Experiment:1

Database Schema for a customer-sale scenario

Customer (**Cust id : integer,** cust_name: string)

Item (**item_id: integer**, item_name: string, price: integer)

Sale (**bill_no: integer**, bill_data: date, **cust_id: integer, item_id: integer**, qty_sold: integer)

For the above schema, perform the following—

a) Create the tables with the appropriate integrity constraints
b) Insert around 10 records in each of the tables
c) List all the bills for the current date with the customer names and item numbers
d) List the total Bill details with the quantity sold, price of the item and the final amount
e) List the details of the customer who have bought a product which has a price>200
f) Give account of how many products have been bought by each customer
g) Give a list of products bought by a customer having cust_id as 5
h) List the item details which are sold as of today
i) Create a view which lists out the bill_no, bill_date, cust_id, item_id, price, qty_sold, amount
j) Create a view which lists the daily sales date wise for the last one week

Aim: Create the tables with the appropriate integrity constraints and Insert around 10 records in each of the tables

a) SQL>create table customer(cid number(4) primary key, cname varchar2(15) not null);

Name	Null?	Type
CID	NOT NULL	NUMBER(4)
CNAME	NOT NULL	VARCHAR2(15)

SQL> create table item (itemid number(4) primary key, item_name varchar2(20) unique, price number(5));

Name	Null?	Type
ITEMID	NOT NULL	NUMBER(4)
ITEM_NAME		VARCHAR2(20)
PRICE		NUMBER(5)

SQL>create table sale(billno number(5) primary key, billdate date, cid number(4) references customer(cid), itemid number(4) references item(itemid), quantity_sold varchar2(10));

SQL> desc sale

Name	Null?	Type
BILLNO	NOT NULL	NUMBER(5)
BILLDATE		DATE
CID		NUMBER(4)
ITEMID		NUMBER(4)
QUANTITY_SOLD		VARCHAR2(10)

(b) insert into customer values(10,'kumar');

SQL> select * from customer;

CID	CNAME
1	hari
2	pavan
3	damani
4	datri
5	damayanthi
6	suma
7	kamala
8	deepu
9	vijay
10	kumar

SQL> delete from item where itemid=1111;

1 row deleted.

SQL> select * from item;

ITEMID	ITEM_NAME	PRICE
1000	burger	22
1001	rice	42
1002	salt	13
1003	sugar	43
1004	jagerry	67
1005	mirchi	7
1006	tamarind	37
1007	pepper	33
1008	hideandseek	35
1009	mariegold	24
1010	bangle	3

b) insert into sale values(2009,'30-oct-2015',3,1002,'4kg');

SQL> select * from sale;

BILLNO	BILLDATE	CID	ITEMID	QUANTITY_S
2000	12-JAN-15	1	1000	2kg
2001	12-JAN-13	2	1001	1kg
2002	01-FEB-14	3	1002	half kg
2003	23-MAR-12	4	1010	4kg
2004	13-APR-10	4	1009	5kg
2005	13-APR-10	5	1008	3kg

2006	10-MAY-11	7	1003	44kg
2007	13-DEC-11	8	1005	44kg
2008	01-NOV-12	9	1006	4kg
2009	30-OCT-15	3	1002	4kg

c) List all the bills for the current date with the customer names and item numbers

SQL>insert into sale values(2011,'16-jul-2015',1,1009,'3kg');

SQL>select c.cid,c.cname,s.billno from customer c,item i,sale s where c.cid=s.cid and i.itemid=s.itemid and s.billdate=to_char(sysdate);

CID	CNAME	BILLNO
1	hari	2011

d) List the total Bill details with the quantity sold, price of the item and the final amount

SQL>select s.billno,s.billdate,s.quantity_sold,i.price, (i.price*s.quantity_sold) from item i,sale s

where i.itemid=s.itemid;

e)List the details of the customer who have bought a product which has a price<20

SQL>select c.cid,c.cname,i.item_name,i.price from customer c,item i,sale s where c.cid=s.cid and i.itemid=s.itemid and i.price<20;

