LABSHEET- 1

Creation, altering and dropping of tables and inserting rows into a table (use constraints while creating tables) examples using SELECT command.

Creating Tables:-

Syntax: Create table tablename (columnname 1 datatype 1,
Columnname 2 datatype 2,
: :
Columnname n datatype n ...);

Example:

```
SQL> Create table student(  rollno      number,
                           name         varchar2(10),
                           Dob          date,
                           city         varchar2(10),
                           State        varchar2(10) );
```

Table Created

Observing Table Information:-

The easiest way to get description about a table is with the DESC command:

Syntax: DESC table name*;*

Example:

```
SQL>desc  student;
```

Name	Null?	Type
rollno		number
name		varchar2(10)
dob		date
city		varchar2(10)
state		varchar2(10)

Altering Tables:-

To alter a table use the alter table command:

Syntax1:

Alter table tablename **add (**columnname datatype*,* ...**);**

Syntax2:

Alter table tablename **modify (**column datatype**);**

Syntax3:

Alter table tablename **drop column** columnname**;**

Syntax4:

Alter table tablename **rename** column oldcolumn_name to Newcolumn_*name*;

Example 1:

SQL>alter table student **add** (pin number (10));

Table altered

Name	Null?	Type
rollno		number
name		varchar2(10)
dob		date
city		varchar2(10)
state		varchar2(10)
pin		number(10)

Example 2:

SQL>alter table student **modify** (city char (12));

Table altered

SQL> desc student;

Name	Null?	Type
rollno		number
name		varchar2(10)
dob		date
city		char(12)
state		varchar2(10)
pin		number(10)

Example 3

SQL>**alter** table student **drop** column pin;

Table altered

SQL> **desc** student;

Name	Null?	Type
rollno		number
name		varchar2(10)
dob		date
city		char(12)
state		varchar2(10)

Example 4:

SQL>**Alter** table student **rename** column name to sname;

Table altered

SQL> **desc** student;

Name	Null?	Type
rollno		number
sname		varchar2(10)
dob		date
city		char(12)

state varchar2(10)

Dropping Tables:-

To delete a table use the following:

Syntax: **Drop** table tablename;

Example:

Table dropped.

SQL> **desc** student;

Object student does not exist

Inserting rows into a table:-

Syntax:

Insert into tablename **values** (value list);

Example:

SQL>insert into student values (01,'anil','12-feb-90','hyd','a.p');
1 row created.

SQL>insert into student values (43,'bhavya','16-oct-91','delhi','newdelhi');
1 row created.

(or)

SQL> Insert into student values (&rollno,'&sname','&dob','&city','&state');

Enter value for rollno: 12
Enter value for sname: rajendra
Enter value for dob: 23-aug-1984
Enter value for city: Chennai
Enter value for state: T.N.
1 row created.

SQL> /
Enter value for rollno: 30
Enter value for sname: ashok
Enter value for dob: 27-mar-1990
Enter value for city: mumbai
Enter value for state: maharastra.
1 row created.

SQL> /

Displaying data in Table:-

SQL> Select * from student;

ROLLNO	SNAME	D.O.B.	CITY	STATE
01	ANIL	12-FEB-90	HYD	A.P
43	BHAVYA	16-OCT-91	DELHI	NEWDELHI
12	RAJENDRA	23-AUG-84	CHENNAI	T.N
30	ASHOK	27-MAR-90	MUMBAI	MAHARASTRA
42	DIVYA	16-OCT-91	DELHI	NEWDELHI
28	SRINIVAS	29-MAY-88	BANGLOUR	KARNATAKA
02	RAJESH	07-JUL-89	GUNTUR	A.P

CONSTRAINT:

You can place constraints to limit the type of data that can go into a table.

Common types of constraints include the following:

- **UNIQUE Constraint** : Ensures that all values in a column are distinct.
- **PRIMARY KEY Constraint** : Ensures that all values in a column are distinct and a Column can't have NULL value.
- **NOT NULL Constraint** : Ensures that a column cannot have NULL value.
- **FOREIGN KEY Constraint** : Used to ensure referential integrity of the data.
- **CHECK Constraint** : Makes sure that all values in a column satisfy certain criteria.
- **DEFAULT Constraint** : Provides a default value for a column when none is specified.

Each constraint is discussed in the following sections.

UNIQUE Constraint:-

```
SQL> CREATE TABLE Customer (    SID          integer Unique,
                                First_Name   varchar2(30),
                                Last_Name    varchar2(30) );
```

"Column "SID" has a unique constraint, and hence cannot include duplicate values"**.**

SID	FIRST_NAME	LAST_NAME
01	Anand	Babu
02	Raghu	Babu
03	Raja	Ram
04	Ravi	kumar
05	Mohan	Rao
06	Usha	Rani

Executing the following SQL statement:

SQL> INSERT INTO Customer values ('02','Raghu','Babu');

It will result in an error because '02' already exists in the SID column, thus trying to insert another row with that value violates the UNIQUE constraint.

Primary Key:-

```
SQL> CREATE TABLE Customer ( SID    integer ,Last_Name  varchar2(30),
First_Name     varchar2(30)  PRIMARY KEY(SID));
```

Below are examples for specifying a primary key by altering a table :

```
SQL>ALTER TABLE Customer ADD PRIMARY KEY (SID);
```

NOT NULL Constraint:-

By default, a column can hold NULL. If you not want to allow NULL value in a column, you will want to place a constraint on this column specifying that NULL is now not an allowable value.

```
SQL>CREATE TABLE Customer (      SID              integer        NOT NULL,
                                 Last_Name        varchar (30)   NOT NULL,
                                 First_Name       varchar (30));
```

Columns "SID" and "Last_Name" cannot include NULL, while "First_Name" can include NULL.

