

List of experiments in DBMS**SQL**

1. Queries for Creating, Dropping, and Altering Tables, Views, and Constraints [CO1]
2. Queries to facilitate acquaintance of Built-In Functions, String Functions, Numeric Functions, Date Functions and Conversion Functions. [CO1]
3. Queries using operators in SQL [CO2]
4. Queries to Retrieve and Change Data: Select, Insert, Delete, and Update [CO2]
5. Queries using Group By, Order By, and Having Clauses [CO2]
6. Queries on Controlling Data: Commit, Rollback, and Save point [CO2]
7. Queries to Build Report in SQL *PLUS [CO2]
8. Queries on Joins and Correlated Sub-Queries [CO2]
9. Queries on Working with Index, Sequence, Synonym, Controlling Access, and Locking Rows for Update, Creating Password and Security features [CO2]

PL/SQL

1. Write a PL/SQL Code using Basic Variable, Anchored Declarations, and Usage of Assignment Operation [CO3]
2. Write a PL/SQL Code Bind and Substitution Variables. Printing in PL/SQL [CO3]
3. Write a PL/SQL block using SQL and Control Structures in PL/SQL [CO3]
4. Write a PL/SQL Code using Cursors, Exceptions and Composite Data Types [CO3]
5. Write a PL/SQL Code using Procedures, Functions, and Packages FORMS [CO4]
6. Write a PL/SQL Code Creation of forms for any Information System such as Student Information System, Employee Information System etc. [CO4]
7. Demonstration of database connectivity [CO4]

SQL**1. Queries for creating , Dropping, and Altering Tables , Views, and Constraints****CREATE A SCHEMA FOR SAILORS RELATION**

```
SQL> CREATE TABLE SAILORS (
    SID NUMBER,
    SNAME VARCHAR2 (25),
    RATING NUMBER,
    AGE REAL,
    CONSTRAINT SID_CON PRIMARY KEY (SID)
);
```

CREATE AN INSTANCE FOR SAILORS RELATION

```
SQL> SELECT * FROM SAILORS;
```

SID	SNAME	RATING	AGE
22	DUSTIN	7	45
29	BRUTUS	1	33
31	LUBBER	8	55.5
32	ANDY	8	25.5
58	RUSTY	10	35
64	HORATIO	7	35
71	ZORBA	10	16

74 HORATIO	9	35
85 ART	3	25.5
95 BOB	3	63.5

10 rows selected.

CREATE A SCHEMA FOR BOATS RELATION

```
SQL> CREATE TABLE BOATS (
    BID NUMBER,
    BNAME VARCHAR2 (25),
    COLOR VARCHAR2 (25),
    CONSTRAINT BID_CON PRIMARY KEY (BID)
);
```

CREATE AN INSTANCE FOR BOATS RELATION

```
SQL> SELECT * FROM BOATS;
```

BID	BNAME	COLOR
101	INTERLAKE	BLUE
102	INTERLAKE	RED
103	CLIPPER	GREEN
104	MARINE	RED

CREATE A SCHEMA FOR RESERVERS RELATION

```
SQL> CREATE TABLE RESERVES (
    SID NUMBER,
    BID NUMBER,
    DAY DATE,
    CONSTRAINT SID_CON PRIMARY KEY (SID),
    FOREIGN KEY (SID) REFERENCES SAILORS (SID),
    FOREIGN KEY (BID) REFERENCES BOATS(BID)
);
```

CREATE AN INSTANCE FOR RESERVES RELATION

```
SQL> SELECT * FROM RESERVES;
```

SID	BID	DAY
22	101	10-OCT-98
22	102	10-OCT-98
22	103	10-AUG-98
22	104	10-JUL-98
31	102	11-NOV-98
31	103	11-JUN-98
31	104	11-DEC-98
64	101	09-MAY-98
64	102	09-AUG-98
74	103	09-AUG-98

DROPPING TABLE SYNTAX:

TABLE DROPPED

ALTER TABLE statement is a powerful statement to add, manage or update table structure.

ALTER TABLE Statement to you can do following thing,

- SQL TABLE RENAME
- ADD NEW COLUMN IN TABLE
- MODIFY EXISTING COLUMN IN TABLE
- RENAME COLUMN IN TABLE
- DROP THE EXISTING COLUMN IN TABLE

SYNTAX:

ALTER TABLE table_name

RENAME TO new_table_name;

SQL>ALTER TABLE user_info RENAME TO user_info;

Table altered.

SQL>ALTER TABLE user_info

ADD (city VARCHAR2(30),

country VARCHAR2(30)

);

Table altered.

Creation of Views:-

Syntax:-

```
CREATE VIEW viewname AS
SELECT columnname, columnname
FROM tablename
WHERE columnname=expression_list;
```

Renaming the columns of a view:-

Syntax:-

```
CREATE VIEW viewname AS  
SELECT newcolumnname...  
FROM tablename  
WHERE columnname=expression_list;
```

Selecting a data set from a view-

Syntax:-

```
SELECT columnname, columnname  
FROM viewname  
WHERE search condition;
```

Destroying a view-

Syntax:-

DROP VIEW viewname;

Type of SQL Constraints

- **PRIMARY KEY**: value in specified column must be unique for each row in a table and not a NULL. Primary key used to identify individual records.
- **FOREIGN KEY**: value in specified column must have reference in another table (That existing record have primary key or any other constraint).
- **NOT NULL**: Column value must not be a NULL.
- **UNIQUE**: Check column value must be unique across the given field in table.
- **CHECK**: Specific condition is specified, which must evaluate to true for constraint to be satisfied.
- **DEFAULT**: Default value assign if none of the value specified of given field.
- Syntax:

```
ALTER TABLE table_name
DROP constraint_namecolumn_name;
SQL>CREATETABLEemp_info(
    no NUMBER(3,0),
    name VARCHAR(30),
    address VARCHAR(70),
    contact_no VARCHAR(12),
    PRIMARYKEY(no)
);
```

Table created.

```
SQL>CREATETABLEemp_info(
no NUMBER(3,0)PRIMARYKEY,
nameVARCHAR(30),
addressVARCHAR(70),
contact_no NUMBER(12,0)
);
```

Table created.

```
SQL>CREATETABLEemp_salary(
no NUMBER(3,0)PRIMARYKEY,
users_no NUMBER(3,0),
salary NUMBER(12),
CONSTRAINTfk_usernoFOREIGNKEY(users_no)REFERENCESemp_info(no)
);
```

Table created.**2. Queries to facilitate acquaintance of Built-In Functions, String Functions, Numeric Functions, Date Functions and Conversion Functions.****SQL FUNCTIONS:**

SQL Functions are used to perform calculations on data. Manipulate output from groups of rows. It can also format date members for display. It can also be used for modifying individual data items. SQL function sometimes takes arguments and always returns value.

There are two distinct types of functions:

1. Single Row functions
2. Multiple Row functions

Single Row functions:

Single row functions operate on single rows only and return one result per row.

