Ex No:1 **Data Definition Language**
Date:

1 : Create a table called emp with following structure.

Name	Type
ENAME	VARCHAR2(20)
JOB	VARCHAR2(10)
MGR	NUMBER(4)
DEPTNO	NUMBER(3)
SAL	NUMBER(7,2)
EMPNO	NUMBER(6)

Allow NULL for all columns except ename and job.

```
SQL> create table emp (ename varchar2(20) not null ,job varchar2(10) not null ,MGR number(4),deptno
number(3),sal number(7,2),empno number(6));
```

Table created.

```
SQL>desc emp;
```

Name	Null?	Type
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(10)
MGR		NUMBER(4)
DEPTNO		NUMBER(3)
SAL		NMBER(7,2)
EMPNO		NUMBER(6)

2: Add a column commission to emptable .
Commission numeric null allowed

```
SQL> alter table emp add (commission number(10));
```

Table altered.

```
SQL>desc emp;
```

Name	Null?	Type
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(10)
MGR		NUMBER(4)
DEPTNO		NUMBER(3)
SAL		NUMBER(7,2)
EMPNO		NUMBER(6)
COMMISSION		NUMBER(10)

3: Modify the column width of the job field of emp table.

```
SQL> alter table emp modify (job varchar2(20));
```

Table altered.

SQL>desc emp;

Name	Null?	Type
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(20)
MGR		NUMBER(4)
DEPTNO		NUMBER(3)
SAL		NUMBER(7,2)
EMPNO		NUMBER(6)
COMMISSION		NUMBER(10)

4:Create dept table with following structure.

Name	Type
DEPTNO	NUMBER(20)
DNAME	VARCHAR2(10)
LOC	VARCHAR2(10)

SQL> create table dept (deptnonumber(20),dname varchar2(10), loc varchar2(10),constraint deptpk primary key (deptno));

Table created.

SQL>desc dept;

Name	Null?	Type
DEPTNO	NOT NULL	NUMBER(20)
DNAME		VARCHAR2(10)
LOC		VARCHAR2(10)

5: Add constraints to the emp table that empno as the primary key and deptno as the forgien key

SQL> alter table emp add constraint fk foreign key (deptno) references dept(deptno);

Table altered.

6:Add constraints to the emp table to check the empno value while entering (i.e)empno>100.

SQL> alter table empadd(constraint ck check(empno>100));

Table altered.

7:Salary value by default is 5000,otherwise as entered values.

SQL> alter table empmodify(sal default 5000);

Table altered.

8: Add columns Dob to the emp table.

SQL> alter table emp add dob date;

Table altered.
SQL>desc emp;

Name	Null?	Type
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(20)
MGR		NUMBER(4)
DEPTNO		NUMBER(3)
SAL		NUMBER(7,2)
EMPNO		NUMBER(6)
COMMISSION		NUMBER(10)
COMMISSION		NUMBER(10)
DOB		DATE

9: Add and drop a column DOJ to the emp table.

SQL> alter table emp add (doj date);

Table altered.

SQL>desc emp;

Name	Null?	Type
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(20)
MGR		NUMBER(4)
DEPTNO		NUMBER(3)
SAL		NUMBER(7,2)
EMPNO		NUMBER(6)
COMMISSION		NUMBER(10)
COMMISSION		NUMBER(10)
DOB		DATE
DOJ		DATE

SQL> alter table emp drop column doj;

Table altered.

SQL>desc emp;

Name	Null?	Type
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(20)
MGR		NUMBER(4)
DEPTNO		NUMBER(3)
SAL		NUMBER(7,2)
EMPNO		NUMBER(6)
COMMISSION		NUMBER(10)
COMMISSION		NUMBER(10)
DOB		DATE

10: Insert few rows and truncate those from the pseudoemp table and rename it to psemp and also drop it..(Pseudo table)

SQL> create table pseudoemp as select * from emp;
Table created.

SQL>desc pseudoemp;
SQL>desc emp;

Name	Null?	Type
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(20)
MGR		NUMBER(4)
DEPTNO		NUMBER(3)
SAL		NUMBER(7,2)
EMPNO		NUMBER(6)
COMMISSION		NUMBER(10)
COMMISSION		NUMBER(10)
DOB		DATE

SQL> alter table pseudoemp modify rename to psemp;

Table altered.

SQL> truncate table psemp;

Table truncated.

SQL> select * from psemp;

No rows selected.

SQL> drop table psemp;

Table dropped.

Ex.No: 2 **Data Manipulation Language**
Date:

INSERTION VALUES INTO A TABLE:

a) Inserting values for all columns (no need to specify the list of column names)

Syntax:

INSERT INTO <*table_name*> VALUES(<*expr1*>,<*exp2*>,…<*expn*>);

b) Inserting values only for a set of columns

Syntax*:*

INSERT INTO <*table_name*> (<*col1*>,<*col2*>,…<*coln*>)VALUES(<*expr1*>,<*exp2*>,…<*expn*>);

c) Inserting values using identifier method

Syntax:

INSERT INTO <*table_name*> VALUES(&identifier1,&identifier2,…);

SQL> insert into cust values('Anitha',01,'Chennai',1001,'savings',15000);

1 row created.

SQL> insert into cust values('Shriram',02,'Pondy',1002,'savings',25000);

1 row created.

SQL> insert into cust values('Chamundi',03,'Salem',1003,'fd',36200);

1 row created.

SQL> insert into cust values('&cname',&cid,'&caddr',&caccno,'&cacctype',&cbalance);
Enter value for cname: Subha
Enter value for cid: 04
Enter value for caddr: Salem
Enter value for caccno: 1009
Enter value for cacctype: 5000
Enter value for cbalance: 5000
Old 1: insert into cust values('&cname',&cid,'&caddr',&caccno,'&cacctype',&cbalance)
New 1: insert into cust values('Subha',04,'Salem',1009,'RD',5000)

1 row created.

UPDATING THE CONTENTS OF A FILE TABLE

a) Updating all rows

Syntax:

UPDATE <*table_name*> SET <*col1*>=<*value*>;

b) Updating multiple columns (use comma to separate the columns)

Syntax:

UPDATE <*table_name*> SET <*col1*>=<*value1*>**,**<*col2*>=<*value2*>;

c) Updating selected rows

Syntax:

UPDATE <*table_name*> SET <*col1*>=<*expr*> WHERE <*condition*>;

SQL>update cust set caccno=1111 where cname='Chamundi';

1 row updated

SQL> select * from cust;

CNAME	CID	CADDR	CACCNO	CACCTYPE	CBALANCE
Anusha	1	Chennai	1001	savings	15000
Shriram	2	Pondy	1002	savings	25000
Chamundi	3	Salem	1111	fd	36200
Madhan	4	Salem	1004	checkings	5000
Subha	5	Trichy	1005	checkings	10000
Jayashree	6	Pondy	1006	fd	15000
Sridharan	7	Kanchi	1007	fd	22000

7 rows selected.

DELETE OPERATIONS

a) Remove all rows

Syntax:

DELETE FROM <*table_name*>;

b) Removal of a specified rows

Syntax:

DELETE FROM <*table_name*> WHERE <*condition*>;

Delete from table 'cust' the particular record whose accountype is Fixed Deposit.

SQL>delete from cust where cacctype='fd';

3 rows deleted

VIEWING DATA IN THE TABLE

Once data has been inserted into a table, the next most logical operation would be to view what has been inserted.