Foreign Key :-

A foreign key is a field (or fields) that **points to the primary key** of another table.
The purpose of the foreign key is to ensure referential integrity of the data.
Below we show examples of how to specify the foreign key when creating the ORDERS table:

```
SQL>CREATE TABLE ORDERS (     Order_ID          integer,
                              Order_Date   date,
                              SID               integer,
                              Amount            number (10),
                              PRIMARY KEY(Order_ID),
                    FOREIGN KEY (SID) references CUSTOMER (SID));
```

In the above example, the SID column in the ORDERS table is a foreign key pointing to the SID column in the CUSTOMER table.

Below are examples for specifying a foreign key by altering a table.

```
SQL> ALTER TABLE ORDERS ADD FOREIGN KEY (SID) REFERENCES
       CUSTOMER;
```

CHECK Constraint :-

The CHECK constraint ensures that all values in a column satisfy certain conditions. Once defined, the database will only insert a new row or update an existing row if the new value satisfies the CHECK constraint.
For example, in the following CREATE TABLE statement,

```
SQL> CREATE TABLE Customer (SID          integer,
                            last_Name    varchar2 (30),
                            First_Name   varchar2 (30)
                            CHECK (SID > 0));
```

Column "SID" has a constraint -- its value must only include integers greater than 0. So, attempting to execute the following statement,

```
SQL> INSERT INTO Customer values ('-3','venkat','Reddy');
```

It will result in an error because the values for SID must be greater than 0.

DEFAULT Constraint :-

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

For example, if we create a table as below:

```
SQL> CREATE TABLE Student (   Student_ID          integer,
                              Last_Name           varchar2 (30),
                              Score               number (10) DEFAULT 50);
```

and execute the following SQL statement,

```
SQL> INSERT INTO Student (Student_ID, Last_Name, First_Name) values
('10','Chinna','Rao');
```

Student_ID	Last_Name	First_Name	Score
10	Chinna	Rao	50

Even though we didn't specify a value for the "Score" column in the INSERT INTO statement, it does get assigned the default value of 50 since we had already set 50 as the default value for this column.

LABSHEET-2

Write the Queries (along with sub Queries) using ANY, ALL, IN, EXISTS, NOTEXISTS, UNION, INTERSET.

```
SQL> create table sailors (  sid        number (10),
                             name       varchar2(20),
                             rating     number(3),
                             age        integer,
                             primary key(sid),
                             check(rating>0 and rating<11) );
```

Table created.

```
SQL> desc sailors;
```

Name	Null?	Type
SID	NOT NULL	NUMBER (10)
SNAME		VARCHAR2 (10)
RATING		NUMBER (3)
AGE		NUMBER(38)

```
SQL> Create Table boats (  bid        number (5)
                           bname      varchar2 (20),
                           colour     varchar2(12),
                           primary key (bid));
```

Table created.

```
SQL> desc boats;
```

Name	Null?	Type
BID	NOT NULL	NUMBER (5)
BNAME		VARCHAR2 (20)

```
SQL> Create Table Reserves (  sid        number (10),
                              bid        number (10),
                              day        date,
                              foreign key (sid) references sailors,
                              foreign key (bid) references boats );
```

Table created.

```
SQL> desc reserves;
```

Name	Null?	Type
SID		NUMBER(10)
BID		NUMBER(10)
DAY		DATE

SQL> insert into sailors values (&sid, '&name', &rating, &age);
Enter value for sid: 22
Enter value for name: anjali
Enter value for rating: 08
Enter value for age: 25
SQL> /Enter value for sid: 29

SQL> /

e.t.c.

SQL> select * from sailors;

SID	SNAME	RATING	AGE
22	Anjali	8	25
29	narendra	5	24
31	Ramya	8	32
32	Raju	8	17
58	Srinu	10	27
64	Anil	7	35
71	Suresh	10	26
74	Ravi	9	18
85	mahesh	3	20
95	srikanrh	2	45

SQL> insert into boats values(&bid,'&bname','&colour');

Enter value for bid: 101
Enter value for bname: interlake
Enter value for colour: blue
old 1: insert into boats values(&bid,'&bname','&colour')
new 1: insert into boats values(101,'interlake','blue')
1 row created

SQL> select * from boats;

BID	BNAME	COLOUR
101	interlake	blue
102	interlake	Red
103	clipper	green
104	marine	Red

4 rows selected.

```
SQL> insert into reserves values (&sid, &bid,'&day');
        Enter value for Sid: 22
        Enter value for bid: 101
        Enter value for day: 10-oct-08
old 1: insert into reserves values(&sid,&bid,'&day')
new 1: insert into reserves values(22,101,'10-oct-08')
1 row created

SQL> /Enter value for day: 11-dec-08
old   1: insert into reserves values(&sid,&bid,'&day')
new   1: insert into reserves values(64,101,'11-dec-08')
1 row created.

SQL> /
```

e.t.c.

```
SQL> select * from reserves;
```

ALL:-

Q) Find the Sailors With the highest Rating.

```
SQL> Select S.Sid, S.Name, S.Rating  From Sailors S
        Where S.Rating >=ALL (select  S2.Rating From Sailors S2);
```

SID	NAME	RATING
58	Srinu	10
71	suresh	10

ANY: -

Q) Find the sailors whose rating is better than some sailors called “Anjali”.

```
SQL> Select S.Sid, S.Name From Sailors S
      Where S.Rating >ANY (Select  S2.Rating From Sailors S2 Where S2.Name='anjali');

    SID        SNAME
     58        srinu
     71        suresh
     74        ravi
```

IN:-

```
SQL> Select S.Sid, S.Name From Sailors S Where S.sid IN (Select R.Sid From Reserves R Where R.Bid IN
                                                    (Select B.Bid From Boats B  where
B.Color='Red'));

    SID        SNAME
     22        anjali
     31        ramya
     64        anil
```

EXISTS:-

Q) Find the names of sailors who have reserved a boat 103.

```
SQL> Select S.Sid, S.Name From Sailors S
       Where EXISTS (Select * From Reserves R Where R.Bid =103 and R.Sid=S.Sid);

    SID        SNAME
     22        anjali
     31        ramya
     74        ravi
```

