```

```
Statement processed.  
Addition is: 33
```

---

```
Statement processed.  
Maximum of (23,45): 45
```

```
Statement processed.  
Total no of emp:3
```

---

```
Statement processed.  
Factorial 6 is 720
```

**Result:**

## **Experiment: 12**

### **PL/SQL Cursors**

**Aim:**

**Theory:**

**Cursor:**

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

Types of cursors:

1. Implicit Cursors
2. Explicit Cursors

**Implicit Cursors:**

Automatically created by Oracle whenever SQL statement is created and executed, no explicit cursor is found then.

Programmers cannot control the implicit cursors and information in it.

Attributes in IC are % found, % is OPEN, % IS NOTFOUND and % ROWCOUNT SQL cursors has additional attributes such as % Bulk\_rowcount and % Bulk\_Exceptions , to use with for all statements.

**Explicit Cursors:**

Explicit cursors are programmer defined cursors for gaining more control over context area.

It should be defined in declaration section of the PL/SQL block.

It is created on select statement which returns more than one row.

**Syntax:**

```
CURSOR cursor_name IS select_statement;
```

**Program:**

```
SQL>DECLARE
total_rows number(2);
BEGIN
UPDATE employee1
SET salary = salary + 5000;
IF sql%notfound THEN
dbms_output.put_line('no customers updated'); ELSIF
sql%found THEN total_rows:= sql%rowcount;
dbms_output.put_line( total_rows || 'customers updated');
END IF;
END;
/
3 customers updated
```

PL/SQL procedure successfully completed.

```
SQL> select* from employee1;
```

ID	NAME	AGE	ADDRESS	SALARY
30	sai	26	mumbai	155000
40	sam	27	hyderabad	205000
50	tom	29	pune	45630

## **EXPLICIT CURSOR**

```
SQL> DECLARE
c_id employee1.id%type;
c_name employee1.name%type;
c_addr employee1.address%type;
CURSOR c_employee1 is
SELECT id,name,address FROM employee1;
begin open c_employee1; loop
FETCH c_employee1 into c_id,c_name,c_addr;
EXIT WHEN
c_employee1%notfound;
  dbms_output.put_line(c_id||' '||
c_name||' '||c_addr);
END LOOP;
CLOSE c_emp;
END;
/
```

PL/SQL procedure successfully completed.

## **OUTPUT:**



## Experiment: 13

### PL/SQL Exception Handling

#### AIM:

#### Program:

```
SQL> DECLARE
```

```
c_id emp.id%type:=10;
```

```
c_name emp.name%type;
```

```
c_addr emp.address%type;
```

```
begin
```

```
SELECT name, address into c_name,c_addr FROM emp where id=c_id;
```

```
dbms_output.put_line('Name'||c_name);
```

```
dbms_output.put_line('Address'||c_addr);
```

```
exception
```

```
when no_data_found then dbms_output.put_line('no such customer');
```

```
when others then dbms_output.put_line('error');
```

```
end;
```

```
/
```

```
no such customer
```

PL/SQL procedure successfully completed.

```
SQL> declare
```

```
c_id emp.id%type:=30;
```

```
c_name emp.name%type;
```



```

c_addr emp.address%type;

begin

SELECT name, address into c_name,c_addr FROM emp where id=c_id;

dbms_output.put_line('Name:'||c_name);

dbms_output.put_line('Address:'||c_addr);

exception

when no_data_found then dbms_output.put_line('no such customer');

when others then dbms_output.put_line('error');

end;

```

/

Name:sai

Address:mumbai

PL/SQL procedure successfully completed.

## **USER DEFINED**

SQL> declare