To display all data in the given table

Syntax:

SELECT * FROM <*table_name*>;

SQL> select * from cust;

CNAME	CID	CADDR	CACCNO	CACCTYPE	CBALANCE
Anusha	1	Chennai	1001	savings	15000
Shriram	2	Pondy	1002	savings	25000
Madhan	4	Salem	1004	checkings	5000
Subha	5	Trichy	1005	checkings	10000

FILTERING TABLE DATA

SQL gives a method of filtering out data that is not required.

a) Selected rows and all columns

Syntax:

SELECT * FROM <*table_name*> WHERE <*condition*>;

b) Selected columns and all rows

Syntax:

SELECT <*col1*>,<*col2*> FROM <*table_name*>;

c) Selected columns and selected rows

Syntax:

SELECT * FROM <*table_name*> ORDER BY <*col1*> ASC|DESC; ASC – Ascending Order

DESC – Descending Order

Ex No:3 **Employee Database using constraints**

Date:

AIM:

To create a fully functional employee database with proper constraints.

Question No.1

Create a query to find the annual revenue that is collected by the different companies using the property table

Query: select office_id,sum(prop.prop_rent*12) "Annual Revenue" from prop group by office_id;

Output:

OFFICE_ID	Annual Revenue
1207	678000
1208	1680000
1205	714000
1204	432000
1206	342000

Question No.2

Create a query to find the inspection date, location of property and the rent using the inspection and property tables using joins

Query: select inspec.insp_date, prop.prop_city, prop.prop_rent from prop left outer join inspec on inspec.prop_id=prop.prop_id;

Output:

INSP_DATE	PROP_CITY	PROP_RENT
12-JAN-15	Sydney	21000
08-FEB-15	Berlin	15000
27-MAR-16	Zurich	28000
15-APR-15	Rome	31500
07-DEC-15	Kolkatta	16000
10-OCT-15	Agra	12500
30-JUL-15	Tokyo	95000
16-OCT-15	Paris	35500
	Hong Kong	45000
	Shimla	21000

Question No.3

Create a query to find when an employee is going to visit the property owned by the various companies

Query: select inspec.insp_date, prop.prop_city, employee.emp_name from inspec,prop,employee where inspec.emp_id=employee.emp_num and inspec.prop_id=prop.prop_id;

Output:

INSP_DATE	PROP_CITY	EMP_NAME
10-OCT-15	Agra	Sarulakshmi
30-JUL-15	Tokyo	Kenshin Himura
16-OCT-15	Paris	Relicum Eleon
12-JAN-15	Sydney	Tony Stark
02-FEB-15	Berlin	Johnny Gates
27-MAR-16	Zurich	Eleon Blade
15-APR-15	Rome	Drakkus Storm
07-DEC-15	Kolkatta	Gordon Ramsey

Question No.4

Create a query to find the rent paid per room and the location of the property with the company that owns it

Query: select round((p.prop_rent/p.prop_rooms),2) "Cost per room", p.prop_city, o.office_name from prop p,office o where p.office_id=o.office_num;

Output:

Cost per room	PROP_CITY	OFFICE_NAME
3750	Berlin	Stark Ind.
4200	Sydney	Stark Ind.
9333.34	Zurich	Storm Factory
5250	Rome	Storm Factory
4000	Kolkatta	Adam Lines
4166.67	Agra	Adam Lines
4200	Shimla	Eleon Estates
5916.67	Paris	Eleon Estates

Question No.5

Create a query to get the employee id and the location of all employees with their names

Query: select emp_num, concat(emp_name,concat(' lives in ',emp_city)) "Location" from employee;

Output:

EMP_NUM	Location
4624	Tony Stark lives in New York
4625	Pepper Potts lives in New York
4626	Johnny Gates lives in New York
4627	Drakkus Storm lives in Goa
4628	Eleon Blade lives in London
4629	Adam Lambert lives in Paris
4630	Gordon Ramsey lives in Italy
4631	Sarulakshmi lives in Mumbai
4632	Relicum Eleon lives in New Delhi

Question No.6

Create a query to display the employee their location and the comment given by them for the property inspected by them

Query: select e.emp_name, e.emp_city, i.insp_comment from employee e inner join inspec i on i.emp_id=e.emp_num;

Output:

EMP_NAME	EMP_CITY	INSP_COMMENT
Sarulakshmi	Mumbai	Architecture is good. Some improvisions need to be done
Kenshin Himura	Tokyo	Information Classified. Details will be given after purchase or clearance only
Relicum Eleon	New Delhi	Infrastructure is defective but with little capital it can be fixed. Busy routes with high passenger count
Tony Stark	New York	No Comment
Johnny Gates	New York	Infrastructure is defective but with little capital it can be fixed. Busy routes with high passenger count
Eleon Blade	London	Infrastructure is defective but with little capital it can be fixed. Busy routes with high passenger count
Drakkus Storm	Goa	Architecture is good. Some improvisions need to be done
Gordon Ramsey	Italy	No Comment

Question No.7

Create a query to list all cities present in both Employee and property tables using union

Query: select emp_city "Cities Used" from employee union select prop_city from prop;

Output:

Cities Used
Agra
Berlin
Goa
Hong Kong
Italy
Kolkatta
London
Mumbai
New Delhi
New York
Paris
Rome
Shimla

Question No.8

Create a query to select all details of employees who lives in a city that starts with “New”

Query: select emp_num,emp_name,emp_phone from employee where emp_city like 'New%';

Output:

EMP_NUM	EMP_NAME	EMP_PHONE
4624	Tony Stark	9841293456
4625	Pepper Potts	9941593644
4626	Johnny Gates	9845263414
4632	Relicum Eleon	5241602510

Question No.9

Create a query to update property table where the city name of property is changed from ‘Kolkatta’ to ‘Bengaluru’ using Replace function and display the result

Query: update prop set prop_city = replace(prop_city,'Kolkatta','Bengaluru');

Output: 10 rows updates

Query: select * from prop;

Output:

PROP_ID	PROP_CITY	PROP_RENT	PROP_ROOMS	OFFICE_ID
1265	Berlin	15000	4	1204
1684	Sydney	21000	5	1204
8524	Zurich	28000	3	1205
6512	Rome	31500	6	1205
9856	Bengaluru	16000	4	1206
7542	Agra	12500	3	1206
3285	Paris	35500	6	1207
1001	Hong Kong	45000	9	1208
1002	Tokyo	95000	8	1208

Question No.10

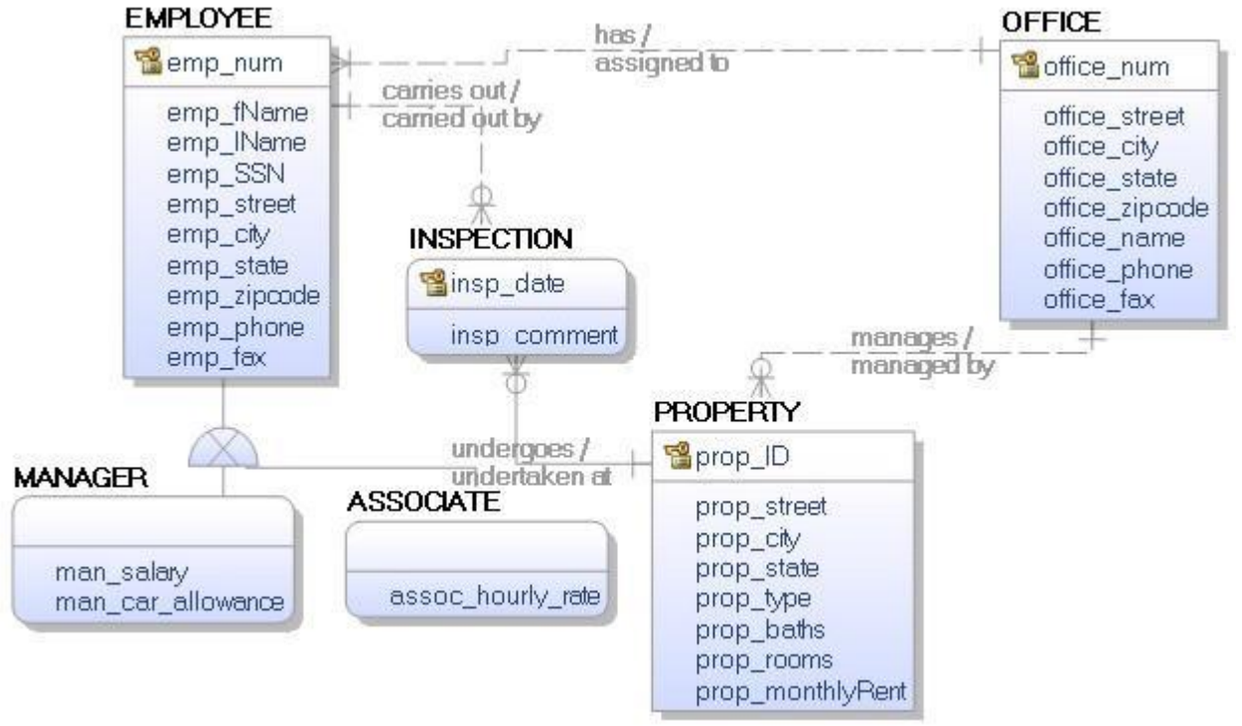
Create a query to select distinct companies with their details which owns properties that have more than 5 rooms