NOT EXISTS:-

Q) Find the names of sailors who have reserved all boats.

```
SQL> Select S.Sid, S.Name
      From Sailors S
      Where  NOT EXISTS (Select B.Bid  From Boats B MINUS Select R.Bid From Reserves R
                                                          Where R.Sid=S.Sid);

      SID        SNAME
      ---------- ----------
      22         anjali
```

UNION :-

Q) Find the Sid, Bid and Boat Color of sailors who reserved a Green or Blue boats and both.

```
SQL> (Select R.sid ,B.Bid, B.Color from Reserves R,Boats B
                  where B.Colour='Green' and B.Bid=R.Bid)
    UNION
      (Select R.sid, B.Bid, B.Color  from Reserves R,Boats B
```

where B.Colour='Blue' and B.Bid=R.Bid);

SID	BID	COLOUR
22	101	blue
22	103	green
31	103	green
64	101	blue
74	103	green

Q) Find the Sid's of sailors who have reserved at least one Boat.

SQL > (Select S.Sid from Sailors S) INTERSECT (Select R.sid from Reserves R);

SID
22
31
64
74

LABSHEET-3

Aggregate functions:

COUNT ( ):

Q) Find the no. of Sailors.

SQL> Select **COUNT (*)** from Sailors;

COUNT (*)
10

SUM ( ):

Q) Find the SUM of Rating of all Sailors.

SQL> Select **SUM (S.Rating)** from Sailors S;

SUM (RATING)
70

AVG ( ):

Q) Find the AVERAGE AGE of all Sailors.

SQL> Select **AVG (S.Age)** from Sailors S;

AVG (S.AGE)
26.9

Q) Find the AVERAGE AGE of all Sailors with a Rating of 8.

SQL> Select **AVG (S.Age)** from Sailors S where S.Rating=8;

AVG (S.AGE)
23.6666667

MAX ( ):

Q) Find the AGE of the oldest sailor.

SQL> Select **MAX (S.Age)** from Sailors S;

MAX (S.AGE)

45

MIN ( ):

Q) Find the AGE of the youngest sailor.

SQL> Select **MIN (S.Age**) from Sailors S;

17

GROUP BY:

Q) Find the AGE of the youngest sailor at each Rating Level.

SQL> Select S.Rating, MIN (S.Age) From Sailors S **GROUP BY** S.Rating;

RATING	**MIN(S.AGE)**
2	**45**
5	**24**
8	**17**
7	**35**
3	**20**
10	**26**
9	**18**

7 rows selected.

HAVING:-

Q) Find the Average AGE of sailors who are of voting age (18 Years) for each rating level that has at least two sailors.

SQL> Select S.Rating, AVG (S.Age) From Sailors S

Where S.Age >= 18 **GROUP BY** S.Rating

HAVING 1< (Select COUNT (*) From Sailors S2 Where S2.Age>=18 and S2.Rating=S.Rating);

RATING	**AVG (S.AGE)**
8	28.5
10	26.5

Creation and dropping of Views:

Q) Create a view on the sailors who are having voting age (18) with name, Sid, & age as attributes.

SQL> **create view** voting_sailors (sailor_name, sailor_id, sailor_age) as

(select s.sname,s.sid,s.age from sailors s where s.age>=18);

View created.

SQL> select * from voting_sailors;

anjali	22		25	
narendra		29		24
ramya	31		32	
srinu	58		27	
anil	64		35	
suresh	71		26	
ravi	74		18	
mahesh		85		20
srikanrh		95		45

9 rows selected.

Q) Dropping a View.

SQL> DROP VIEW Voting_Sailors;

View dropped.

----------********----------

LABSHEET-4

Queries using Conversion functions (*to_char, to_number and to_date*),

String functions (Concatenation, lpad, rpad, ltrim, rtrim,
lower, upper, initcap, length, substr and instr).

Date functions (Sysdate, next_day, add_months, last_day,
months_between, trunc, round, to_char, to_date).

Conversion functions (to_char, to_date and **to_number):**

Q) What is the time now?

SQL>**select to_char(sysdate,'HH24:MI:SS') as** Time from dual**;**

TIME
10:50:54

Q) What is the day today?
SQL>select To_char(sysdate,' DD ') as Dates from dual;

Dates
14

Q) name of this month

SQL>select To_char(sysdate,'MON') aS Month from dual;

MONTH
FEB

Q) Name of this year

SQL>select To_char(sysdate,'YYYY') as Year from dual**;**

Year
2010

Q) Day number

SQL> select To_char(sysdate,'D') as day_no from dual;

DAY_NO

3

Q) Date and week number.

SQL>select 'Today is ' ||sysdate|| ' and it is day ' ||To_char(sysdate,'D')|| ' in this week.' as WEEKDAY from dual;

WEEKDAY

Today is 23-FEB-10 and it is day 3 in this week.

Q)Add 13 days to the date 17-aug-1985.

SQL>select **to_date**('17-aud-85','dd-mm-yy')+13 as Dates from dual;

DATE

30-AUG-85

Q) Add three months to the date ’03-jul-2010.

SQL> select ADD_months('03-jul-10',3) as Month from dual;

ADD_MONTH

03-oct-10

to_number : The to_Char can convert a string into a number.

Q) Add the character type of data ‘17’ and ‘16’ .

SQL> select to_number('17'+'16')as result from dual;

RESULT

33

String functions (*Concatenation, lpad, rpad,ltrim, rtrim,lower, upper, initcap, length, substr ,instr, and Replace, Ascii*) **:**

Concatenation :

Q) Perfor a concatenation operation on sname of the sailors table with the string’====hai===>’.