CID	CNAME	ITEM_NAME	PRICE
3	damani	salt	13
4	datri	bangle	3
8	deepu	mirchi	7

3 damani salt 13

f) Give a count of how many products have been bought by each customer

SQL>select c.cid,count(i.itemid)from customer c,item i,sale s where c.cid=s.cid and i.itemid=s.itemid group by(c.cid);

CID	COUNT(I.ITEMID)
1	2
2	1
4	2
5	1
8	1
3	2
7	1
9	1

SQL>select c.cid,i.item_name,c.cname,count(i.itemid)from customer c,item i,sale s where c.cid=s.cid and i.itemid=s.itemid group by(c.cid,c.cname,i.item_name);

CID	ITEM_NAME	CNAME	COUNT(I.ITEMID)
5	hideandseek	damayanthi	1
7	sugar	kamala	1
2	rice	pavan	1
8	mirchi	deepu	1
1	mariegold	hari	1
3	salt	damani	2
1	burger	hari	1

4	bangle	datri	1
4	mariegold	datri	1
9	tamarind	vijay	1

g) Give a list of products bought by a customer having cust_id as 5

SQL>select i.itemid,i.item_name,i.price,c.cid,c.cname from customer c,item i,sale s

where c.cid=s.cid and i.itemid=s.itemid and s.cid=5

ITEMID	ITEM_NAME	PRICE	CID	CNAME
1008	hideandseek	35	5	damayanthi

h)List the item details which are sold as of today

SQL>select i.itemid,i.item_name,i.price from item i,sale s where s.itemid=i.ite

mid and s.billdate='16-jul-2015';

ITEMID	ITEM_NAME	PRICE
1009	mariegold	24

i)Create a view which lists out the bill_no, bill_date, cust_id, item_id, price, qty_sold, amount
SQL>create view cis_view as(select s.billno,s.billdate,c.cid,i.itemid,i.price, s.quantity_sold, (i.price*s.quantity_sold) amount from customer c,item i,sale s where c.cid=s.cid and i.itemid=s.itemid)

View created.

j) Create a view which lists the daily sales date wise for the last one week
SQL>create view cis_view as(select s.billno,s.billdate,c.cid,i.itemid,i.price, s.quantity_sold, (i.price*s.quantity_sold) amount from customer c,item i,sale s where c.cid=s.cid and i.itemid=s.itemid) and s.billdate>'12-jan-2015';
View created.

Experiment:2

Database Schema for a Student Library scenario

Student(**Stud_no : integer,** Stud_name: string)

Membership(**Mem_no: integer**, **Stud_no: integer**)

Book(**book_no: integer**, book_name:string, author: string)

Iss_rec**(iss_no:integer**, iss_date: date, **Mem_no: integer**, **book_no: integer**)

For the above schema, perform the following—

a) Create the tables with the appropriate integrity constraints
b) Insert around 10 records in each of the tables
c) List all the student names with their membership numbers
d) List all the issues for the current date with student and Book names
e) List the details of students who borrowed book whose author is CJDATE
f) Give a count of how many books have been bought by each student
g) Give a list of books taken by student with stud_no as 5
h) List the book details which are issued as of today
i) Create a view which lists out the iss_no, iss _date, stud_name, book name
j) Create a view which lists the daily issues-date wise for the last one week

AIM: Create the tables with the appropriate integrity constraints

Insert around 10 records in each of the tables

a) create table student(sid number(3) primary key, sname varchar2(15));

SQL> desc student;

Name	Null?	Type
SID	NOT NULL	NUMBER(3)
SNAME		VARCHAR2(15)

SQL> insert into student values (10,'rahul');

1 row created.

SQL> select * from student;

SID SNAME

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

1 deepu

2 lavanya

3 ujjwala

4 kiran

5 kamala

6 vijay

7 ratnam

8 mani

9 ravi

10 rahul

SQL> create table membership(mid number(4) primary key, sid number(3) references student(sid));

Table created.

SQL> desc membership;

Name Null? Type

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

MID NOT NULL NUMBER(4)

SID NUMBER(3)

SQL> insert into membership values(100,1);

10 rows selected.