Query: select office_num,office_name,office_city from office where office_num in (select office_id from prop where prop_rooms>5);

Output:

OFFICE_NUM	OFFICE_NAME	OFFICE_CITY
1205	Storm Factory	Chennai
1207	Eleon Estate	New Delhi
1208	SHIELD	California

ER DIAGRAM



RESULT:

The desired queries were created and the output was verified and noted down.

Ex. No: 4 **Implementation of View, Index, Sequence and Synonyms**
Date:

AIM:

To study and implement View, Index, Sequence and Synonym concepts using Oracle 9i.

VIEW

GENERAL FORMAT:

CREATE VIEW *view_name* AS

SELECT <*column_list*> FROM *table_name* [WHERE *condition*];

The view_name is the name of the VIEW. The SELECT statement is used to define the columns and rows that have to be displayed.

FOR HORIZONTAL VIEW:

CREATE VIEW *view_name* AS

SELECT * FROM *table_name* [WHERE *condition*];

FOR VERTICAL VIEW:

CREATE VIEW *view_name* AS

SELECT <column_list> FROM *table_name;*

DROPPING A VIEW:

DROP VIEW *view_name*;

The WITH CHECK OPTION:

The following is an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION:

CREATE VIEW CUSTOMERS_VIEW

AS SELECT * FROM CUSTOMERS WHERE age IS NOT NULL

WITH CHECK OPTION;

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View:

So if a view satisfies all the above-mentioned rules then you can update a view. Following is an example to update the age of Ramesh:

```
UPDATE CUSTOMERS_VIEW SET AGE = 35 WHERE name='Ramesh';
```

Force View Creation

'Force' keyword is used while creating a view. This keyword force to create View even if the table does not exist. After creating a force View if we create the base table and enter values in it, the view will be automatically updated. Syntax for forced View is,

```
CREATE or REPLACE force view view_name AS
    SELECT <column_name> FROM table_name WHERE condition
```

Read-Only View

A view can be created with read-only option to restrict access to the view. Syntax to create a view with Read-Only Access is as follows,

```
CREATE or REPLACE force view view_name AS SELECT
<column_name> FROM table_name WHERE <condition> with
read-only
```

The above syntax will create view for read-only purpose, Any Update or Insert data into read-only view, will throw an error.

INDEX

INTRODUCTION:

Single-Column Indexes:

A single-column index is one that is created based on only one table column. The basic syntax is as follows:

```
CREATE INDEX index_name ON table_name (column_name);
```

Unique Indexes:

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows:

```
CREATE UNIQUE INDEX index_name on table_name (column_name);
```

Composite Indexes:

A composite index is an index on two or more columns of a table. The basic syntax is as follows:

CREATE INDEX index_name on table_name (column1, column2);

Implicit Indexes:

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

RENAME AN INDEX

ALTER INDEX old_index_name RENAME TO new_index_name;

DROP AN INDEX

An index can be dropped using SQL DROP command. Care should be taken when dropping an index because performance may be slowed or improved. The basic syntax is as follows:

DROP INDEX index_name;

CREATING A SEQUENCE

Syntax:

CREATE SEQUENCE <sequence-name>

start with <*initial-value*>

increment by <*increment-value*>

maxvalue <*maximum-value*>

cycle | nocycle

USING A SEQUENCE

Syntax:

*sequence_name***.nextval**

Example:

CREATE SEQUENCE cse_batchno **start with** 171001 **increment by** 1

maxvalue 171999 **nocycle;**

This will create a new sequence for generating student batch numbers. When a new student details is inserted, the sequence can be used for assigning the batch no for that student. It is not necessary to remember the batch number every time for insertion.

INSERT INTO STUDENT VALUES(cse_batchno.nextval,'John',18,'M');

INSERT INTO STUDENT VALUES(cse_batchno.nextval,'Joy',19,'F');

INSERT INTO STUDENT VALUES(cse_batchno.nextval,'Pete',17,'M');

INSERT INTO STUDENT VALUES(cse_batchno.nextval,'Will',18,'M');

Output:

BatchNo	Name	Age	Gender

171001	John	18	M
171002	Joy	19	F
171003	Pete	17	M
171004	Will	18	M

SQL SYNONYMS

CREATING SYNONYMS

Create or replace [public] synonym <synonym_name> FOR <table_name>; Example:

Create synonym cse_stu FOR student;

Select * From student;

BatchNo	Name	Age	Gender
171001	John	18	M
171002	Joy	19	F
171003	Pete	17	M
171004	Will	18	M

Select * From cse_stu;

BatchNo	Name	Age	Gender
171001	John	18	M
171002	Joy	19	F
171003	Pete	17	M
171004	Will	18	M

Notice:

Fetch the data from the table “student” or from the synonym “cse_stu” will produce the same result.

DROP SYNONYM

The drop synonym command is used to drop public and private synonyms.

To drop public synonym

DROP PUBLIC SYNONYM <synonym_name>;

To drop private synonym

DROP SYNONYM <synonym_name>;

Ex.No:5
Date

Creating Relationship between Databases

Nested Queries:

Nesting of queries one within another is known as a nested queries.

Sub queries

The query within another is known as a sub query. A statement containing sub query is called parent statement. The rows returned by sub query are used by the parent statement.

Example: *select ename, eno, address where salary >(select salary from employee where ename ='jones');*

Types

1. Sub queries that return several values

Sub queries can also return more than one value. Such results should be made use along with the operators in and any.

Example: *select ename, eno, from employee where salary <any (select salary from employee where deptno =10');*

2. Multiple queries

Here more than one subquery is used. These multiple subqueries are combined by means of 'and' & 'or' keywords.

3. Correlated subquery

A subquery is evaluated once for the entire parent statement whereas a correlated subquery is evaluated once per row processed by the parent statement.

Example: *select * from emp x where x.salary > (select avg(salary) from emp where deptno =x.deptno);*

Above query selects the employees details from emp table such that the salary of employee is > the average salary of his own department.

1. Display the names of all the employees who belong to the transport department.

SYNTAX:

SELECT * FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT ename FROM emp WHERE deptno= (SELECT dno FROM dept WHERE dname in ('Transport'));

OUTPUT:

ENAME
Ward

Clark

2. Find the names of all employees who work in the same job as Clark.

SYNTAX:

SELECT * FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT * FROM emp WHERE job= (SELECT job FROM emp WHERE ename='Clark');

OUTPUT:

ENAME
Ward

3.Find the minimum salary among all employees working in the 2nd department such that this salary should be greater than the maximum salary of the 3rd department.

SYNTAX:

SELECT attributes FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT min (sal) from emp where deptno=2 AND sal> (select max (sal) from emp where deptno=3);

OUTPUT:

Min(sal)
5000

4.Display the names of all the employees who either have the same job as Clark or the same salary.