SQL> select concat((sname),' ====hai===>')as sname,rating from sailors;

SNAME		RATING
Anjali	====hai===>	8
Narendra	====hai===>	5
Ramya	====hai===>	8
Raju	====hai===>	8
Srinu	====hai===>	10
Anil	====hai===>	7
Suresh	====hai===>	10
Ravi	====hai===>	9
Mahesh	====hai===>	3
Srikanrh	====hai===>	2

Lpad :

SID	LPAD(SNAME,10,'*')
22	****anjali
29	**narendra
31	*****ramya
32	******raju
58	*****srinu
64	******anil
71	****suresh
74	******ravi
85	****mahesh
95	**srikanrh

Rpad:

Q) Perfor rpad operation on sname of the sailors table with a string '*'.

SQL> select sid ,rpad(sname,10,'*')from sailors ;

SID	RPAD(SNAME,10,'*')
22	anjali****
29	narendra**
31	ramya*****
32	raju******
58	srinu*****
64	anil******
71	suresh****
74	ravi******
85	mahesh****

PRINCIPAL Ltrim :

Q) Perfor ltrim operation on the string" professorDr. G.J.Rao".

SQL> select ltrim('professorDr. G.J.Rao','professor')as principal from dual ;

Dr. G. J. Rao

Q) Perfor ltrim operation on the string" professorDr. G.J.Rao".

SQL> select rtrim('professorDr. G. J. Rao','Dr. G. J. Rao')as principal from dual ;

PRINCIPAL

professor

Lower :

Q)Display the names of the sailors in lowercase whose rating is less than 5.

SQL> select sid, lower(sname),rating from sailors where rating <5;

SID	LOWER(SNAM	RATING
85	mahesh	3
95	srikanrh	2

Upper :

Q) Display the names of the sailors in uppercase whose rating is equal to 8.

SQL> select sid, upper(sname),rating from sailors where rating =8;

SID	UPPER(SNAME)	RATING
22	ANJALI	8
31	RAMYA	8
32	RAJU	8

Initcap :

Q) Display the names of the sailors starting with capital letters whose rating is equal to 8.

SQL> select sid, initcap(sname),rating from sailors where rating =8;

SID	INITCAP(SNAME)	RATING
22	Anjali	8
31	Ramya	8
32	Raju	8

length :

SID	SNAME	LENGTH(SNAME)
58	srinu	5
71	suresh	6
74	ravi	4

Q) Display the names of the sailors starting with third character in a name string whose rating is greater than 8.

SQL> select sid,sname,substr(sname,3)from sailors where rating>8;

SID	SNAME	SUBSTR(SNAME,3)
58	srinu	inu
71	suresh	resh
74	ravi	vi

Q) Find the firs occurrence of a character ‘J ‘ in the name(string)’s of sailors whose rating is equal to 8.

SQL> select sid,sname,instr(sname,'j')from sailors where rating=8;

SID	SNAME	INSTR(SNAME,'J')
22	anjali	3
31	ramya	0
32	raju	3

Replace:

Q) Replace a character ‘g’ in a place of ‘j’ in the name(string)’s sailors whose rating is equal to 8.

SQL> select sid,sname,replace(sname,'j','g')from sailors where rating=8;

SID	SNAME	REPLACE(SNAME)
22	anjali	angali
31	ramya	ramya
32	raju	ragu

ASCII :

Q) Find the ASCII value for a character 'L'

SQL> select ascii('L') from dual;

```
ASCII('L')
76
```

Q) Find the ASCII value for a character 'l'.

SQL> select ascii(l) from dual;

```
108
```

Date functions: (Sysdate, next_day, add_months, last_day,months_between, trunc and round,Power)

Sysdate:

Q) Display today's date.

SQL> select sysdate from dual;

```
SYSDATE
-----------
24-FEB-10
```

next_day :

Q) Find the date of next Sunday.

SQL> select next_day(sysdate,'sunday')as next_sunday from dual;

```
NEXT_SUND
28-FEB-10
```

add_months :

Q) Add one month extra for the sailors who reserved a boat no104.

SQL> select sid,bid,day,add_months(day,1) from reserves where bid =104;

SID	BID	DAY	ADD_MONTH
22	104	11-NOV-08	11-DEC-08
31	104	11-JUL-08	11-AUG-08

last_day :

Q) Find the last day of this month.

SQL> select sysdate,last_day(sysdate) from dual;

SYSDATE	LAST_DAY(sysdate)
24-FEB-10	28-FEB-10

months_between :

Q) Find number of months between reserved date to current date of a boat with bid=103.

SQL> select sid,sysdate,trunc(months_between(sysdate,day))as no_of_months from reserves where bid=103;

SID	SYSDATE	NO_OF_MONTHS
22	24-FEB-10	16
31	24-FEB-10	20
74	24-FEB-10	18
71	24-FEB-10	8

SQL> SELECT 927.16, ROUND(927.76), TRUNC(927.99), POWER(3,2) FROM DUAL;

927.16	ROUND(927.76)	TRUNC(927.99)	POWER(3,2)
927.16	928	927	9

----------**********----------

What is PL/SQL?

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.

A Simple PL/SQL Block:

Each PL/SQL program consists of SQL and PL/SQL statements which from a PL/SQL block.

A PL/SQL Block consists of three sections:

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

Declaration Section:

The Declaration section of a PL/SQL Block starts with the reserved keyword DECLARE. This section is optional and is used to declare any placeholders like variables, constants, records and cursors, which are used to manipulate data in the execution section. Placeholders may be any of Variables, Constants and Records, which stores data temporarily. Cursors are also declared in this 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 statement and SQL statements form the part of execution section.

Exception Section:

The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION. This section is optional. Any errors in the program can be handled in this section, so that the PL/SQL Blocks terminates gracefully. If the PL/SQL Block contains exceptions that cannot be handled, the Block terminates abruptly with errors.

Every statement in the above three sections must end with a semicolon **;** . PL/SQL blocks can be nested within other PL/SQL blocks. Comments can be used to document code.