The different type single row functions are

2. Character functions
3. Number Functions
4. Date functions.
5. Conversion Function
6. General Functions

1.Character Functions:

Upper	Returns char with all letters into upper case
lower	Converts the mixed case or uppercase character strings to lowercase
Initcap	Converts the first letter of each word to upper case and remaining letters to lowercase
Concat	Joins values together you are limited to two arguments with concat
Substr	This extracts a string of determined length
Length	Shows the length of a string as a numeric value
Instr	Finds numeric position of named character

Lpad	Pads the character value right justified
rpad	Pads the character value left justified
Trim	Trims heading or trailing characters from a character string
Raplace	To replace a set of character (String based)
Translat e	Change a character to a new described character(character based)

```
SQL> select upper ('oracle') "UPPER" from dual;
UPPER
```

```
-----
ORACLE
```

```
SQL> select lower ('ORACLE') "LOWER" from dual;
LOWER
```

```
-----
Oracle
```

```
SQL> select initcap('sql functions') "initcap" from dual;
initcap
```

```
-----
Sql Functions
```

```
SQL> select concat('sql','functions') from dual;
CONCAT('SQL'
```

```
-----
Sqlfunctions
```

```
SQL> select substr('sqlfunctions',1,5) from dual;
SUBST
```

```
-----
sqlfu
```

```
SQL> select substr('sqlfunctions',2,5) from dual;
SUBST
```

```
-----
qlfun
```

```
SQL> select substr('sqlfunctions',4,5) from dual;
SUBST
```

```
-----
funct
```

```
SQL> select length('sqlfunctions') from dual;
```

```
LENGTH('SQLFUNCTIONS')
```

```
-----
                12
```

```
SQL> select instr('sqlfunctions','f') from dual;
INSTR('SQLFUNCTIONS','F')
```

```
-----
                4
```

```
SQL> select lpad(sal,15,'*') from emp;
LPAD(SAL,15,'*')
```

```
-----
*****800
*****1600
*****1250
*****2975
*****1250
*****2850
*****2450
*****3000
*****5000
*****1500
*****1100
*****950
*****3000
*****1300
```

```
SQL> select rpad(sal,15,'*') from emp;
RPAD(SAL,15,'*')
```

```
-----
800*****
1600*****
1250*****
2975*****
1250*****
2850*****
2450*****
3000*****
5000*****
1500*****
1100*****
950*****
```


3000*****
1300*****

SQL> select trim('s' from 'ssmiths') from dual;
TRIM

mith

SQL> select ltrim('ssmiths','s') from dual;
LTRIM

miths

SQL> select rtrim('ssmiths','s') from dual;
RTRIM(

ssmith

SQL> select replace('jack and jue','j','bl') from dual;
REPLACE('JACKA

black and blue

SQL> select translate('jack','j','b') from dual;
TRAN

----BACK

Number Functions:

Round	Rounds the value to specified decimal
Trunc	Truncates the column, expression, or value to n decimal places
Power	Calculates the power of the given value
Mod	Finds the remainder of value1 divided by value1
Ceil	Takes the height decimal value
Floor	Takes the lowest decimal value

SQL> select round(35.823,2), round(35.823,0), round(35.823,-1) from dual;
ROUND(35.823,2) ROUND(35.823,0) ROUND(35.823,-1)

35.82 36 40

SQL> select trunc(35.823,2), trunc(35.823), trunc(35.823,-2) from dual;
TRUNC(35.823,2) TRUNC(35.823) TRUNC(35.823,-2)

```
-----
35.82      35      0
```

```
SQL> select mod(5,2) from dual;
MOD(5,2)
```

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

```
SQL> select mod(sal,2000) from emp where job like 'SALESMAN';
MOD(SAL,2000)
```

```
-----
1600
1250
1250
1500
```

```
SQL> select sal from emp where job like 'SALESMAN';
SAL
```

```
-----
1600
1250
1250
1500
```

```
SQL> select ceil(35.23), ceil(35.5), ceil(35.6) from dual;
```

```
CEIL(35.23) CEIL(35.5) CEIL(35.6)
```

```
-----
36      36      36
```

```
SQL> select floor(35.23), floor(35.5), floor(35.6) from dual;
```

```
FLOOR(35.23) FLOOR(35.5) FLOOR(35.6)
```

```
-----
35      35      35
```

Date Functions:

SYSDATE is a pseudo column that returns the current date and time. When we select sysdate it will display in a dummy table called DUAL. Oracle date range between 1st Jan 4712 BC and 31st Dec 4712 AD.

Months_between	It returns the numeric value. Finds the no. of months between date1 and date2, result may be positive or negative.
Add_months	It returns the date datatype. Adds n number of calendar months to date, n must be an integer and it can be negative
Last_day	It returns the date datatype. Date of the

Next_day	It returns the date datatype. Date of the next specified day of the week following date1, char may be number representing a day, or a character
----------	---

SQL> select sysdate from dual;

SYSDATE

08-JUL-10

SQL> select months_between(sysdate, hiredate) from emp;

MONTHS_BETWEEN(SYSDATE,HIREDATE)

354.728983
352.632208
352.567692
351.212854
345.374144
350.245112
348.987047
278.664466
343.728983
346
277.535434
343.180596
343.180596
341.535434

14 rows selected.

SQL> select months_between('01-jan-2010', sysdate) from dual;

MONTHS_BETWEEN('01-JAN-2010',SYSDATE)

-6.2451325

SQL> select last_day(sysdate) from dual;

LAST_DAY(

31-JUL-10

SQL> select last_day(hiredate),last_day('15-feb-88') from emp;

LAST_DAY(LAST_DAY(

31-DEC-80 29-FEB-88
28-FEB-81 29-FEB-88
28-FEB-81 29-FEB-88
30-APR-81 29-FEB-88

30-SEP-81 29-FEB-88
 31-MAY-81 29-FEB-88
 30-JUN-81 29-FEB-88
 30-APR-87 29-FEB-88
 30-NOV-81 29-FEB-88
 30-SEP-81 29-FEB-88
 31-MAY-87 29-FEB-88
 31-DEC-81 29-FEB-88
 31-DEC-81 29-FEB-88
 31-JAN-82 29-FEB-88

Sunday	1
Monday	2
Tuesday	3
Wednesday	4
Thursday	5
Friday	6
Saturday	7

SQL> select last_day(hiredate),last_day('15-feb-88') from emp;
 LAST_DAY(LAST_DAY(

 31-DEC-80 29-FEB-88
 28-FEB-81 29-FEB-88
 28-FEB-81 29-FEB-88
 30-APR-81 29-FEB-88
 30-SEP-81 29-FEB-88
 31-MAY-81 29-FEB-88
 30-JUN-81 29-FEB-88
 30-APR-87 29-FEB-88
 30-NOV-81 29-FEB-88
 30-SEP-81 29-FEB-88
 31-MAY-87 29-FEB-88
 31-DEC-81 29-FEB-88
 31-DEC-81 29-FEB-88
 31-JAN-82 29-FEB-88

14 rows selected.

SQL> select next_day(sysdate, 'friday') from dual;
 NEXT_DAY(

 09-JUL-10

```
SQL> select next_day(hiredate,'friday'), next_day(hiredate,6) from emp where deptno=10;
NEXT_DAY( NEXT_DAY(
```

```
-----
12-JUN-81 12-JUN-81
20-NOV-81 20-NOV-81
29-JAN-82 29-JAN-82
```

Month mid value	1-15
Day mid value	Sunday
Year mid value	30-jun

```
SQL> select round(sysdate,'day') from dual;
ROUND(SYS
```

```
-----
11-JUL-10
```

```
SQL> select round(sysdate,'year') from dual;
ROUND(SYS
```

```
-----
01-JAN-11
```

```
SQL> select round(sysdate,'month') from dual;
ROUND(SYS
```

```
-----
01-JUL-10
```

```
SQL> select trunc(sysdate,'month'), trunc(sysdate,'year') from dual;
TRUNC(SYS TRUNC(SYS
```

```
-----
01-JUL-10 01-JAN-10
```

Conversion Functions:

To_char(number date,['fmt'])	Converts numbers or date to character format fmt
To_number(char)	Converts char, which contains a number to a NUBER
To_date	Converts the char value representing date, into a date value according to fmt specified. If fmt is omitted, format is DD-MM-YYYY

```
SQL> select to_char(3000, '$9999.99') from dual;
```

```
TO_CHAR(3
```

```
-----
```

```
$3000.00
```

```
SQL> select to_char(sysdate, 'fmday, ddth month yyyy') from dual;
```

```
TO_CHAR(SYSDATE,'FMDAY,DDTHMON
```

```
-----
```

```
thursday, 8th july 2010
```

```
SQL> select to_char(sysdate, 'hh:mi:ss') from dual;
```

```
TO_CHAR(
```

```
-----
```

```
03:04:27
```

```
SQL> select to_char(sal, '$9999.99') from emp;
```

```
TO_CHAR(S
```

```
-----
```

```
$800.00
```

```
$1600.00
```

```
$1250.00
```

```
$2975.00
```

```
$1250.00
```

```
$2850.00
```

```
$2450.00
```

```
$3000.00
```

```
$5000.00
```

```
$1500.00
```

```
$1100.00
```

```
$950.00
```

```
$3000.00
```

```
$1300.00
```

```
SQL> select empno,ename, job,sal from emp where sal>to_number('1500');
```

```
EMPNO ENAME JOB SAL
```

```
-----
```

```
7499 ALLEN SALESMAN 1600
```

```
7566 JONES MANAGER 2975
```

```
7698 BLAKE MANAGER 2850
```

```

7782 CLARK    MANAGER    2450
7788 SCOTT    ANALYST     3000
7839 KING     PRESIDENT   5000
7902 FORD     ANALYST     3000

```

SQL> update emp set hiredate=to_date('1998 05 20', 'yyyy mm dd') where ename='SMITH';

1 row updated.

General Functions:

Uid	This function returns the integer value corresponding to the user currently logged in
User	This function returns the login user name, which is in varchar2 datatype
Nvl	This function is used in case where we want to consider null values
Vsize	This function returns the number of bytes in the expression, if expression is null it returns zero.
Case	Case expression let you use IF-THEN-ELSE logic in SQL statements without having invoke procedures
Decode	Decodes and expression in a way similar IF-THEN-ELSE logic. Decodes and expression after comparing it to each search value.

SQL> select uid from dual;

```

      UID
-----
      59

```

SQL> select user from dual;

```

      USER
-----
      SCOTT

```

SQL> select ename, nvl(comm,0) from emp;

```

ENAME    NVL(COMM,0)
-----
SMITH          0
ALLEN        300
WARD         500
JONES          0
MARTIN       1400
BLAKE          0
CLARK          0
SCOTT          0

```

KING	0
TURNER	0
ADAMS	0
JAMES	0
FORD	0
MILLER	0

```
SQL> select vsize('hello') from dual;
VSIZE('HELLO')
```

```
-----
5
```

```
SQL> select vsize(ename) from emp;
VSIZE(ENAME)
```

```
-----
5
5
4
5
6
5
5
5
4
6
5
5
4
6
```

```
SQL> select ename,job,sal ,
case job when 'CLERK' then 1.10*sal
when 'MANAGER' then 1.15*sal
elsesal end "revised salary" from emp;
```

ENAME	JOB	SAL	revised salary
SMITH	CLERK	800	880
ALLEN	SALESMAN	1600	1600
WARD	SALESMAN	1250	1250
JONES	MANAGER	2975	3421.25
MARTIN	SALESMAN	1250	1250
BLAKE	MANAGER	2850	3277.5
CLARK	MANAGER	2450	2817.5
SCOTT	ANALYST	3000	3000
KING	PRESIDENT	5000	5000
TURNER	SALESMAN	1500	1500
ADAMS	CLERK	1100	1210
JAMES	CLERK	950	1045


```
FORD    ANALYST    3000    3000
MILLER  CLERK       1300    1430
```

```
SQL> select ename,job,sal ,
decode(job,'CLERK',1.10*sal,'MANAGER',1.15*sal,'SALESMAN',1.20*sal,sal) "revised salary" from
emp;
```

```
ENAME      JOB          SAL revised salary
-----
SMITH      CLERK          800          880
ALLEN      SALESMAN       1600         1920
WARD       SALESMAN       1250         1500
JONES      MANAGER        2975        3421.25
MARTIN     SALESMAN       1250         1500
BLAKE      MANAGER        2850        3277.5
CLARK      MANAGER        2450        2817.5
SCOTT      ANALYST        3000         3000
KING       PRESIDENT      5000         5000
TURNER     SALESMAN       1500         1800
ADAMS      CLERK          1100         1210
JAMES      CLERK          950          1045
FORD       ANALYST        3000         3000
MILLER     CLERK          1300         1430
```

Multiple Row functions:

A group function returns a result based on a group of rows. Some of these are just purely mathematical functions. This group function operate on sets of rows of rows to give one result per group. These sets may be the whole table or the table split into groups.

Sum	To obtain the sum of a range of values of a record set
Avg	This function will return the average of values of the column specified in the argument of column
Min	This function will give the least value of all values of the column present in the argument.
Max	This function will give the maximum value of all values of the column present in the argument.
Count	This function will return the number of rows contained to the related column

```
SQL> select sum(sal) from emp;
SUM(SAL)
```

```
-----
29025U
```

```
SQL> select avg(Sal) from emp;
```

AVG(SAL)

2073.21429

SQL> select min(sal) from emp;

MIN(SAL)

800

SQL> select max(sal) from emp;

MAX(SAL)

5000

SQL> select count(*) from emp;

COUNT(*)

14

3.Queries using operators in SQL

FIND THOSE EMPLOYEES WHOSE COMMISSION IS GREATER THAN THEIR SALARY.

SQL> SELECT ENO,ENAME,SAL,COMM FROM EMP WHERE SAL<NVL(COMM,0);

ENO	ENAME	SAL	COMM
7654	MARTIN	1252	1400

DISPLAY THOSE EMPLOYEES WHOSE SALARY IS BETWEEN 1000 AND 2000.

SQL> SELECT * FROM EMP WHERE SAL BETWEEN 1000 AND 2000;

ENO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DNO
7654	MARTIN	SALESMAN	7698	28-SEP-81	1252	1400	
7499	ALLEN	SALESMAN	7698	20-FEB-81	1602	300	

LIST ALL EMPLOYEES WHOSE NAME WITH S.

SQL> SELECT * FROM EMP WHERE ENAME LIKE 'S%';

ENO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DNO
7369	SMITHH	CLERK	7902	17-DEC-80	802	20	200

LIST ALL EMPLOYEES WHO HAVE NAME EXACTLY 4 CHARACTERS IN LENGTH.

SQL> SELECT ENO,ENAME FROM EMP WHERE LENGTH(RTRIM(ENAME))=4;

ENO	ENAME
7839	KING
7521	WARD
7902	FORD

4. Queries to Retrieve and Change Data: Select, Insert, Delete , and Update

1) Find the names of sailors who have reserved boat number 103.

SQL> select s.sname from sailors s, reserves r where s.sid=r.sid and r.bid=103;

SNAME

DUSTIN
LUBBER
HORATIO

2) Find the names of sailors who have reserved a red boat.

SQL> select distinct s.sname from sailors s,reserves r, boats b where s.sid=r.sid and r.bid=b.bid and b.color='RED';

SNAME

DUSTIN
DUSTIN
LUBBER
LUBBER
HORATIO

3) Find the names and ages of all sailors;

SQL> select sname, age from sailors;

SNAME	AGE
DUSTIN	45
BRUTUS	33
LUBBER	55.5
ANDY	25.5

RUSTY	35
HORATIO	35
ZORBA	16
HORATIO	35
ART	25.5
BOB	63.5

10 rows selected.

SQL> select distinct s.sname, s.age from sailors s; /* With distinct clause */

SNAME	AGE
ANDY	25.5
ART	25.5
BOB	63.5
BRUTUS	33
DUSTIN	45
HORATIO	35
LUBBER	55.5
RUSTY	35
ZORBA	16

4) Find all sailors with a rating above 7

SQL> select s.sid,s.sname,s.rating,s.age from sailors s where s.rating>7;

SID	SNAME	RATING	AGE
31	LUBBER	8	55.5
32	ANDY	8	25.5
58	RUSTY	10	35
71	ZORBA	10	16
74	HORATIO	9	35

5) Find the names of boats reserved by lubber

SQL> select b.color from sailors s, boats b, reserves r
 wheres.sid=r.sid and r.bid=b.bid and s.sname=upper('lubber');
 COLOR

 RED
 GREEN
 RED

6) Find the names of sailors who have reserved at least one boat.