SYNTAX:

SELECT attribute FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT * FROM emp WHERE job= (SELECT job FROM emp WHERE ename='Clark') OR sal=(SELECT sal FROM emp WHERE ename='Clark');

OUTPUT:

ENO	ENAME	JOB	DOB	SAL	COMM	DEPTNO
105	CLARK	MANAGER	15/07/1987	15435.46	122	20
100	WARD	MANAGER	25/11/1986	46435.8	132	10

5. Display the employee details of all the employees whose salary is greater than the average salary.

SYNTAX:

SELECT attribute FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT empid, ename, sal, deptid FROM emp e WHERE sal > (SELECT avg(sal) FROM employee WHERE deptid=e.deptid);

OUTPUT:

EMPID	ENAME	SAL	DEPTID
115	Allen	23546.7	20

6.Display the names of all employees who belong to the department admin

SYNTAX:

SELECT attributes FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT ename FROM emp WHERE deptno= (SELECT deptno FROM department where dname='Administration');

OUTPUT:

ENAME
Kent
Ward

7.Display the names of all the employees who are working under the management department and earn more than 5000.

SYNTAX:

SELECT attribute FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT * FROM emp WHERE deptno=(SELECT deptno FROM department where dname='Management');

OUTPUT:

EMPID	ENAME	JOB	DOB	SAL	COMM	DEPTNO
105	Allen	Manager	15/07/1987	15435.6	122	20
115	Clark	Clerk	19/05/1988	23456.8	132	20

8.Find the department that has the least number of employees working under it.

SYNTAX:

SELECT attributes FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT dname FROM department where deptno IN (SELECT deptno FROM emp GROUP BY deptno HAVING count(empid)=(SELECT min(count(empid)) FROM emp GROUP BY deptno));

OUTPUT:

DNAME
Sales
Transport

9.Find the department that has the most number of employees working under it.

SYNTAX:

SELECT attributes FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT dname FROM department where deptno IN (SELECT deptno FROM emp GROUP BY dno HAVING count(empno)=(SELECT max(count(empno)) FROM emp GROUP BY deptno));

OUTPUT:

DNAME
Management

10.Display the names of all employees and their department names using left outer join.

SYNTAX:

SELECT attributes FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT e.ename,d.dname FROM emp e LEFT OUTER JOIN dept d on (e.deptno=d.deptno);

OUTPUT:

ENAME	**DNAME**
MILLER	ACCOUNTING
KING	ACCOUNTING
CLARK	ACCOUNTING
FORD	RESEARCH
ADAMS	RESEARCH
SCOTT	RESEARCH
JONES	RESEARCH
SMITH	RESEARCH
JAMES	SALES

11.Implement the 10th query using a right outer join.

SYNTAX:

SELECT attributes FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT e.ename,d.dname FROM emp eRIGHT OUTER JOIN dept d on(e.deptno=d.deptno);

OUTPUT:

ENAME	**DNAME**
CLARK	ACCOUNTING
KING	ACCOUNTING
MILLER	ACCOUNTING
JONES	RESEARCH
FORD	RESEARCH
ADAMS	RESEARCH
SMITH	RESEARCH
SCOTT	RESEARCH
WARD	SALES
TURNER	SALES
ALLEN	SALES

12.Implement the 10th query using a full outer join.

SYNTAX:

SELECT attributes FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT e.ename,d.dname FROM emp e FULL OUTER JOIN dept d on(e.deptno=d.deptno);

OUTPUT:

ENAME	DNAME
MILLER	ACCOUNTING
KING	ACCOUNTING
CLARK	ACCOUNTING
FORD	RESEARCH
ADAMS	RESEARCH
SCOTT	RESEARCH
JONES	RESEARCH
SMITH	RESEARCH
JAMES	SALES
TURNER	SALES
BLAKE	SALES

13.Display the employee names and their manager names.

SYNTAX:

SELECT attributes FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT e.ename,m.ename FROM emp e,emp m WHERE e.mgr=m.empno;

OUTPUT:

ENAME	ENAME
FORD	JONES
SCOTT	JONES
TURNER	BLAKE
ALLEN	BLAKE
WARD	BLAKE
JAMES	BLAKE
MARTIN	BLAKE
MILLER	CLARK
ADAMS	SCOTT
BLAKE	KING
JONES	KING
CLARK	KING
SMITH	FORD

14.Write a query to display the employee names and the location where they are working in.

SYNTAX:

SELECT attributes FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT e.ename ,d.loc FROM emp e ,dept d WHERE e.deptno=d.deptno;

OUTPUT:

ENAME	LOC
CLARK	NEW YORK
KING	NEW YORK
MILLER	NEW YORK

JONES	DALLAS
FORD	DALLAS
ADAMS	DALLAS
SMITH	DALLAS
SCOTT	DALLAS
WARD	CHICAGO
TURNER	CHICAGO
ALLEN	CHICAGO

15.Display the names of all the employees working under a department that has not yet been accredited.

SYNTAX:

SELECT attributes FROM table_name WHERE column_name = (select * FROM table_name WHERE condition);

SOLUTION:

SELECT e.ename,d.dname FROM emp e,dept d where e.deptno=d.deptno AND deptno NOT IN(SELECT deptno FROM accredit);

OUTPUT:

ENAME	DNAME
MILLER	ACCOUNTING
KING	ACCOUNTING
CLARK	ACCOUNTING
FORD	RESEARCH
ADAMS	RESEARCH

JOINS

EMPLOYEE TABLE:

EMP ID	LOCATION
100	NEWYORK
101	NEWJERSEY
102	AUCKLAND
103	BRISBANE
104	SYDNEY

LOCATION TABLE:

EMPID	EMPNAME
100	MILLER
101	KING
102	CLARK
103	FORD
110	ADAMS

LEFT OUTER JOIN:

Syntax;

Select * from table1 left outer join table2 on table1.colname=table2.colname;

Eg;

Select * from employee left outer join location on employee.empid=location.empid;

Output:

EMPID	EMPNAME	EMPID	LOCATION
100	MILLER	100	NEWYORK
101	KING	101	NEWJERSEY
102	CLARK	102	AUCKLAND
103	FORD	103	BRISBANE
104	ADAMS		

RIGHT OUTER JOIN:

Syntax:

Select * from table1 right outer join table2 on table1.colname=table2.colname;

Eg;

Select * from employee right outer join location on employee.id=location.id;

Output :

EMPID	EMPNAME	EMPID	LOCATION
100	MILLER	100	NEWYORK
101	KING	101	NEWJERSEY
102	CLARK	102	AUCKLAND
103	FORD	103	BRISBANE
		110	SYDNEY

FULL OUTER JOIN:

Syntax:

select * from table1.full outer join table2 on table1.colname=table2.colname;

Eg:

select * from employee full outer join location on employee.id=location.id;

Output:

EMPID	EMPNAME	EMPID	LOCATION
100	MILLER	100	NEWYORK
101	KING	101	NEWJERSEY
102	CLARK	102	AUCKLAND
103	FORD	103	BRISBANE
104	ADAMS		
		110	SYDNEY

INNER JOIN:

Syntax:

select * from table1 inner join table2 on table1.colname=table2.colname;

Eg:

Select * from employee inner join location on employee.id=location.id;

Output:

EMPID	EMPNAME	EMPID	LOCATION
100	MILLER	100	NEWYORK
101	KING	101	NEWJERSEY
102	CLARK	102	AUCKLAND
103	FORD	103	BRISBANE

CROSS JOIN:

Syntax:

Select * from table1 cross join table2;

Eg:

select * from employee cross join location;

Output:

EMPID	EMPNAME	EMPID	LOCATION
100	MILLER	100	NEWYORK
100	MILLER	101	NEWJERSEY
100	MILLER	102	AUCKLAND
100	MILLER	103	BRISBANE
100	MILLER	110	SYDNEY
101	KING	100	NEWYORK
101	KING	101	NEWJERSEY
101	KING	102	AUCKLAND
101	KING	103	BRISBANE
101	KING	110	SYDNEY
102	CLARK	100	NEWYORK
102	CLARK	101	NEWJERSEY
102	CLARK	102	AUCKLAND
102	CLARK	103	BRISBANE
102	CLARK	110	SYDNEY
103	FORD	100	NEWYORK
103	FORD	101	NEWJERSEY
103	FORD	102	AUCKLAND
103	FORD	103	BRISBANE
103	FORD	110	SYDNEY
104	ADAMS	100	NEWYORK
104	ADAMS	101	NEWJERSEY
104	ADAMS	102	AUCKLAND
104	ADAMS	103	BRISBANE
104	ADAMS	110	SYDNEY

Ex. No:6 **Study of PL/SQL Programming**
Date:

INTRODUCTION:

- PL/SQL stands for Procedural Language extension of SQL. PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.
- It extends SQL by adding control structures found in other procedural languages. Procedural constructs blend seamlessly with oracle SQL, resulting in structured, powerful language.
- PL/SQL is unique as it combines the flexibility of SQL with power and configurability of third generation language. The procedural constructs and database access are present in PL/SQL.
- PL/SQL can be used in oracle relational database and in client side application development tools. Oracle supports PL/SQL which can support many enhancement of oracle, including large objects, object oriented design and development & collections.

ADVANTAGES:

PL/SQL is completely portable and high performance. Transaction processing language offers following advantages,

- Supports declaration and manipulation of object type and collections.
- Allows the calling of external functions and procedures.
- Contains new libraries of built-in packages.
- Higher productivity with less effort.

PL/SQL has been upgraded to support directly most of new features of DBMS but not partitioned objects. There is extensive support for new type system and support for tables has been further extended.

SUPPORT FOR SQL:

PL/SQL allows us to use all SQL data manipulation statements, transaction control statements, SQL functions [except group functions], operators and pseudo columns, thus allowing us to manipulate data in table more flexibly and effectively.

BETTER PERFORMANCE:

Without PL/SQL, Oracle must process SQL statement one at a time. With PL/SQL, en entire block of statements can be processed in a single command line statement. This reduces the time required for communication between the application and the oracle server which may increase the overall performance.

PORTABILITY:

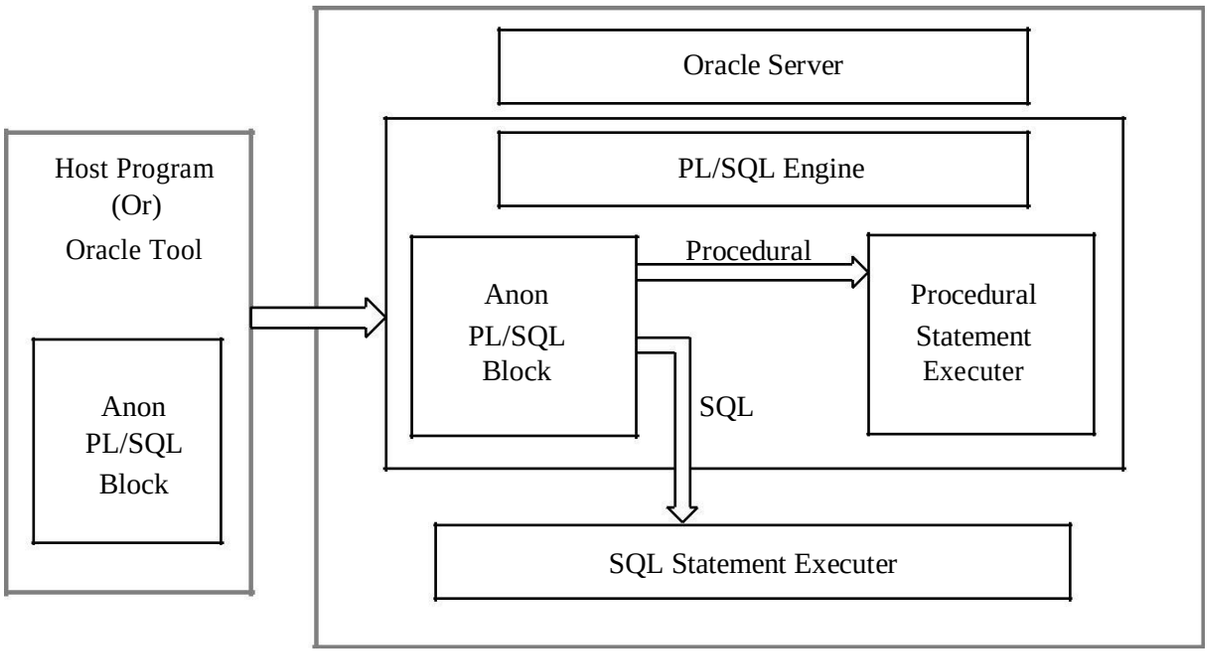
Applications written in PL/SQL are portable to any operating systems or platforms on which oracle is installed.

INTEGRATION WITH ORACLE:

Both PL/SQL and oracle have their foundations in SQL. PL/SQL supports all SQL data types and it integrates PL/SQL with oracle data dictionary.

ARCHITECTURE OF PL/SQL:

The PL/SQL engine executes PL/SQL blocks and it only executes the procedural statements and sends the statements to the executer in the oracle server. The PL/SQL engine resides in oracle server.



PL/SQL BLOCK:

A PL/SQL block can be divided into 3 parts namely,

- The Declaration section (optional).
- The Execution section (mandatory).
- The Exception Handling (or Error) section (optional).

The order is as shown below:

```
DECLARE
        -- Variable Declaration
BEGIN
        -- Program Executable Statements
EXCEPTION
        -- Exception Handling Statements
END;
```

DECLARATIVE SECTION:

The declarative section of PL/SQL block, starts with the reserved keyboard DECLARE and this section is an optional. It is used to declare any places hold like variables, consonants, records and cursors which are used to manipulate data in the execution section.

EXECUTION SECTION:

The execution section of a PL/SQL block starts with the reserved keyword BEGIN and ends with END. This is a mandatory section and is the section where the program logic is written to perform any task.

The programmatic constructs like loops, conditional statements and SQL statements form the part of the execution section.

EXCEPTION SECTION:

The exception section of a PL/SQL block starts with the reserved keyword EXCEPTION and it is optional. Any errors in the program can be handled in this section.

- Every statement in the above three sections must end with a semicolon (;). The PL/SQL blocks can be nested with in other PL/SQL blocks comments can be used to document code.

- Objects can be declared in declarative part, which can be used in executable part for further manipulations. All procedural statements are included between BEGIN and END statements. Errors that occur during execution are dealt in exception handling part.

- Before proceeding to learn about three parts, we need to have brief idea on the character set and lexical units used in PL/SQL block.

- The PL/SQL character set includes the following:

  o Alphabets are not case sensitive except within strings and enhance character literals. o Numbers for 0 to 9

  o All special symbols and characters.

  o Tab, space and carriage return

- PL/SQL text can contain groups of characters known as lexical units such as Identifiers, Literals, Comments and Delimiters (simple and compound symbols)

% Type:

The % type attribute provides the data type of a variable or database column. This is particularly useful when declaring variables that will hold database values. For example, assume there is a column named last_name in a table named employees to declare a variable named v_last_name.

v_last_name has the same datatype as column title, use dot notations and % type attribute as follows,

Syntax:

variable_name table_name, attribute_name %TYPE;

e.g.:

v_ename employees.ename %TYPE;

%ROW TYPE:

In PL/SQL records are used to group data. A record consists of a number of related fields in which data values can be started. The %ROW TYPE attributes provides a record type that represents a single row in a table. The record can store an entire row of data selected from the table or fetched from a cursor or cursor variables.