This is how a sample PL/SQL Block looks.

```
DECLARE
        Variable declaration
BEGIN
        Program Execution
EXCEPTION
        Exception handling
END;
```

Advantages of PL/SQL

These are the advantages of PL/SQL.

- ***Block Structures*:** PL SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.

- ***Procedural Language Capability*:** PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).

- ***Better Performance*:** PL SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.

- ***Error Handling*:** PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

Write a sample program for PL/SQL structure

```
declare
begin
dbms_output.put_line('welcome to pace MBA');
end;
/
```

Output:

```
SQL>set serveroutput on;
SQL>edit 1.sql;
SQL> @ 1
```

PL/SQL procedure successfully completed.

Write a sample PL/SQL program using variable declaration

```
declare
a varchar2(50):=’welcome to IT LAB-I’;
begin
dbms_output.put_line(a);
end;
/
```

Output:

```
SQL> @ 1
welcome to IT LAB-I
```

PL/SQL procedure successfully completed.

Aim: Write a PL/SQL program using arthimetic operations

```
declare
a number(2);
b number(2);
begin
a:=10;
b:=20;
dbms_output.put_line('addition'||(a+b));
dbms_output.put_line('subtraction'||(a-b));
dbms_output.put_line('multiplication'||(a*b));
dbms_output.put_line('division'||(a/b));
end;
/
SQL> @ 1
addition30
subtraction-10
multiplication200
division.5
```

PL/SQL procedure successfully completed.

Aim: Write a PL/SQL program for finding average of 3 numbers

```
declare
a number(2);
b number(2);
c number(2);
d number(2);
begin
a:=&a;
b:=&b;
dbms_output.put_line('average of 3 numbers'||d);
end;
/
```

Output:

```
SQL> @ 1
Enter value for a: 5
old  7: a:=&a;
new  7: a:=5;
Enter value for b: 6
old  8: b:=&b;
new  8: b:=6;
Enter value for c: 7
old  9: c:=&c;
new  9: c:=7;
average of 3 numbers6

PL/SQL procedure successfully completed.
```

Write a PL/SQL program to finding the salary of given employee.

```
declare
i emp%rowtype;
newsal emp.sal%type;
begin
i.empno:=&empno;
select ename,sal into i.ename,i.sal from emp where empno=i.empno;
newsal:=i.sal+i.sal;
dbms_output.put_line('empno'||'  '||'ename'||'  '||'sal'||'  '||'newsal');
dbms_output.put_line(i.empno||'  '||i.ename||'  '||i.sal||'  '||newsal);
end;
/
```

Output:

```
SQL> @ 1
Enter value for empno: 7698
old  5: i.empno:=&empno;
```

```
new  5: i.empno:=7698;
empno ename sal newsal
7698 BLAKE 2850 5700
```

LABSHEET-V

Creation of simple PL/SQL program which includes declaration section, executable section and exception – Handling section
(Ex. Student marks can be selected from the table and print along with his/her Grade, and an exception can be raised if no records were found)

Exception Handling

In this section we will discuss about the following,

1) What is Exception Handling.

3) Types of Exception Handling.

1) What is Exception Handling?

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly. When an exception occurs a messages which explains its cause is recieved.

PL/SQL Exception message consists of three parts.

1) Type of Exception

2) An Error Code

3) A message

By Handling the exceptions we can ensure a PL/SQL block does not exit abruptly.

2) Structure of Exception Handling.

The General Syntax for coding the exception section

```
DECLARE
  Declaration section
BEGIN
  Exception section
EXCEPTION
WHEN ex_name1 THEN
   -Error handling statements
WHEN ex_name2 THEN
   -Error handling statements
WHEN Others THEN
  -Error handling statements
END;
```

General PL/SQL statments can be used in the Exception Block.

When an exception is raised, Oracle searches for an appropriate exception handler in the exception section.

For example in the above example, if the error raised is 'ex_name1 ', then the error is handled according to

the statements under it. Since, it is not possible to determine all the possible runtime errors during testing fo the code, the 'WHEN Others' exception is used to manage the exceptions that are not explicitly handled. Only one exception can be raised in a Block and the control does not return to the Execution Section after the error is handled.

If there are nested PL/SQL blocks like this.

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

In the above case, if the exception is raised in the inner block it should be handled in the exception block of the inner PL/SQL block else the control moves to the Exception block of the next upper PL/SQL Block. If none of the blocks handle the exception the program ends abruptly with an error.

3) Types of Exception.

There are 3 types of Exceptions.

a) Named System Exceptions

b) Unnamed System Exceptions

c) 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.

Named system exceptions are:

1) Not Declared explicitly,

2) Raised implicitly when a predefined Oracle error occurs,

3) caught by referencing the standard name within an exception-handling routine.

Exception Name	Reason	Error Number
CURSOR_ALREADY_OPEN	When you open a cursor that is already open.	ORA-06511
INVALID_CURSOR	When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened.	ORA-01001
NO_DATA_FOUND	When a SELECT...INTO clause does not return any row from a table.	ORA-01403
TOO_MANY_ROWS	When you SELECT or fetch more than one row into a record or variable.	ORA-01422
ZERO_DIVIDE	When you attempt to divide a number by zero.	ORA-01476

For Example: Suppose a NO_DATA_FOUND exception is raised in a proc, we can write a code to handle the exception as given below.

```
BEGIN
  Execution section
EXCEPTION
WHEN NO_DATA_FOUND THEN
 dbms_output.put_line ('A SELECT...INTO did not return any row.');
 END;
```

b) Unnamed System Exceptions

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

There are two ways to handle unnamed sysyem 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.

We can assign a name to unnamed system exceptions using a **Pragma** called **EXCEPTION_INIT. EXCEPTION_INIT** will associate a predefined Oracle error number to a programmer_defined exception name.

Steps to be followed to use unnamed system exceptions are

- They are raised implicitly.
- If they are not handled in WHEN Others they must be handled explicity.
- To handle the exception explicity, they must be declared using Pragma EXCEPTION_INIT as given above and handled referecing the user-defined exception name in the exception section.