SQL> select distinct s.sname from sailors s, reserves r where s.sid=r.sid;

SNAME

DUSTIN
HORATIO
LUBBER

7) Compute increments for the ratings of persons who have sailed two different boats on the same day.

SQL> select s.sname,s.rating+1 as rating from sailors s, reserves r1, reserves r2
2 where s.sid=r1.sid and s.sid=r2.sid and r1.day=r2.day and r1.bid<>r2.bid;

SNAME	RATING
DUSTIN	8
DUSTIN	8

8) Find the ages of sailors whose name begins and ends with B and has not at least three characters

SQL> select s.age from sailors s where s.sname like 'B_%B';

AGE
63.5

5.Queries using GroupBy, OrderBy, and HavingClause

Find the age of the youngest sailor who is eligible to vote (age>18) for each rating level with at least two such sailors;

```
SQL> select s.rating,min(s.age) as minage from sailors s
where s.age>18 group by s.rating
having count(*)>1
```

RATING	MINAGE
3	25.5
7	35
8	25.5

For each boat, find the number of reservations for this boat

```
SQL> select b.bid,count(*) as reservationcount from boats b, reserves r
2 where r.bid=b.bid and b.color='RED'
3 group by b.bid;
```

BID	RESERVATIONCOUNT
102	3
104	2

Find the average age of sailors for each rating level that has at least two sailors

```
SQL> select s.rating,avg(s.age) as average from sailors s
2 group by s.rating
3 having count(*)>1;
```

RATING	AVERAGE
3	44.5
7	40
8	40.5
10	25.5

6.Queries on Controlling Data: Commit, Rollback, and Savepoint

Commit command is used to mark the changes as permanent.

Commit command's syntax

commit;

Save Point command is used to temporarily save a transaction so that you can roll back to that point whenever required.

savepoint command's syntax

Savepoint savepoint_name;

```
SQL>CREATE TABLE emp_data(
no NUMBER(3),
name VARCHAR(50),
code VARCHAR(12)
);
```

Table created.

```
SQL>SAVEPOINT table_create;
```

Savepoint created.

```
SQL>insert into emp_data VALUES(1,'Opal','e1401');
```

1 row created.

```
SQL>SAVEPOINT insert_1;
```

Savepoint created.

```
SQL>insert into emp_data VALUES(2,'Becca','e1402');
```

1 row created.

```
SQL>SAVEPOINT insert_2;
```

Savepoint created.

```
SQL>SELECT * FROM emp_data;
```

NO	NAME	CODE
1	Opal	e1401
2	Becca	e1402

ROLLBACK command execute at the end of current transaction and undo/undone any changes made since the begin transaction.

ROLLBACK[To SAVEPOINT_NAME];

SQL>ROLLBACKTO insert_1;

Rollback complete.

SQL>SELECT*FROMMemp_data;

NO	NAME	CODE
1	Opal	e1401

7. Queries toBuild Report in SQL*PLUS

The objective of the lab is to create a form using a parent table and a child table to take advantage of the schema's relationships.

A data block in Oracle Forms = A table in the database.

Each block contains items which equal table columns or fields.

These are arranged into records.

1. Start Schema Builder. Open S_Customer and S_Order orS_Order1.
2. Start Form Builder. Use the data block wizard to create a form for S_Customer, including the Name, ID, Phone, and Addresscolumns.
3. After the form is created, click on Window on the Object Navigator to expand it. Right click on Window1. Click on Property Pallet. Go to Physical on property pallet. Make sure Show Horizontal Scroll Bar and Show Vertical Scroll Bar both areYES.
4. Run the form. Execute the Questionry. Notice that data already exists in thistable.
5. Highlight Data Blocks in the Object Navigator. Go up to Tools – Data BlockWizard.
6. Create a form for S_Order orS_Order1.
7. Include the columns ID – Customer_ID – Date_Ordered – Date_Shipped –Total.
8. Click Create Relationship. Click OK. Make sure AutojoinDatablocks ischecked.
9. Check Detail Item to Customer_ID and Master Item to ID. This says that the parent table, the table on the one side of the relationship has the primary key of ID in the S_Customer table, and the foreign key on the many side is Customer_ID in the S_Order table. This relationship can be seen if you open schema builder and look at the tables and the relationship between them.
10. Make the layouttabular.
11. Records displayed will be 5 and Display Scrollbar will be checkedoff.
12. Run the form and execute the Questionry. Scroll through the data and notice that the orders are linked with thecustomers.
- 13.If you input a detail, the foreign key is automatically filled with the value of the current primary key displayed by thecustomer.

8. Queries on Joins and Correlated Sub-Queries

Joint Multiple Table (Equi Join): Some times we require to treat more than one table as though manipulate data from all the tables as though the tables were not separate object but one single entity. To achieve this we have to join tables. Tables are joined on column that have same data type and data with in tables.

The tables that have to be joined are specified in the FROM clause and the joining attributes in the WHERE clause.

Algorithm for JOIN in SQL:

1. Cartesian product of tables (specified in the FROM clause)
2. Selection of rows that match (predicate in the WHERE clause)
3. Project column specified in the SELECT clause.

1. Cartesian product:-

Consider two table student and course

```
Select B.*,P.*
FROM student B, course P;
```

2. INNER JOIN:

Cartesian product followed by selection

```
Select B.*,P.*
FROM student B, Course P
WHERE B.course # P.course# ;
```

3. LEFT OUTER JOIN:

LEFT OUTER JOIN = Cartesian product + selection but include rows from the left table which are unmatched with nulls in the values of attributes belonging to the second table

Exam:

```
Select B.*,P*
FROM student B left join course p
ON B.course # P.course #;
```

4. RIGHT OUTER JOIN:

RIGHT OUTER JOIN = Cartesian product + selection but include rows from right table which are unmatched

Exam:

```
Select B.*,P.*
From student B RIGHT JOIN course P
B.course# = P course # ;
```

5. FULL OUTER JOIN

Exam

```
Select B.*,P.*
```

Correlated Sub-Queries & Non Co Related Sub queries:

Note: For Co related and non co related sub queris Refer Class Notes and write down that examples in record.

9. Queries on Working with Index, Sequence, Synonym, Controlling Access, and Locking Rows for Update, Creating Password and Security features.

An index is a performance-tuning method of allowing faster retrieval of records. An index creates an entry for each value that appears in the indexed columns. By default, Oracle creates Btree indexes. Create an Index: The syntax for creating an index in Oracle/PLSQL is:

```
CREATE [UNIQUE] INDEX index_name
ON table_name (column1, column2, ... column_n);
```

SQL> create index AB on emp1(SAL);

Index created.

We could also create an index with more than one field as in the example below:

SQL > create index AB on emp1(SAL,ENAME,EID);

check out index is created or not:

SQL> select INDEX_NAME,INDEX_TYPE from user_indexes where index_name='AB';

INDEX_NAME	INDEX_TYPE
AB	NORMAL

drop index:

drop index index_name;

Index created.

SQL>select sal from emp;

Result: it will return all emp salaries in descending order.

Sequence:

A sequence is a special database object that generates integers according to specified rules at the time the sequence was created. Users select data from them using two special keywords to denote pseudo (or partial or virtual) columns in the database.

```
REATE SEQUENCE sequence_name
START WITH initial_value
INCREMENT BY increment_value
MINVALUE minimum value
MAXVALUE maximum value
CYCLE|NOCYCLE ;
```

SQL> create table class(id number(10),name varchar2(10));

Table created.

SQL> insert into class values(101,'hai');

1 row created.

SQL> insert into class values(102,'hai');

1 row created.