Syntax:

variable table_name %ROWTYPE;

e.g.:

d dept %ROWTYPE;

Ex. No: 6 b **CONTROL STRUCTURES**

AIM:

To write the PL/SQL blocks to demonstrate the usage of control statements such as conditional statements and looping statements.

INTRODUCTION:

In addition to SQL commands, PL/SQL can also process data using flow of statements. The flow of control statements are classified into the following categories.

i. Conditional control - Branching
ii. Iterative control – Looping

i) CONDITIONAL STATEMENT in PL/SQL:

Sequence of statements can be executed on satisfying certain condition. If statements are being used and different forms of 'if' are:

2. Simple IF
3. ELSIF
4. ELSE IF

SIMPLE IF STATEMENT:

Syntax:

```
IF condition THEN
        statement1;
        statement2;
END IF;
```

IF-THEN-ELSE STATEMENT:

Syntax:

```
IF condition THEN
        statement1;
ELSE
        statement2; END
IF;
```

ELSIF STATEMENTS:

Syntax:

```
IF condition1 THEN
        statement1;
ELSIF condition2 THEN
        statement2;
ELSIF condition3 THEN
        statement3;
ELSE
        statementn; END
IF;
```

NESTED IF STATEMENT:

Syntax**:**

```
IF condition THEN
        statement1;
ELSE
        IF condition THEN
                statement2;
        ELSE
                statement3; END
        IF;
END IF;
```

ii) ITERATIONS IN PL/SQL

Sequence of statements can be executed any number of times using loop construct. It is broadly classified into:

1. Simple Loop
2. For Loop
3. While Loop

SIMPLE LOOP

Syntax:

```
LOOP
        statement1;
EXIT [ WHEN Condition]; END
LOOP;
```

Example:

```
Declare
        A number:=10;
Begin
        Loop
                exit when a=250; end
        loop;
        dbms_output.put_line(to_char(a));
end;
```

WHILE LOOP

Syntax:

```
WHILE condition LOOP
        statement1; statement2;
END LOOP;
```

Example:

```
Declare
i    number:=0;
j    number:=0;
begin
        While i<=100 Loop j := j+i;
                i := i+2;
```

```
        end loop;
                                        dbms_output.put_line('the value of j is' ||j);
    end;
```

FOR LOOP

Syntax:

```
    FOR counter IN [REVERSE]
        LowerBound..UpperBound
    LOOP
        statement1;
        statement2; END
    LOOP;
```

Example:

```
Begin
    For I in 1..2 Loop
        Update emp set field = value where condition End loop;
End;
```

Ex. No.: 6 c **Exception Handling in PL/SQL**

INTRODUCTION:

An error condition during a program execution is called an exception in PL/SQL. PL/SQL supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error condition. PL/SQL provides a feature to handle the exceptions which occur in a PL/SQL Block known as 'Exception Handling'. By handling the exceptions we can ensure a PL/SQL block does not exit abruptly.

When an exception occurs a messages which explains its cause is received. PL/SQL Exception message consists of following three parts such as,

1) Type of Exception
2) An Error Code
3) A message

Structure of Exception Handling

```
DECLARE
        Declaration section
BEGIN
        Execution section
EXCEPTION
        Exception section
END;
```

Types of Exception

There are 3 types of Exceptions.

3. Named System Exceptions
4. Unnamed System Exceptions
5. User-defined Exceptions

a) Named System Exceptions

System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as Named System Exceptions.

For example: NO_DATA_FOUND and ZERO_DIVIDE are called Named System exceptions.

b) Unnamed System Exceptions

Those system exception for which oracle does not provide a name is known as unnamed system exception. These exception do not occur frequently. These Exceptions have a code and an associated message.

There are two ways to handle unnamed system exceptions:

1. By using the WHEN OTHERS exception handler, or
2. By associating the exception code to a name and using it as a named exception.

c) User-defined Exceptions

Apart from system exceptions we can explicitly define exceptions based on business rules. These

are known as user-defined exceptions.

Steps to be followed to use user-defined exceptions:

- They should be explicitly declared in the declaration section.
- They should be explicitly raised in the Execution Section.
- They should be handled by referencing the user-defined exception name in the exception section.

RAISE_APPLICATION_ERROR ( )

RAISE_APPLICATION_ERROR is a built-in procedure in oracle which is used to display the user-defined error messages along with the error number whose range is in between -20000 and -20999.

Whenever a message is displayed using RAISE_APPLICATION_ERROR, all previous transactions which are not committed within the PL/SQL Block are rolled back automatically.

RAISE_APPLICATION_ERROR raises an exception but does not handle it and it is used for the following reasons,

a) To create a unique id for a user-defined exception.

b) To make the user-defined exception look like an Oracle error.

The General Syntax:

RAISE_APPLICATION_ERROR (error_number, error_message);

Ex.No:7 **Creation of Procedures**
Date:

A procedure is a block that can take parameters (also known as arguments) and can be invoked by other programs for doing specific tasks.

Procedures are sub programs, used to provide modularity and encapsulation in an application. Once the procedure are built and validated, then they can be used in number of applications.

Syntax for procedure:

```
CREATE [OR REPLACE] PROCEDURE <procedure_name> (parameter1 [IN | OUT | IN
OUT] datatype ,...
parametern [IN | OUT | IN OUT] datatype) AS
        <local variable declaration>
BEGIN
        PL/SQL executable statements;
        ..
EXCEPTION
        Exception handlers;
END;
```

************************************ADDITION****************************

```
SQL> declare
  2   a number;
  3   b number;
  4  c number;
  5   begin
  6  a:=:a;
  7  b:=:b;
  8  c:=a+b;
  9  dbms_output.put_line('sum of'||a||'and'||b||'is'||c);
 10  end;

Enter value for a: 5
old   6: a:=:a;
new   6: a:=5;
Enter value for b: 5
old   7: b:=:b;
new   7: b:=5;
sum of5and5is10

PL/SQL procedure successfully completed.
```

******************************MAXIMUM NO.******************************

```
SQL> declare
  2   a number;
  3   b number;
  4   c number;
  5   d number;
  6   begin
  7    dbms_output.put_line('enter a:');
  8  a:=:a;
  9   dbms_output.put_line('enter b:');
 10  b:=:b;
 11   dbms_output.put_line('enter c:');
 12  c:=:b;
 13  if(a>b)and(a>c) then
 14   dbms_output.put_line('A is maximum');
 15  elsif(b>a)and(b>c)then
 16   dbms_output.put_line('B is maximum');
 17  else
 18   dbms_output.put_line('C is maximum');
 19  end if;
 20* end;


Enter value for a: 9
old   8: a:=:a;
new   8: a:=9;
Enter value for b: 6
old  10: b:=:b;
new  10: b:=6;
Enter value for c: 10
old  12: c:=:b;
new  12: c:=10;
enter a:
enter b:
enter c:
C is maximum
```

PL/SQL procedure successfully completed.