The general syntax to declare unnamed system exception using EXCEPTION_INIT is:

```
DECLARE
  exception_name EXCEPTION;
  PRAGMA
  EXCEPTION_INIT (exception_name, Err_code);
BEGIN
 WHEN exception_name THEN
   handle the exception
END;
```

c) User-defined Exceptions

Apart from sytem exceptions we can explicity 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.

For Example: Lets consider the product table and order_items table from sql joins to explain user-defined exception.

Lets create a business rule that if the total no of units of any particular product sold is more than 20, then it is a huge quantity and a special discount should be provided.

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 (i.e. change due to INSERT, UPDATE, or DELETE statements).

RAISE_APPLICATION_ERROR raises an exception but does not handle it.

RAISE_APPLICATION_ERROR is used for the following reasons,

a) to create a unique id for an user-defined exception.

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

The General Syntax to use this procedure is:

RAISE_APPLICATION_ERROR (error_number, error_message);

• The Error number must be between -20000 and -20999

• The Error_message is the message you want to display when the error occurs.

Steps to be folowed to use RAISE_APPLICATION_ERROR procedure:

1. Declare a user-defined exception in the declaration section.

2. Raise the user-defined exception based on a specific business rule in the execution section.

Using the above example we can display a error message using RAISE_APPLICATION_ERROR.

Create of simple PL/SQL program which includes declaration section, execution section and exception handling section

```
declare
v_ename emp.ename%type;
v_sal emp.sal%type;
begin
select ename,sal into v_ename,v_sal from emp where empno=&empno;
dbms_output.put_line(v_ename||'salary is'||v_sal);
exception
when no_data_found then
dbms_output.put_line('empno not found');
end;
/
```

Output:

```
SQL> @ 1
Enter value for empno: 1
old  5: select ename,sal into v_ename,v_sal from emp where empno=&empno;
new  5: select ename,sal into v_ename,v_sal from emp where empno=1;
empno not found

PL/SQL procedure successfully completed.

SQL> /
Enter value for empno: 7902
old  5: select ename,sal into v_ename,v_sal from emp where empno=&empno;
new  5: select ename,sal into v_ename,v_sal from emp where empno=7902;
FORDsalary is3000

PL/SQL procedure successfully completed.
```

Write a PL/SQL program for exception handling

```
DECLARE
v_deptno    emp.deptno%type := &sv_deptno;
v_total_emp NUMBER;
BEGIN
IF v_deptno < 0 THEN
RAISE_APPLICATION_ERROR (-20000, 'deptno cannot be negative');
ELSE
SELECT COUNT(*) INTO v_total_emp FROM emp WHERE deptno = v_deptno;
END;
/
```

Output:

```
SQL> @ b.sql
Enter value for sv_deptno: 20
old  2: v_deptno    emp.deptno%type := &sv_deptno;
new  2: v_deptno    emp.deptno%type := 20;
Depatment has 5 employees
PL/SQL procedure successfully completed.
SQL> /
Enter value for sv_deptno: -20
old  2: v_deptno    emp.deptno%type := &sv_deptno;
new  2: v_deptno    emp.deptno%type := -20;
DECLARE
*
ERROR at line 1:
ORA-20000: deptno cannot be negative
ORA-06512: at line 6
```

```
SQL> SET SERVEROUTPUT ON;    /* To Enable the buffer to display the output */

SQL> ed stud.sql;                          /* To create pl/sql file */
DECLARE
        S_Num Number(10);
        S_Name Varchar(20);
        S_Marks Number(10);
        S_Grade Char(10);

BEGIN
        s_num:=&s_num;
        SELECT S.sName,S.Marks,S.Grade

        INTO S_Name,S_Marks,S_Grade
```

```
        FROM Student S
        WHERE S.Sno=S_Num;

        DBMS_OUTPUT.PUT_LINE('Student Name: ' ||S_Name );
        DBMS_OUTPUT.PUT_LINE('Total marks: ' ||s_Marks);
        DBMS_OUTPUT.PUT_LINE('Grade: ' ||S_Grade );

EXCEPTION
        WHEN NO_DATA_FOUND THEN



SQL> @ stud.sql;

Enter value for s_num: 1201
old  9: s_num:=&s_num;
new  9: s_num:=1201;
Student Name: purna
Total marks: 75
Grade: a

PL/SQL procedure successfully completed.

SQL> /
Enter value for s_num: 507
old  9: s_num:=&s_num;
new  9: s_num:=507;
There is no student with Student number : 507

PL/SQL procedure successfully completed.
```

----------**********----------

LAB SHEET-VI

Develop a program that includes the features NESTED IF, CASE and CASE expression. The program can be extended using the NULLIF and COALESCE functions.

Write a PL/SQL program to find given number is even or odd

```
declare
i number(2);
begin
i:=&i;
dbms_output.put_line(i||'is odd');
end if;
end;
/
```

Output:

```
SQL> @ 1
Enter value for i: 3
old  4: i:=&i;
new  4: i:=3;
3is odd

PL/SQL procedure successfully completed.

SQL> @ 1
Enter value for i: 4
old  4: i:=&i;
new  4: i:=4;
4is even

PL/SQL procedure successfully completed.
```

Write a PL/SQL program to find commission for given employee

```
declare
i emp%rowtype;
newcomm emp.comm%type;
begin
i.empno:=&empno;
select comm into i.comm  from emp where empno=i.empno;
if i.comm is null then
newcomm:=1000;
```

elsif i.comm<300 then

```
newcomm:=i.comm+3000;
elsif i.comm=0 then
newcomm:=i.comm+2000;
else
newcomm:=i.comm+4000;
end if;

dbms_output.put_line(i.comm||' '||newcomm);
end;
/
SQL> @ 1.sql;
Enter value for empno: 7698
old  5: i.empno:=&empno;
new  5: i.empno:=7698;
1000
PL/SQL procedure successfully completed.
SQL> /
Enter value for empno: 7844
old  5: i.empno:=&empno;
new  5: i.empno:=7844;
0      3000
PL/SQL procedure successfully completed.
```

LAB SHEET-VII

Program development using WHILE LOOPS, numeric FOR LOOPS, nested loops using ERROR Handling, BUILT –IN Exceptions, USE defined Exceptions, RAISE- APPLICATION ERROR.