SQL> select * from class;

ID	NAME
101	hai
102	hai

SQL> insert into class values(seq_1.nextval,'anu');

1 row created.

SQL> select * from class;

ID	NAME
101	hai
102	hai
1	anu

SQL> insert into class values(seq_1.nextval,'janu');

1 row created.

SQL> select * from class;

ID	NAME
101	hai
102	hai
1	anu
2	janu

Synonym:

SQL> create or replace synonym ee for emp;

Synonym created.

SQL> select * from ee;

EID	ENAME	SAL
101	abc	2192.2
102	dfe	1092.2
104	fhgd	4092.2
403	cfgdd	992.2

Creating Password and Security features:

Grant Command:

(another window-1):

SQL> conn sys as sysdba;

Enter password:

Connected.

SQL> create user oq identified by oq;

User created.

SQL> grant connect,resource to oq;

Grant succeeded.

SQL> conn oq/oq;

Connected.

Main window:

Conn system/manager;

SQL> grant all on emp4 to oq;

Grant succeeded.

(another window-1):

SQL> select * from system.emp4;

EID	ENAME	SAL
101	abc	2212.2
102	dfe	2112.2
104	fhgd	4112.2
403	cfgdd	1012.2

Main window:

SQL> select * from emp4 where eid=403 for update;

EID	ENAME	SAL
403	cfgdd	1012.2

(another window-1):

SQL> update system.emp4 set sal=sal+100 where eid=403; (cursor is struct)

Go to main window and type **commit;**

Table level lock: (main window)

SQL> lock table emp4 in share mode;

Table(s) Locked.

Another window:

SQL> update system.emp4 set sal=sal+100 where eid=403;

Not Updates(cursor is in inactive mode)

PL/SQL**1. Write a PL/SQL Code using Basic Variable, Anchored Declarations, and Usage of Assignment Operation.****Ex-1:****Sql >** set server output on;

```

SQL> SQL> DECLARE
2   var_num1 number;
3   var_num2 number;
4   var_mult number;
5 BEGIN
6   var_num1 := 100;
7   var_num2 := 200;
8   var_mult := var_num1 * var_num2;
9   dbms_output.put_line(var_mult);
10  end;
11 /

```

PL/SQL procedure successfully completed.

SQL> set serveroutput on;

SQL> /

20000

PL/SQL procedure successfully completed.

Example 2:

```

SQL> DECLARE
sal1 number(6);
id number(6) := 102;
BEGIN
  select sal into sal1 from emp where eid = id;
  dbms_output.put_line(sal1);
  dbms_output.put_line('The employee ' || id || ' has salary ' || sal1);
end;
/

```

SQL> set serveroutput on;

SQL> /

1092

The employee 102 has salary 1092

PL/SQL procedure successfully completed

Example 3:

```

SQL> declare
  name emp.ename %type;

```

```

begin
select ename into name from emp where eid=&no;
dbms_output.put_line(name);
end;
/

```

Enter value for no: 101

old 4: select ename into name from emp where eid=&no;

new 4: select ename into name from emp where eid=102;

dfe

PL/SQL procedure successfully completed.

Example 4:%rowtype

```

declare
cursor c is select * from dept;
rec c%ROWTYPE;
BEGIN
open c;
loop
fetch c into rec;
exit when c%rowcount=4;
Dbms_output.put_line('deptno='||rec.deptno);
Dbms_output.put_line('dname='||rec.dname);
Dbms_output.put_line('loc='||rec.loc);
end loop;
close c;
end;

```

Output:

setserveroutput on

```

>deptno10
dname=ACCOUNTING
loc=NEW YORK
deptno=20
dname=RESEARCH
loc=DALLAS
deptno=30
dname=SALES
loc=CHICAGO
PL/SQL procedure successfully completed.

```

PL/SQL variables must be declared in the declaration section or in a package as a global variable. When you declare a variable, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

The syntax for declaring a variable is –

variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]

Where, *variable_name* is a valid identifier in PL/SQL, *datatype* must be a valid PL/SQL data type or any user defined data type which we already have discussed in the last chapter. Some valid variable declarations along with their definition are shown below –

```
sales number(10, 2);

pi CONSTANT double precision := 3.1415;

name varchar2(25);

address varchar2(100);
```

Whenever you declare a variable, PL/SQL assigns it a default value of NULL. If you want to initialize a variable with a value other than the NULL value, you can do so during the declaration, using either of the following –

- The **DEFAULT** keyword
- The **assignment** operator

For example –

```
counterbinary_integer := 0;

greetings varchar2(20) DEFAULT 'Have a Good Day';
```

You can also specify that a variable should not have a **NULL** value using the **NOT NULL** constraint.

If you use the NOT NULL constraint, you must explicitly assign an initial value for that variable.

It is a good programming practice to initialize variables properly otherwise, sometimes programs would produce unexpected results. Try the following example which makes use of various types of variables –

When the above code is executed, it produces the following result –

```
SQL>DECLARE
```

```
a integer :=10;
```

```
b integer :=20;
```

```
c integer;
```

```
f real;
```


BEGIN

c:= a + b;

dbms_output.put_line('Value of c: '|| c);

f:=70.0/3.0;

dbms_output.put_line('Value of f: '|| f);

END;

/

OUTPUT:

Value of c: 30

Value of f: 23.333333333333333333

PL/SQL procedure successfully completed.

An initialization using the assignment operator (:=)

SQL> -- An initialization using the assignment operator (:=).

SQL> set serverout on;

SQL>

SQL> DECLARE

2 X NUMBER(11,2) := 10;

3

4 BEGIN

5 DBMS_OUTPUT.PUT_LINE(x);

6 END;

7

8 /

Output:

10

PL/SQL procedure successfully completed.

2. Write a PL/SQL Code Bind and Substitution Variables Printing in PL/SQL

These substitutions are carried out by the interface being used. In this example we're going to use SQL*Plus as our interface...

So let's take a bit of code with substitution variables:

```
SQL> create table dept(deptno number(10),dname varchar2(10));
```

Table created.

```
SQL> create or replace function myfn return varchar2 is
    v_dname varchar2(20);
begin
    select dname
    into   v_dname
    from   dept
    where  deptno = &p_deptno;
    return v_dname;
end;
/
```

OUTPUT:

Enter value for p_deptno: 203

old 7: where deptno = &p_deptno;

new 7: where deptno = 203;

Function created.

```
SQL> select dbms_metadata.get_ddl('FUNCTION', 'MYFN', USER) from dual;
DBMS_METADATA.GET_DDL('FUNCTION','MYFN',USER)
```

```
-----
CREATE OR REPLACE FUNCTION "SCOTT"."MYFN" return varchar2 is
    v_dname varchar2(20);
begin
    select dname
    into   v_dname
    from   dept
    where  deptno = 20;
    return v_dname;
end;
```

The database itself knows nothing about any substitution variable... it just has some code, fixed with the value we supplied to SQL*Plus when we compiled it.

The only way we can change that value is by recompiling the code again, and substituting a new value for it.

Also, with substitution variables we don't necessarily have to use them just for 'values' (though that is typically what they're used for)... we can use them to substitute any part of the code/text that we are supplying to be compiled.. e.g.

```
SQL> create or replace function myfn(x in number, y in number) return number is
begin
    return &what_do_you_want_to_return;
```

```
end;
/
```

OUTPUT:

```
Enter value for what_do_you_want_to_return: y*power(x,2)
old 3: return &what_do_you_want_to_return;
new 3: return y*power(x,2);
Function created.
```

```
SQL> select dbms_metadata.get_ddl('FUNCTION', 'MYFN', USER) from dual;
DBMS_METADATA.GET_DDL('FUNCTION','MYFN',USER)
```

```
-----
CREATE OR REPLACE FUNCTION "SCOTT"."MYFN" (x in number, y in number) return
number is
begin
return y*power(x,2);
end;
```

Binding ex :

```
SQL> create or replace function f2(j number)
return varchar2 is
begin
if mod(j,2)=0 then
return 'even';
else
return 'odd';
end if;
end;
/
```

Function created.