********** **REVERSING THE NO.**********

```
SQL> declare
  2  given_number varchar(5):='1234';
  3  str_length number(2);
  4  inverted_number varchar(5);
  5  begin
  6  str_length:=length(given_number);
  7  for cntr in reverse 1..str_length
  8  loop
  9  inverted_number:=inverted_number||substr(given_number,cntr,1);
 10  end loop;
 11  dbms_output.put_line('the given no is'||given_number);
 12  dbms_output.put_line('the inverted number is'|| inverted_number);
 13 end;


the given no is1234
the inverted number is4321

PL/SQL procedure successfully completed.
```

******************************SUM OF 100 NO.**************************

```
SQL> declare
  2  a number;
  3  s1 number default 0;
  4  begin
  5  a:=1;
  6  loop
  7  s1:=s1+a;
  8  exit when (a=100);
  9  a:=a+1;
 10  end loop;
 11  dbms_output.put_line('sum bt 1 to 100 is'|| s1);
 12  end;

sum bt 1 to 100 is5050

PL/SQL procedure successfully completed.
```

******************************NET SALARY***************************

```
SQL> declare
 2  ename varchar2(15);
 3  basic number;
 4  da number;
 5  hra number;
 6  pf number;
 7  netsalary number;
 8  begin
 9  ename:=:ename;
10  basic:=:basic;
11  da:=basic*(41/100);
12  hra:=basic*(15/100);
13  if(basic<3000)
14  then
15  pf:=basic*(5/100);
16  elsif(basic>=3000 and basic<=5000)
17  then
18  pf:=basic*(7/100);
19   elsif(basic>=5000 and basic<=8000)
20  then
21   pf:=basic*(8/100);
22  else
23   pf:=basic*(10/100);
24  end if;
25  netsalary:=basic+da+hra-pf;
26  dbms_output.put_line('employee name:'||ename);
27  dbms_output.put_line('providend fund:'||pf);
28  dbms_output.put_line('net salary:'||netsalary);
29* end;

enter value for ename: 'ice'
old   9: ename:=&ename;
new   9: ename:='ice';
enter value for basic: 1000
old  10: basic:=&basic;
new  10: basic:=1000;
employee name:ice
providend fund:50
net salary:1510
```

PL/SQL procedure successfully completed.

**************************AREA CALCULATION**************************

```
SQL> declare
  2    pi constant number(4,2):=3.14;
  3    radius number(5);
  4    area number(14,2);
  5    begin
  6    radius:=3;
  7    while radius<=7
  8    loop
  9    area:=pi*power(radius,2);
 10     insert into areas values(radius,area);
 11     radius:=radius+1;
 12     end loop;
 13     end;
 14     /

PL/SQL procedure successfully completed.

SQL> select* from areas;

    RADIUS       AREA
---------- ----------
         3      28.26
         4      50.24
         5       78.5
         6     113.04
         7     153.86
         7        154
         5      78.54
         3      28.26
         4      50.24
         5       78.5
         6     113.04

    RADIUS       AREA
---------- ----------
         7     153.86

12 rows selected.
```

Ex.No:8(a) **Creation of Database Triggers and Functions**
Date:

SQL> create table stud(rno number(2),mark1 number(3),mark2 number(3),total number(3),primary key(rno));

Table created.

SQL> desc stud;

Name	Null?	Type
RNO	NOT NULL	NUMBER(2)
MARK1		NUMBER(3)
MARK2		NUMBER(3)
TOTAL		NUMBER(3)

SQL> select * from stud;

RNO	MARK1	MARK2	TOTAL
1	80	85	0
2	75	84	0
3	65	80	0
4	90	85	0

SQL> create or replace function stude(rnum number) return number is
2 total number;
3 m1 number;
4 m2 number;
5 ***begin***
6 select mark1,mark2 into m1,m2 from stud where rno=rnum;
7 total:=m1+m2;
8 return total;
9 ***end;***
10 /

Function created.

SQL> select stude(2) from dual;

STUDE(2)

159

SQL> create table purchase (icode number(3),iname varchar2(13),price number(6),quantity number(3),rate number(8),primary key(icode),unique(iname));

Table created.

SQL> desc purchase;

Name	Null?	Type
ICODE	NOT NULL	NUMBER(3)
INAME		VARCHAR2(13)
PRICE		NUMBER(6)

QUANTITY NUMBER(3)
RATE NUMBER(8)

SQL> select * from purchase;

ICODE	INAME	PRICE	QUANTITY	RATE
100	PenSet	20	10	0
101	ParkerPen	60	10	0
102	180pg Note	24	10	0
103	80pg Note	10	25	0
104	StickFile	10	20	0

SQL> create or replace function pur(itmcd number) return number is
2 qt number;
3 pr number;
4 rate number;
5 begin
6 select price,quantity into pr,qt from purchase where icode=itmcd;
7 rate:=qt*pr;
8 return rate;
9 end;

Function created.

SQL> select pur(102) from dual;

PUR(102)

240

Ex:8(b) **CURSOR**

Date:

SQL> create table ssss(cname varchar2(20),sal number);

Table created.

SQL> desc Sample;

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CNAME                                              VARCHAR2(20)
 SAL                                                NUMBER
```

SQL> select * from sample;

```
CNAME                SAL
-------------------- ----------
viji                 17000
sree                  12000
santhi                30000
babu                  20000
```

```
SQL>  declare
  2    cursor emp_cur is select cname,sal from ssss;
  3    name ssss.cname%type;
  4    salary ssss.sal%type;
  5    begin
  6   open emp_cur;
  7    dbms_output.put_line('name salary');
  8    dbms_output.put_line('-----------');
  9    loop
 10     fetch emp_cur into name,salary;
 11     exit when(emp_cur%notfound);
 12    dbms_output.put_line(name||'          '||salary);
 13   end loop;
 14    close emp_cur;
 15    end;

name salary
-----------
viji          17000
sree           12000
santhi          30000
babu           20000

PL/SQL procedure successfully completed.
```

```
SQL>  CREATE OR REPLACE FUNCTION FIND
  2   (cnam  in varchar2)
  3   RETURN number
  4   is
```

```
  5  cid number;
  6  cursor c1
  7  is
  8  select cno from ttt where cname=cnam;
  9  begin
 10   open c1;
 11   fetch c1 into cid;
 12  if c1%notfound then
 13  cid:=9999;
 14  end if;
 15  close c1;
 16  return cid;
 17  end;
```

Function created.

SQL> select * from ttt;

```
       CNO CNAME                MARK
---------- -------------------- ----------
         1 cts                  99
         2 tcs                  55
         3 ibm                   89
```

SQL> select FIND('tcs') from dual;

```
FIND('TCS')
-----------
          2
```

SQL> select FIND('ibm') from dual;

```
FIND('IBM')
-----------
          3
```

SQL> select FIND('hcl') from dual;

```
FIND('HCL')
-----------
       9999
```

TRIGGER WITH BEFORE UPDATE

TABLE

SQL> create table orders(order_id number(5),quantity number(4),cost_per_item number(6,2),total_cost number(8,2),updated_date date,updated_by varchar2(10));

Table created.

INSERT

SQL> insert into orders(order_id,quantity,cost_per_item)
values(&order_id,&quantity,&cost_per_item);

Enter value for order_id: 1
Enter value for quantity: 4
Enter value for cost_per_item: 20
old 1: insert into orders(order_id,quantity,cost_per_item) values(&order_id,&quantity,&cost_per_it
new 1: insert into orders(order_id,quantity,cost_per_item) values(1,4,20)

1 row created.

SQL> /
Enter value for order_id: 2
Enter value for quantity: 5
Enter value for cost_per_item: 30
old 1: insert into orders(order_id,quantity,cost_per_item) values(&order_id,&quantity,&cost_per_it
new 1: insert into orders(order_id,quantity,cost_per_item) values(2,5,30)

1 row created.

SQL> /
Enter value for order_id: 3
Enter value for quantity: 6
Enter value for cost_per_item: 25
old 1: insert into orders(order_id,quantity,cost_per_item) values(&order_id,&quantity,&cost_per_it
new 1: insert into orders(order_id,quantity,cost_per_item) values(3,6,25)

1 row created.

SQL> select * from orders;

ORDER_ID	QUANTITY	COST_PER_ITEM	TOTAL_COST	UPDATED_D	UPDATED_BY
1	4	20			
2	5	30			
3	6	25			

TRIGGER SCRIPT

SQL> create or replace trigger orders_before_update
2 before update
3 on orders
4 for each row
5 declare
6 v_username varchar2(10);
7 begin
8 select user into v_username from dual;
9 :new.updated_date:=sysdate;
10 :new.updated_by:=v_username;
11 end;

Trigger created.

SQL> update orders set total_cost=3000 where order_id=2;

1 row updated.

SQL> select * from orders;

ORDER_ID	QUANTITY	COST_PER_ITEM	TOTAL_COST	UPDATED_D	UPDATED_BY
1	4	20			
2	5	30	3000	19-SEP-07	CSE3101
3	6	25			

TRIGGER WITH AFTER UPDATE

TABLE

SQL> create table orders30(order_id number(5),quantity number(4),cost_per_item number(6,2),total_cost number(8,2));
Table created.

SQL> create table orders_audit(order_id number,quantity_before number,quantity_after number,username varchar2(20));
Table created.

SQL> insert into orders30(order_id,quantity,cost_per_item)
values(&order_id,&quantity,&cost_per_item);
Enter value for order_id: 100
Enter value for quantity: 5
Enter value for cost_per_item: 10
old 1: insert into orders30(order_id,quantity,cost_per_item) values(&order_id,&quantity,&cost_per_
new 1: insert into orders30(order_id,quantity,cost_per_item) values(100,5,10)
1 row created.

SQL> /
Enter value for order_id: 101
Enter value for quantity: 4
Enter value for cost_per_item: 20
old 1: insert into orders30(order_id,quantity,cost_per_item) values(&order_id,&quantity,&cost_per_
new 1: insert into orders30(order_id,quantity,cost_per_item) values(101,4,20)
1 row created.

SQL> /
Enter value for order_id: 102
Enter value for quantity: 5
Enter value for cost_per_item: 30
old 1: insert into orders30(order_id,quantity,cost_per_item) values(&order_id,&quantity,&cost_per_
new 1: insert into orders30(order_id,quantity,cost_per_item) values(102,5,30)
1 row created.

SQL> create or replace trigger orders_after_update
2 AFTER UPDATE
3 ON orders30
4 for each row
5 declare
6 v_username varchar2(10);