Write a PL/SQL program to find given number type

```
Declare
I number(3);
begin
for i in 1..10 loop
dbms_output.put_line(' This is divisible by 3: '||i);
elsif i=2 or i=4 or i=6 or i=8 or i=10 then
dbms_output.put_line(' This is divisible by 2: '||i);
else
dbms_output.put_line(' Is this a prime number?? '||i);
end if;
end loop;
end;
/
```

Output:

```
SQL> @ b.sql
Is this a prime number?? 1
This is divisible by 2: 2
This is divisible by 3: 3
This is divisible by 2: 4
Is this a prime number?? 5
This is divisible by 3: 6
Is this a prime number?? 7
This is divisible by 2: 8
This is divisible by 3: 9
This is divisible by 2: 10

PL/SQL procedure successfully completed.
```

Write a PL/SQL program to find the sum of given numbers

```
declare
N NUMBER(3);
S NUMBER(4):=0;
I NUMBER(3):=1;
begin
N:=&N;
WHILE I <= N LOOP
  S:=S+I;
```

```
  I:=I+1;
END LOOP;
DB
MS_OUTPUT.PUT_LINE('SUM OF GIVEN NUMBERS'||S);
END;
/
```

Output:

```
SQL> @ 1.sql;
Enter value for n: 9
old  6: N:=&N;
SUM OF GIVEN NUMBERS45

PL/SQL procedure successfully completed.



SQL> @ 1
Enter value for n: 0
old  6: N:=&N;
new  6: N:=0;
SUM OF GIVEN NUMBERS0

PL/SQL procedure successfully completed.
```

Write a PL/SQL program to print the numbers in forwarding and reversing order

```
declare
begin
dbms_output.put_line(' the numbers are');
for n in 1..5 loop
dbms_output.put_line(n);
end loop;
dbms_output.put_line('end of numbers');
dbms_output.put_line(' reverse numbers are');
for n in reverse 1..5 loop
dbms_output.put_line(n);
end loop;
dbms_output.put_line('end of numbers');
end;
/
SQL> @ 1
the numbers are
1
2
3
4
end of numbers
```

the reverse numbers are
4
3
2
1
end of numbers

PL/SQL procedure successfully completed.

```
declare
num number(5):=&num;
f1 number(5):=0;
f2 number(5):=1;
f3 number(5);
i number(5):=3;
begin
dbms_output.put_line('The fibonacci series is');
dbms_output.put_line(f1);
dbms_output.put_line(f2);
while(i<=num) loop
f3:=f1+f2;
dbms_output.put_line(f3);
f1:=f2;
f2:=f3;
i:=i+1;
end loop;
end;
/
```

Output:

```
SQL> /
Enter value for num:5
old  2: num number(5):=&num;
new  2: num number(5):=9;
The fibonacci series is
0
1
1
2
3
PL/SQL procedure successfully completed.
```

LAB SHEET-VIII

Programs development using creation of procedures, passing parameters IN and OUT of PROCEDURES.

Stored Procedures

What is a Stored Procedure?

1) IN-parameters

2) OUT-parameters

3) IN OUT-parameters

A procedure may or may not return any value.

General Syntax to create a procedure is:

CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters]

IS

Declaration section

BEGIN

Execution section

EXCEPTION

Exception section

END;

IS - marks the beginning of the body of the procedure and is similar to DECLARE in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section.

Write a Procedure to find given year is leap year or not

```
CREATE OR REPLACE procedure leap(x number) is
a number;
BEGIN
a:=x;
if mod(a,4)=0
then
dbms_output.put_line(a||'is leap year');
```

```
else
dbms_output.put_line(a||'is not leap year');
end if;
END;
/
```

Output:

```
SQL> @ 1

PL/SQL procedure successfully completed.

SQL> set serveroutput on;
SQL> exec leap(2000);
2000is leap year


PL/SQL procedure successfully completed.

SQL> exec leap(1997);
1997is not leap year

PL/SQL procedure successfully completed.
```

Programs development using creation of procedures, passing parameters IN and OUT of PROCEDURES.

```
CREATE OR REPLACE PROCEDURE find_ename(v_empno IN emp.empno%TYPE, v_ename OUT
VARCHAR2) AS
BEGIN
SELECT ename INTO v_ename FROM emp WHERE empno = v_empno;
EXCEPTION
WHEN OTHERS
THEN
DBMS_OUTPUT.PUT_LINE('Error in finding employee:'||v_empno);
END find_ename;
/
```

Output:

```
SQL> @ a.sql;
Procedure created.
SQL> variable x varchar2(25);
SQL> exec find_ename(7934,:x);
PL/SQL procedure successfully completed.
SQL> print x;
```

X

MILLER

SQL> /

Procedure created.

SQL> exec find_ename(7111,:x)

Error in finding employee:7111

LAB SHEET-IX

Program development using creation of stored functions, invoke functions in SQL Statements and write complex functions.

PL/SQL Functions

What is a Function in PL/SQL?

The General Syntax to create a function is:

CREATE [OR REPLACE] FUNCTION function_name [parameters]

RETURN return_datatype;

IS

Declaration_section

BEGIN

Execution_section

Return return_variable;

EXCEPTION

exception section

Return return_variable;

END;

1) **Return Type:** The header section defines the return type of the function. The return datatype can be any of the oracle datatype like varchar, number etc.

2) The execution and exception section both should return a value which is of the datatype defined in the header section.