Creation of bind variable:

```
SQL> variable z varchar2(10);
SQL> begin
2  :z:=f2(6);
3  end;
4  /
```

Output:

3

PL/SQL procedure successfully completed.

3. Write a PL/SQL block using SQL and Control Structures in PL/SQL**Using IF statement:**

```
SQL>DECLARE
```

```
sales NUMBER(8,2) := 10100;
```

```
quota NUMBER(8,2) := 10000;

bonus NUMBER(6,2);

emp_id NUMBER(6) := 120;

BEGIN

IF sales > (quota + 200) THEN

bonus := (sales - quota)/4;

UPDATE employees SET salary = salary + bonus WHERE employee-id = emp-id;

END IF;

END;
```

Output:

PL/SQL procedure successfully completed.

SQL> select sal from emp;

```
      SAL
-----
    10000
     5000
     2400
     9000
     4000
     6000
    35000
     1000
     1000
      SAL
-----
     2000
```

13 rows selected.

Using CASE Statement:

SQL> DECLARE

grade CHAR(1);

BEGIN

grade := 'B';

CASE grade

```
WHEN 'A' THEN DBMS-OUTPUT.PUT-LINE('Excellent');  
WHEN 'B' THEN DBMS-OUTPUT.PUT-LINE('Very Good');  
WHEN 'C' THEN DBMS-OUTPUT.PUT-LINE('Good');  
WHEN 'D' THEN DBMS-OUTPUT.PUT-LINE('Fair');  
WHEN 'F' THEN DBMS-OUTPUT.PUT-LINE('Poor');  
ELSE DBMS-OUTPUT.PUT-LINE('No such grade');  
  
END CASE;  
  
END;  
OUTPUT:  
Very Good
```

Using Case Statement:

```
SQL>DECLARE
```

```
p NUMBER := 0;
```

```
BEGIN
```

```
FOR k IN 1..500 LOOP -- calculate pi with 500 terms
```

```
p := p + ( ( (-1) ** (k + 1) ) / ((2 * k) - 1) );
```

```
END LOOP;
```

```
p := 4 * p;
```

```
DBMS_OUTPUT.PUT_LINE( 'pi is approximately : ' || p );
```

```
END;
```

Output:

```
pi is approximately : 3.13959265558978323858464061338053947907
```

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

4. Write a PL/SQL Code using Cursors, Exceptions and Composite Data Types

A .Write a PL/SQL cursor to display employee name

```
SQL> declare
cursorempcursor
is
select *
from emp;
v_empdataempcursor%rowtype;
begin
openempcursor;
loop
fetchempcursor into v_empdata;
exit when empcursor%notfound;
dbms_output.put_line(v_empdata.ename);
end loop;
closeempcursor;
end;
/
```

OUTPUT:-

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

PL/SQL procedure successfully completed.

B .Write a PL/SQL cursor to display employee name and display number of records processed.

```
SQL> declare
cursorempcursor is
select * from emp;
v_empdataempcursor%rowtype;
begin
openempcursor;
loop
fetch emp cursor into v_empdata;
```

```

exit when empcursor%notfound;
dbms_output.put_line('Record Number: '||empcursor%rowcount||' '||v_empdata.ename);
end loop;
close empcursor;
end;
/

```

OUTPUT:-

```

RecordNumber: 1 SMITH
RecordNumber: 2 ALLEN
RecordNumber: 3 WARD
RecordNumber: 4 JONES
RecordNumber: 5 MARTIN
RecordNumber: 6 BLAKE
RecordNumber: 7 CLARK
RecordNumber: 8 SCOTT
RecordNumber: 9 KING
RecordNumber: 10 TURNER
RecordNumber: 11 ADAMS
RecordNumber: 12 JAMES
RecordNumber: 13 FORD
RecordNumber: 14 MILLER
RecordNumber: 15 SUDHEER

```

PL/SQL procedure successfully completed.

C .Write a program to fetch all the data from a table and display it on the screen using %row type attribute.

```

SQL>declare
cursor c is select * from dept;
rec c%ROWTYPE;
BEGIN
  open c;
  loop
    fetch c into rec;
    exit when c%notfound;
    Dbms_output.put_line('deptno:'||rec.deptno);
    Dbms_output.put_line('dname:'||rec.dname);
  End loop;
  close c;
end;

```

Output:

```

Set serveroutput on
deptno:10
dname: ACCOUNTING
deptno:20
dname:RESEARCH
deptno:30
dname:SALES

```

PL/SQL procedure successfully completed.

D .Write a program to display the number of records of any given table %rowcount.

```
SQL>declare
cursor c is select * from dept;
rec c%ROWTYPE;
BEGIN
open c;
loop
fetch c into rec;
exit when c%rowcount=4;
Dbms_output.put_line('deptno='||rec.deptno);
Dbms_output.put_line('dname='||rec.dname);
Dbms_output.put_line('loc='||rec.loc);
end loop;
close c;
end;
```

Output:

setserveroutput on

```
>deptno10
dname=ACCOUNTING
loc=NEW YORK
deptno=20
dname=RESEARCH
loc=DALLAS
deptno=30
dname=SALES
loc=CHICAGO
PL/SQL procedure successfully completed.
```

F .Write a program to check whether the cursor is opened or not. if cursor is opened “Display Cursor Already Opened” else open the cursor and display the message “Opened the cursor”.

```
SQL>declare
cursor c1 is select * from emp;
begin
open c1;

if c1%isopen then
dbms_output.put_line('cursor is already open');
else
open c1;
dbms_output.put_line('opened cursor');
end if;
close c1;
end;
```


Output:

```
setserveroutput
SQL> @pp2
15 /
cursor is already open'
```

G .Write a program to fetch all the data from salgrade table using cursor FOR loop.

```
SQL>declare
cursor c1 is select * from salgrade ;
begin
for rec in c1 loop
dbms_output.put_line('grade='||rec.grade);
dbms_output.put_line('hisal='||rec.hisal);
dbms_output.put_line('losal='||rec.losal);
end loop;
end;
```

Output:

```
setserveroutput
grade=12
hisal=10
```

Aim: Write PL/SQL procedure for an application using exception handling.

TYPES OF PREDEFINED EXCEPTIONS:**NO_DATA_FOUND EXCEPTION:**

```
SQL>
1 declare
2 v_empnoemp.empno%type:=&eno;
3 v_enameemp.ename%type;
4 v_salemp.sal%type;
5 begin
6 selectename,sal into v_ename,v_sal from emp where empno=v_empno;
7 dbms_output.put_line('Name: ' ||v_ename || 'Salary: '||v_sal);
8 exception
9 when NO_DATA_FOUND then
10 dbms_output.put_line('Sorry, Data is not found.');
```

```
11* end;
SQL> /
```

Output:

```
Enter value for eno: 7788
old 2: v_empnoemp.empno%type:=&eno;
new 2: v_empnoemp.empno%type:=7788;
Name: SCOTTSalary: 3000
```

PL/SQL procedure successfully completed.

```
SQL> /
Enter value for eno: 5
old 2: v_empnoemp.empno%type:=&eno;
new 2: v_empnoemp.empno%type:=5;
Sorry, Data is not found.
```

PL/SQL procedure successfully completed.