```
c_id emp.id%type:=&id;
```

```
c_name emp.name%type;
```

```
c_addr emp.address%type;
```

```
ex_invalid_id exception;
```

```
begin
```

```
if c_id<=0 then
```

```
raise ex_invalid_id;
```

```
else
```

```
select name,address into c_name,c_addr from emp where id=c_id;
dbms_output.put_line('name:'||c_name);
dbms_output.put_line('address:'||c_addr);
end if;
exception
when ex_invalid_id then dbms_output.put_line('id should be greater than zero');
when no_data_found then dbms_output.put_line('no such customer');
when others then dbms_output.put_line('error');
end;
/
```

Enter value for id: -6

```
old 2: c_id emp.id%type:=&id;
new 2: c_id emp.id%type:=-6;
id should be greater than zero
```

PL/SQL procedure successfully completed.

SQL> /

Enter value for id: 30

```
old 2: c_id emp.id%type:=&id;
new 2: c_id emp.id%type:=30;
name:sai
address:mumbai
```

PL/SQL procedure successfully completed.

## OUTPUT:

```
SQL> declare
2  c_id emp.id%type:=&id;
3  c_name emp.name%type;
4  c_addr emp.address%type;
5  ex_invalid_id exception;
6  begin
7  if c_id<=0 then
8  raise ex_invalid_id;
9  else
10 select name,address into c_name,c_addr from emp where id=c_id;
11 dbms_output.put_line('name'||c_name);
12 dbms_output.put_line('address'||c_addr);
13 end if;
14 exception
15 when ex_invalid_id then dbms_output.put_line('id should be greater than zero');
16 when no_data_found then dbms_output.put_line('no such customer');
17 when others then dbms_output.put_line('error');
18 end;
19 /
Enter value for id: 30
old 2: c_id emp.id%type:=&id;
new 2: c_id emp.id%type:=30;
c_id emp.id%type:=30;
*
```

```
SQL> DECLARE
2  c_id customers.id%type := 5;
3  c_name customers.name%type;
4  c_addr customers.address%type;
5  BEGIN
6  SELECT name, address INTO c_name, c_addr
7  FROM customers
8  WHERE id = c_id;
9  DBMS_OUTPUT.PUT_LINE ('Name: ' || c_name);
10 DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
11 EXCEPTION
12 WHEN no_data_found THEN
13 dbms_output.put_line('No such customer!');
14 WHEN others THEN
15 dbms_output.put_line('Error!');
16 END;
17 /

PL/SQL procedure successfully completed.
```

## Result:

## **Experiment: 14**

### **PL/SQL Trigger**

#### **AIM:**

#### **Program:**

```
SQL> CREATE OR REPLACE TRIGGER display_salary_changes
  BEFORE DELETE OR INSERT OR UPDATE ON emp
  FOR EACH ROW
  WHEN (NEW.ID > 0)
  DECLARE
    sal_diff number;
  BEGIN
    sal_diff := :NEW.salary - :OLD.salary;
    dbms_output.put_line('Old salary: ' || :OLD.salary);
    dbms_output.put_line('New salary: ' || :NEW.salary);
    dbms_output.put_line('Salary difference: ' || sal_diff);
  END;
/
```

Trigger created.

```
SQL> DECLARE
  total_rows number(2);
  BEGIN
```

```
UPDATE emp
SET salary = salary + 5000;
IF sql%notfound THEN
dbms_output.put_line('no customers updated');
ELSIF sql%found THEN
total_rows := sql%rowcount;
dbms_output.put_line( total_rows || ' customers updated ');
END IF;
END;
/
```

Old salary: 155000

New salary: 160000

Salary difference: 5000

Old salary: 205000

New salary: 210000

Salary difference: 5000

Old salary: 45630

New salary: 50630

Salary difference: 5000

3 customers updated

PL/SQL procedure successfully completed.

**Output:**