Program development using creation of stored functions, invoke functions in SQL Statements and write complex functions.

```
CREATE OR REPLACE FUNCTION show_desc(v_deptno emp.deptno%TYPE) RETURN varchar2
AS
v_description varchar2(50);
BEGIN
SELECT dname INTO v_description FROM dept WHERE deptno = v_deptno;
RETURN v_description;
```

```
EXCEPTION
WHEN NO_DATA_FOUND
THEN
RETURN('The department is not in the database');
WHEN OTHERS
THEN
RETURN('Error in running show_description');
END;
/
```

Output:

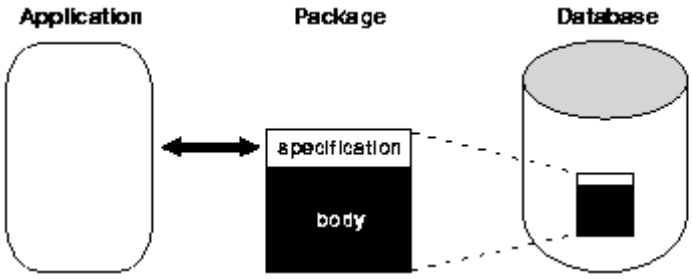
```
SQL> select show_desc(20) from dual;
SHOW_DESC(20)

RESEARCH
SQL> select show_desc(30) from dual;
SHOW_DESC(30)

SALES
SQL> select show_desc(70) from dual;
SHOW_DESC(70)
```

LAB SHEET-X

Program development using creation of package specification, package bodies, private objects, package variables and cursors and calling stored packages.




```
CREATE [OR REPLACE] PACKAGE package_name
  [AUTHID {CURRENT_USER | DEFINER}]
  {IS | AS}
  [PRAGMA SERIALLY_REUSABLE;]
  [collection_type_definition ...]
  [record_type_definition ...]
  [subtype_definition ...]
  [collection_declaration ...]
  [constant_declaration ...]
  [exception_declaration ...]
  [object_declaration ...]
  [record_declaration ...]
  [variable_declaration ...]
  [cursor_spec ...]
  [function_spec ...]
  [procedure_spec ...]
  [call_spec ...]
  [PRAGMA RESTRICT_REFERENCES(assertions)  ...]
END [package_name];
```

```
CREATE OR REPLACE PACKAGE emp_actionsAS  -- spec
  TYPE EmpRecTyp IS RECORD (emp_id INT, salary REAL);
  CURSOR desc_salary RETURN EmpRecTyp;
  PROCEDURE hire_employee (
ename  VARCHAR2,
job    VARCHAR2,
mgr    NUMBER,
sal    NUMBER,
comm   NUMBER,
deptno NUMBER);
  PROCEDURE fire_employee (emp_id NUMBER);
END emp_actions;
```

```
CREATE OR REPLACE PACKAGE BODY emp_actionsAS  -- body
  CURSOR desc_salary RETURN EmpRecTyp IS
     SELECT empno, sal FROM emp ORDER BY sal DESC;
  PROCEDURE hire_employee (
ename VARCHAR2,
job    VARCHAR2,
mgr    NUMBER, sal
NUMBER, comm
NUMBER,


    INSERT INTO emp VALUES (empno_seq.NEXTVAL,
ename, job, mgr, SYSDATE, sal, comm, deptno);
  END hire_employee;

  PROCEDURE fire_employee (emp_id
 NUMBER) IS BEGIN
    DELETE FROM emp WHERE empno =
  emp_id; END fire_employee;
END emp_actions;
```

LAB SHEET-XI

Develop programs using features parameters in a CURSOR, FOR UPDATE CURSOR, WHERE CURRENT of clause and CURSOR variables

```
select *
from emp;
v_empdata empcursor%rowtype;
begin
open empcursor;
loop
  fetch empcursor into v_empdata;
  exit when empcursor%notfound;
  dbms_output.put_line(v_empdata.ename);
end loop;
close empcursor;
end;
/
```

OUTPUT:-

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

PL/SQL procedure successfully completed.
```

/*CURSOR2*/

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

SQL> declare

```
cursor empcursor is
select * from emp;
v_empdata empcursor%rowtype;
begin
open empcursor;
loop
 fetch empcursor into v_empdata;
  exit when empcursor%notfound;
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.
```

LABSHEET -XII

Develop Programs using BEFORE and AFTER Triggers, Row and Statement Triggers and INSTEAD OF Triggers

```
Name                          Null?    Type
-----------------------------------------------------
 ENO                                    NUMBER(38)
ENAME                                   VARCHAR2(12)
EADD                                    VARCHAR2(10)
```

```
SQL>  select * from emp;

 ENO  ENAME         EADD
------          ------------       ---------
111             krishna          hyd
222             vinay            bang
555             kirn             Mumbai
```

```
SQL>create table emp_login(    who          varchar2(10),
                               action       varchar2(10),
                               when         date );
Table created.

SQL>Desc emp_login;
        Name                          Null?    Type
        ----------------
        WHO                                    VARCHAR2(10)
        ACTION                                 VARC HAR2(10)

SQL>Ed Trigger.sql;

CREATE OR REPLACE TRIGGER Emp_hist
 BEFORE  INSERT OR  UPDATE OR  DELETE ON emp
DECLARE
U_action emp_log.action%type;
```

```
BEGIN
        if INSERTING then
        U_action := 'Insert';
elsif UPDATING then
      U_action := 'Update';
elsif DELETING then
        U_action := 'Delete';
else

end if;

END;
```

Executing the file:

SQL> @ Trigger.sql;

Trigger created.

Testing the working of trigger:

Insert into emp (eno,Ename,eadd) values(527,'kpr', 'Kuwait');

1 row created.

SQL> select * from emp;

ENO	ENAME	EADD
111	krishna	hyd
222	vinay	bang
555	kirn	mumbai
527	kpr	kuwait

SQL> select * from emp_log;

WHO	ACTION	WHEN
SYSTEM	Insert	19-MAR-10

----------**********----------