TOO_MANY_ROWS EXCEPTION:

```
SQL> declare
2 v_empemp%rowtype;
3 v_salemp.sal%type:=&sal;
4 begin
5 select * into v_emp from emp where sal=v_sal;
6 dbms_output.put_line('Name: ' ||v_emp.ename || 'Salary: '||v_emp.sal);
7 exception
8 when TOO_MANY_ROWS then
9 dbms_output.put_line('More Than one employee having same salary');
10 end;
11 /
```

Output:

```
Enter value for sal: 800
old 3: v_salemp.sal%type:=&sal;
new 3: v_salemp.sal%type:=800;
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 5
```

```
SQL> /
Enter value for sal: 5000
old 3: v_salemp.sal%type:=&sal;
new 3: v_salemp.sal%type:=5000;
More Than one employee having same salary
```

PL/SQL procedure successfully completed.

```
SQL> /
Enter value for sal: 1500
old 3: v_salemp.sal%type:=&sal;
new 3: v_salemp.sal%type:=1500;
Name: TURNERSalary: 1500
```

PL/SQL procedure successfully completed.

INVALID_NUMBER EXCEPTION:

SQL>ed Wrote file afiedt.buf

```

1 declare
2 v_empno varchar2(4):=&empno';
3 v_ename varchar2(20):=&ename';
4 v_deptno varchar2(2):=&deptno';
5 begin
6 insert into emp(empno,ename,deptno) values(v_empno,v_ename,v_deptno);
7 exception
8 when INVALID_NUMBER then
9   dbms_output.put_line('Given employee number or department number is invalid');
10* end;
/

```

Output:

```

Enter value for empno: 10
old 2: v_empno varchar2(4):=&empno';
new 2: v_empno varchar2(4):='10';
Enter value for ename: xyz
old 3: v_ename varchar2(20):=&ename';
new 3: v_ename varchar2(20):='xyz';
Enter value for deptno: 10
old 4: v_deptno varchar2(2):=&deptno';
new 4: v_deptno varchar2(2):='10';

```

PL/SQL procedure successfully completed.

SQL> /

```

Enter value for empno: 11
old 2: v_empno varchar2(4):=&empno';
new 2: v_empno varchar2(4):='11';
Enter value for ename: abc
old 3: v_ename varchar2(20):=&ename';
new 3: v_ename varchar2(20):='abc';
Enter value for deptno: a
old 4: v_deptno varchar2(2):=&deptno';
new 4: v_deptno varchar2(2):='a';
Given employee number or department number is invalid

```

PL/SQL procedure successfully completed.

VALUE_ERROR EXCEPTION:

SQL>

```

1 declare
2 v_num1 number;
3 begin
4 v_num1:='&givenumber1'+&givenumber2';
5 dbms_output.put_line('The result of the operation is: ' || v_num1);

```

```

6 exception
7 when VALUE_ERROR then
8 dbms_output.put_line('Please check- there is source of invalid values in input ');
9* end;

```

Output:

/

```

Enter value for givenumber1: 10
old 4: v_num1:='&givenumber1'+'givenumber2';
new 4: v_num1:='10'+'givenumber2';
Please check- there is source of invalid values in input

```

PL/SQL procedure successfully completed.

CASE_NOT_FOUND EXCEPTION:

SQL>

```

1 declare
2   v_opvarchar(2):='&op';
3   v_num1 number:=&op1;
4   v_num2 number:=&op2;
5   begin
6   case
7   when v_op= '+' then
8     dbms_output.put_line('The sum is:' ||to_number(v_num1+v_num2));
9   when v_op= '-' then
10    dbms_output.put_line('The difference is:' ||to_number(v_num1-v_num2));
11  when v_op= '*' then
12    dbms_output.put_line('The Multiplication is:' ||to_number(v_num1*v_num2));
13  when v_op= '/' then
14    dbms_output.put_line('The Quotient is:' ||to_number(v_num1/v_num2));
15  when v_op= '**' then
16    dbms_output.put_line('The power is:' ||to_number(v_num1**v_num2));
17  end case;
18 exception
19 when CASE_NOT_FOUND then
20   dbms_output.put_line('raised the exception case_not_found');
21* end;
/

```

Output:

```

Enter value for op: ++
old 2: v_opvarchar(2):='&op';
new 2: v_opvarchar(2):='++';
Enter value for op1: 10
old 3: v_num1 number:=&op1;
new 3: v_num1 number:=10;
Enter value for op2: 10
old 4: v_num2 number:=&op2;
new 4: v_num2 number:=10;

```

raised the exception case_not_found

PL/SQL procedure successfully completed.

MORE THAN ONE

```
SQL> declare
  2  l_empno emp.empno%type;
  3  l_job emp.job%type;
  4  incment number;
  5 begin
  6  l_empno:=&empno;
  7  select job into l_job from emp where empno=l_empno;
  8  if l_job='CLERK' then
  9      incment:=100;
 10  elsif l_job='SALESMAN' then
 11      incment:=200;
 12  else
 13      incment:=300;
 14  end if;
 15  update emp set sal=sal+incment where empno=l_empno;
 16  Exception
 17      when no_data_found then
 18          dbms_output.put_line('No Employee in Organization');
 19      when too_many_rows then
 20          dbms_output.put_line('Only allowed for one row');
 21 end;
 22 /
```

Output:

```
Enter value for empno: 7788
old 6: l_empno:=&empno;
new 6: l_empno:=7788;
```

PL/SQL procedure successfully completed.

```
SQL> /
Enter value for empno: 2
old 6: l_empno:=&empno;
new 6: l_empno:=2;
No Employee in Organization
```

PL/SQL procedure successfully completed.

ZERO_DIVIDE EXCEPTION:

```
SQL>
  1 declare
  2 v_num1 number:=&num1;
  3 v_num2 number:=&num2;
```

```

4 v_result number;
5 begin
6 v_result:=v_num1/v_num2;
7 dbms_output.put_line('The Result is: ' || v_result);
8 exception
9 when ZERO_DIVIDE then
10 dbms_output.put_line('Fatal Error-- division by zero occurred');
11* end;
/

```

Output:

```

Enter value for num1: 20
old 2: v_num1 number:=&num1;
new 2: v_num1 number:=20;
Enter value for num2: 2
old 3: v_num2 number:=&num2;
new 3: v_num2 number:=2;
The Result is: 10

```

PL/SQL procedure successfully completed.

```

SQL> /
Enter value for num1: 20
old 2: v_num1 number:=&num1;
new 2: v_num1 number:=20;
Enter value for num2: 0
old 3: v_num2 number:=&num2;
new 3: v_num2 number:=0;
Fatal Error-- division by zero occurred

```

PL/SQL procedure successfully completed.

USER DEFINED EXCEPTIONS

```

SQL> declare
2 nullsal exception;
3 mysalemp.sal%type;
4 begin
5 select sal into mysal from emp where empno=&n;
6 if mysal is null or mysal=0 then
7 raise nullsal;
8 else
9 dbms_output.put_line(mysal);
10 end if;
11 exception
12 when nullsal then
13 dbms_output.put_line('salary is null');
14 end;
15 /

```

Output;

Enter value for n: 7788

old 5: select sal into mysal from emp where empno=&n;

new 5: select sal into mysal from emp where empno=7788;

salary is null

PL/SQL procedure successfully completed.

5 . Write a PL/SQL Code using Procedures, Functions, and Packages FORMS

A. Aim: Write a DBMS program to prepare reports for an application using functions.

Write a PL/SQL procedure to update the employee salary

```
SQL> create or replace procedure mybonus as
cursordeptcursor is select deptno from dept;
begin
for r in deptcursor
loop
updatedept set sal=sal*0.95 where deptno=r.deptno;
dbms_output.put_line('the bonus information is; ' ||r.deptno);
end loop;
endmybonus;
/
```

Procedure created.

Output:

```
SQL> save p1
Created file p1
SQL> begin
mybonus;
end;
/
the bonus information is; 10
the bonus information is; 20
the bonus information is; 30
the bonus information is; 40
```

PL/SQL procedure successfully completed.