```
Trigger created.
```

```
Old salary:  
New salary: 7500  
Salary difference:
```

```
Old salary: 1500  
New salary: 2000  
Salary difference: 500
```

**Result:**

## Experiment 15

### Frame and execute the appropriate PL/SQL Cursors and Exceptional Handling for the project

#### AIM:

#### Procedure:

1. Create table & insert values & commit.
2. Connecting ADO & ADODC to fetch values in MS Access when SQL is prompted.
3. Use connection string: Ex. Choose authentication tab and select username, password as entered for SQL+
4. Select table or stored procedured name as table created in oracle
5. Select data field and set required field name created in table

#### Program:

SQL> create table vb1(Name varchar2(20), Rollno number(3), Maths number(3), English number(3), Physics number(3), Chemistry number(3)); Table created

SQL> desc vb1;

Name	Null?	Type
-----	-----	-----
NAME		VARCHAR2(20)
ROLLNO		NUMBER(3)
MATHS		NUMBER(3)
ENGLISH		NUMBER(3)
PHYSICS		NUMBER(3)
CHEMISTRY		NUMBER(3)

2. **Insert** all the possible values into the vb1 table.
3. Enter **commit** command.

### **Algorithm for ADO Connection:**

After creating the table in Oracle, Go to start menu.

1. Start→ControlPanel→AdministrativeTools→Datasources(ODBC)→User DSN→Add  
→ Select **Microsoft ODBC for Oracle**→Finish→OK
2. One new window will appear. In that window type Data Source Name as table name created in Oracle. Type username as the user name entered in SQL+, Server as 172.31.4.4 and then click O.K.

Algorithm for ADODC in Visual Basic:

1. In Visual Basic create the labels, command buttons and their text boxes.
2. In Visual Basic go to Project menu->Components-> Microsoft ADO Data Control 6.0 for OLEDB->OK
3. Now drag and drop ADODC Data Control available in toolbox into the form.
4. Right click in ADODC Data Control then click the ADODC Properties.
5. In the new window, choose General tab and use ODBC Data source name as the table name created in Oracle(select table name in drop down menu).
6. Choose Use Connection String:  
Orcl->clickBuild->selectOracle provider for OLE
  - Click->Next
  - Data Source: Orcl
  - User Name: as entered in SQL+ login
  - password: student Click ok.
7. Choose Authentication tab and select username, password as entered for SQL+.
8. Choose Record source->select Command type as adcmdTable.
9. Select Table or Stored procedure name as table created in Oracle.



10. Click Apply>O.K
11. Select the Data Source as ADODC1
12. Select the Data Field and set the required field name created in table

Coding:

1.ADD

```
Private Sub ADD_Click()  
Text1.SetFocus  
Adodc1.Recordset.AddNew  
End Sub
```

2.DELETE

```
Private Sub DELETE_Click()  
If MsgBox(" DELETE IT?", vbOKCancel) = vbOK Then  
Adodc1.Recordset.DELETE  
MsgBox "One row deleted"  
End If  
Text1.Text=" "  
Text2.Text =" "  
Text3.Text=" "  
Text4.Text=" "  
Text5.Text=" "  
Text6.Text=" "  
End Sub
```

3.EXIT

```
Private Sub EXIT_Click()  
Unload Me  
End Sub
```

4.SAVE

```
private Sub SAVE_Click()  
If MsgBox("SAVE IT?", vbOKCancel)=vbOK Then  
Adodc1.Recordset.Update  
Else  
Adodc1.Recordset.CancelUpdate  
End If
```

End Sub

## INSERTING VALUES INTO TABLE

SQL> select \* from vb1;

NAME	ROLLNO	MATHS	ENGLISH	PHYSICS
------	--------	-------	---------	---------

CHEMISTRY

Vipin 99	113	100	95	96
Soundarya 90	95	82	86	88

## DELETING ONE ROW FROM TABLE

SQL> select \* from vb1;

NAME	ROLNO	MATHS	ENGLISH	PHYSICS
------	-------	-------	---------	---------

CHEMISTRY

Vipin 99	113	100	95	96
-------------	-----	-----	----	----

**RESULT:**