```
 7  begin
 8  select user into v_username
 9  from dual;
10  insert into orders_audit
11  (order_id,
12  quantity_before,
13  quantity_after,
14  username)
15  values
16  (:new.order_id,
17  :old.quantity,
18  :new.quantity,
19  v_username);
20  end;
Trigger created.

SQL> update orders30 set quantity=25 where order_id=101;
1 row updated.

SQL> select *from orders_audit;
  ORDER_ID QUANTITY_BEFORE QUANTITY_AFTER        USERNAME
---------- --------------- -------------- ------------------------------------------------------------
       101               4              25                                CSE3090
```

Ex.No:9 **DATABASE CONNECTIVITY WITH PYTHON**
Date:

The following are the steps used to connect oracle (local machine) with python. For this we need to install a package called **cx_Oracle**

STEP 1:
```
import cx_Oracle
```

If the step throws trace back error, the oracle package has not been installed properly

STEP 2: **Assigning the user id and password for connection**
```
con=cx_Oracle.connect("system/cse@127.0.0.1")
```

STEP 3: **Checking for the python connection**
```
print(con.version);
```

STEP 4:**Cursor creation for entire Db**
```
cur=con.cursor()
```

STEP 5: **Query execution**
```
cur.execute(“create table aaa(name char(10),id number(10))”)
cur.execute("insert into aaa values(‘anu’,45)");
cur.execute("select * from aaa");
```

STEP 6: **Fetching the data from table**
```
print(cur.fetchmany()); //To fetch many values
print(cur.fetchall()); // To fetch all the values
```

STEP 7: **Commit the data here**
```
con.commit()
```

STEP 8: **Closing the connection**
```
con.close()
```

Ex.No.9 **Working with advanced databases (Basic Operations)**
Date: **No SQL**

MongoDB is a NoSQL database. There are different types of NoSQL databases, so to be specific MongoDB is an open source document based NoSQL database.

NoSQL databases are different than relational databases like MQSql. In relational database we need to create the table, define schema, set the data types of fields etc before you can actually insert the data. In NoSQL we can directly insert, delete and update.

Install MongoDB on Windows
Step 1: Go to https://www.mongodb.com/download-center#atlas download as shown in the screenshot. A .msi file like this **mongodb-win32-x86_64-2008plus-ssl-3.4.7-signed** will be downloaded in your system. Double click on the file to run the installer.
Step 2: Click Next when the MongoDB installation windows pops up.
Step 3: Accept the MongoDB user Agreement and click Next.
Step 4: When the setup asks you to choose the Setup type, choose Complete.
Step 5: Click Install to begin the installation.
Step 6: Click Finish once the MongoDB installation is complete.
MongoDB Configuration
Step 1: Locate the folder where you have installed MongoDB. If you have followed the above steps then you can find the folder at this location:
C:\Program Files\MongoDB
Here you need to create couple of folders that we need for MongoDB configuration.
1. Create two folders here, name them **data** and **log**.
2. Create another folder inside **data** and name it as **db**, that's where all the data will be stored. That's it close the window.
Step 2: Open command prompt (right click and run as administrator). Navigate to the **bin** folder of MongoDB as shown in the screenshot. The path to the bin folder may be different in your case based on where you have installed the MongoDB.
Step3 : open command prompt type mongodb
Step4 : open another command prompt and type mongo .

```
> show dbs
admin  0.000GB
local  0.000GB
```
We are creating a database "beginnersbook" so the command should be:

```
>use beginnersbook
```
creating a collection **user** and inserting a document in it.

```
> db.user.insert({name: "Chaitanya", age: 30})
WriteResult({ "nInserted" : 1 })

> show dbs
admin         0.000GB
beginnersbook  0.000GB
local         0.000GB
```

Creating the Collection in MongoDB
db.collection_name.insert({key:value, key:value…})

```
> use beginnersbookdb
switched to db beginnersbookdb

db.beginnersbook.insert({
  name: "Chaitanya",
  age: 30,
  website: "beginnersbook.com"
})
```

To check whether the document is successfully inserted, type the following command. It shows all the documents in the given collection.
Syntax: **db.collection_name.find()**

```
> db.beginnersbook.find()

{ "_id" : ObjectId("59bcb8c2415346bdc68a0a66"), "name" : "Chaitanya",
 "age" : 30, "website" : "beginnersbook.com" }
```

This command shows the list of all the collections in the currently selected database.

```
> show collections

beginnersbook
```

To drop a collection , first connect to the database in which you want to delete collection and then type the following command to delete the collection:

```
>db.collection_name.drop()
```

For example I want to delete a collection names “teachers” in my database “beginnersbook.com”. To do this I would write the following commands in the given sequence.

```
> use beginnersbookdb
switched to db beginnersbookdb

> show collections
beginnersbook
students
teachers

> db.teachers.drop()
true

> show collections
beginnersbook
students
```

Updating Document using update () method

Syntax:
db.collection_name.update(criteria, update_data)

Example:
For example: Lets say I have a collection named “got” in the database “beginnersbookdb”. The documents inside “got” are:

```
> db.got.find().pretty()
{
	"_id" : ObjectId("59bd2e73ce524b733f14dd65"),
	"name" : "Jon Snow",
	"age" : 32
}
{
	"_id" : ObjectId("59bd2e8bce524b733f14dd66"),
	"name" : "Khal Drogo",
	"age" : 36
}
{
	"_id" : ObjectId("59bd2e9fce524b733f14dd67"),
	"name" : "Sansa Stark",
	"age" : 20
}
{
	"_id" : ObjectId("59bd2ec5ce524b733f14dd68"),
	"name" : "Lord Varys",
	"age" : 42
}
```

Now suppose if I want to update the name of Jon Snow with the name “Kit Harington”. The command for this would be:

```
db.got.update({"name":"Jon Snow"},{$set:{"name":"Kit Harington"}})
```

Syntax of remove() method:

```
>db.collection_name.remove(delete_criteria)
```

Delete Document using remove() method

Lets say I have a collection `students` in my MongoDB database named `beginnersbookdb`. The documents in `students` collection are:

```
> db.students.find().pretty()
{
	"_id" : ObjectId("59bcecc7668dcce02aaa6fed"),
	"StudentId" : 1001,
	"StudentName" : "Steve",
	"age" : 30
}
```