1. Write a PL/SQL procedure to display the employee details

```
SQL> create or replace procedure getnamesaljob(pempnoemp.empno%type) as
v_enameemp.ename%type;
v_salemp.sal%type;
v_jobemp.job%type;
begin
selectename,sal,job into v_ename,v_sal,v_job from emp where empno=pempno;
dbms_output.put_line('The details of employee: '||pempno);
dbms_output.put_line('The Name of the employee is:'||v_ename);
```

```
dbms_output.put_line('The salary of the employee is'||v_sal);
dbms_output.put_line('The job of the employee is'||v_job);
endgetnamesaljob;
/
```

Procedure created.

Output:

```
SQL> save p2
Created file p2
SQL> exec getnamesaljob(7788);
The details of employee: 7788
The Name of the employee is:SCOTT
The salary of the employee is;3000
The job of the employee is:ANALYST
```

PL/SQL procedure successfully completed.

B. Write a PL/SQL procedure to find the given number is even or odd

```
SQL>ed Wrote file afiedt.buf
```

```
SQL> create or replace procedure oddnumber(num1 number,num2 number) as
mynum number(4);
begin
mynum:=num1;
whilemynum<num2
loop
if mod(mynum,2)!=0 then
dbms_output.put_line('The odd number ' ||mynum);
end if;
mynum:=mynum+1;
end loop;
end;
SQL> /
```

Procedure created.

OUTPUT:

```
SQL> exec oddnumber(10,20);
The odd number 11
The odd number 13
The odd number 15
The odd number 17
The odd number 19
```

PL/SQL procedure successfully completed.

Functions:

SQL>ed Wrote file afiedt.buf

```
create or replace function factorial(num number) return number
is
fact number(4):=1;
begin
formyindex in reverse 1..num
loop
fact:=fact*myindex;
end loop;
return fact;
end;
function created
```

OUTPUT:

SQL>ed Wrote file afiedt.buf

```
declare
v_factorial number(4):=0;
begin
v_factorial:=factorial(5);
    DBMS_OUTPUT.PUT_LINE('The factorial value is : ' || v_factorial);
end;
SQL> /
The factorial value is : 120
```

PL/SQL procedure successfully completed.

SQL>ed Wrote file afiedt.buf

Ex 2:

```
create or replace function combination(num1 number,num2 number) return number
as
combi number(7):=1;
begin
combi:=factorial(num1)+factorial(num2);
returncombi;
end;
SQL> /
```

Function created.

OUTPUT:

SQL> save combination

Created file combination

```
SQL> declare
totals number(5):=0;
begin
totals:=combination(5,5);
  DBMS_OUTPUT.PUT_LINE('The sum of factorial is : ' || totals);
end;
/
The sum of factorial is : 240
```

PL/SQL procedure successfully completed.

Ex 3:

```
SQL> create or replace function empexp(v_empno number) return number as
v_hiredateemp.hiredate%type;
v_exp number(6,2):=1;
begin
selecthiredate into v_hiredate from emp where empno=v_empno;
v_exp:=months_between(sysdate,v_hiredate)/12;
returnv_exp;
end;
/
```

Output:

Function created.

```
SQL>ed
Wrote file afiedt.buf
```

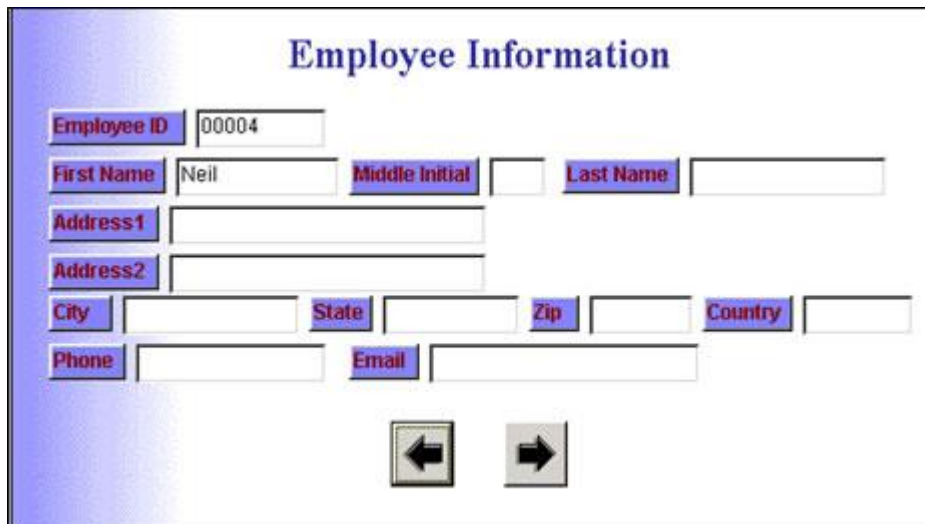
```
declare
exp number;
begin
exp:=empexp(7788);
  DBMS_OUTPUT.PUT_LINE('Given employee experience is: ' || exp || ' Years');
end;
SQL> /
Given employee experience is: 23.42 Years
```

PL/SQL procedure successfully completed.

6. Write a PL/SQL Code Creation of forms for any Information System such as Student Information System, Employee Information System etc.

To be done while developing Mini project.

Example of a Employee Information System, which is been developed by Web Technologies.



The form is titled "Employee Information" in a blue serif font. It contains several input fields with blue labels: "Employee ID" (containing "00004"), "First Name" (containing "Neil"), "Middle Initial" (empty), "Last Name" (empty), "Address1" (empty), "Address2" (empty), "City" (empty), "State" (empty), "Zip" (empty), "Country" (empty), "Phone" (empty), and "Email" (empty). At the bottom, there are two navigation buttons: a left arrow and a right arrow.

7. Demonstration of database connectivity

Connect to SQL:

Step 1: Open data base terminal

Step 2: type "conn user_name/password.

Ex: SQL>conn system/manager.

Step 3:user connected to database.

Step 4: After connected to the database user can create their own tables and doing different operations on created tables.

Step 5: Exit the database use "exit" keyword.

Ex: SQL>exit.

To experiment with JDBC (Java database connectivity) you have to create a database and connect to it. On successful connection you get SQL command prompt `sql>` as follows:

```
C:\>sql -h localhost -u root
```

```
Enter password: *****
```

```
Welcome to the SQL monitor. Commands end with; org.
```

```
Your SQL connection id is 7
```

```
Server version: 5.1.46-community SQL Community Server (GPL)
```

```
Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
```

```
This software comes with ABSOLUTELY NO WARRANTY. This is free software,  
and you are welcome to modify and redistribute it under the GPL v2 license
```

```
Type 'help;' or '\h' for help.Type '\c' to clear the current input statement.
```

```
sql>
```

2. Create Database

To create a database you have to supply `CREATE DATABASE` command followed by the database name and then semicolon.

```
mysql> CREATE DATABASE EXPDB;
Query OK, 1 row affected (0.08 sec)

mysql>
```

3. Use Database

Once you have created the database then you have to select it for use to perform operations on it. Command `USE <DATABASE-NAME>` begins a sql (The SQL Command-line Tool) session and lets you perform database operations. Note that, you need to create database only once but have to use it each time you start a mysql session.

```
sql> USE EXPDB;
Database changed

sql>
```

4. Create a table

The `EXPTABLE`, example table to demonstrate JDBC (Java database connectivity) is created by issuing `CREATE TABLE` command as shown below:

```
sql> CREATE TABLE EXPTABLE (
    ID NUMBER(10) NOT NULL PRIMARY KEY,
    NAME VARCHAR2 (50) );
Table created)

sql>
```

5. Insert Records

Just for illustration, two records into `EXPTABLE` are being inserted, you can insert more if you like. Later we will perform select and edit operations on these records using JDBC (Java database connectivity).

```
sql> INSERT INTO EXPTABLE (NAME) VALUES ("ANUSHKA K");
Query OK, 1 row affected (0.09 sec)

sql> INSERT INTO EXPTABLE (NAME) VALUES ("GARVITA K");
Query OK, 1 row affected (0.00 sec)

sql>
```