SQL> select * from membership;

MID SID

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

100 1

101	2
102	3
103	4
104	5
105	6
106	7
107	8
108	9
109	10

SQL> **create table book(bid number(4) primary key, bname varchar2(15), bauthor varchar2(30));**

Table created.

SQL> desc book;

Name	Null?	Type
BID	NOT NULL	NUMBER(4)
BNAME		VARCHAR2(15)
BAUTHOR		VARCHAR2(30)

SQL> select * from book;

BID	BNAME	BAUTHOR
2000	se	pressman
2001	stm	boris
2002	dbms	korth
2003	ppl	andrew

2004	cn	tanenbaum
2005	os	galvin
2006	uml	booch
2007	c	vara
2008	www	lee
2009	co	archi
2010	android	Richard

SQL> **create table issue(issueno varchar2(10) primary key, issuedate date, mid number(3) references membership(mid), bid number(4) references book(bid));**

Table created.

SQL> desc issue;

Name	Null?	Type
ISSUENO	NOT NULL	VARCHAR2(10)
ISSUEDATE		DATE
MID		NUMBER(3)
BID		NUMBER(4)

b) SQL> insert into issue values('a10','12-apr-2015',102,2006);

SQL> select * from issue;

ISSUENO	ISSUEDATE	MID	BID
3000	22-JUL-15	100	2000
3001	02-JAN-13	101	2001
3002	12-FEB-15	102	2003
3003	02-MAR-15	103	2004

3004	29-JUL-15	104	2005
3005	09-JUL-15	106	2006
3006	19-JUL-15	107	2009
3007	09-AUG-14	108	2010
3008	19-APR-14	109	2008
3009	09-MAY-15	108	2007
3010	09-MAY-15	105	2003

c) List all the student names with their membership numbers

SQL> select s.sname,m.mid,s.sid from student s,membership m where s.sid=m.sid;

SNAME	MID	SID
deepu	100	1
lavanya	101	2
ujjwala	102	3
kiran	103	4
kamala	104	5
vijay	105	6
ratnam	106	7
mani	107	8
ravi	108	9
rahul	109	10

d) List all the issues for the current date with student and Book names

SQL> select s.sname,b.bname,i.issuedate from student s,membership m,book b,issue i where s.sid=m.sid and m.mid=i.mid and b.bid=i.bid and i.issuedate=to_char(sysdate);

SNAME	BNAME	ISSUEDATE

--- --- ---
kamala os 29-JUL-15

e) List the details of students who borrowed book whose author is CJDATE

```
SQL> select s.sid,s.sname,b.bauthor,i.issuedate,i.issueno
from student s,book b,membership m,issue i
where s.sid=m.sid and m.mid=i.mid and b.bid=i.bid and b.bauthor='pressman';
```

SID	SNAME	BAUTHOR	ISSUEDATE	ISSUENO
1	deepu	pressman	22-JUL-15	3000

f) Give a count of how many books have been bought by each student

```
SQL> select s.sid,count(distinct s.sid) from student s,membership m,book b,issue i
      where s.sid=m.sid and m.mid=i.mid and b.bid=i.bid group by(s.sid);
```

SID	COUNT(DISTINCTS.SID)
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

g) Give a list of books taken by student with stud_no as 5

```
SQL> select b.bid,b.bname,i.issuedate from student s,membership m,book b,issue i
where s.sid=m.sid and m.mid=i.mid and b.bid=i.bid and s.sid=5;
```

BID	BNAME	ISSUEDATE
2005	os	29-JUL-15
2000	se	29-JUL-15

h) List the book details which are issued as of today

```
SQL> select b.bid,b.bname,i.issuedate from book b,issue i
     where b.bid=i.bid and i.issuedate=to_char(sysdate);
```

BID	BNAME	ISSUEDATE

```
 2005    os              29-JUL-15
 2000    se              29-JUL-15
```

i) Create a view which lists out the iss_no, iss _date, stud_name, book name

SQL> create view smbi_view as(select s.sname,s.sid,i.issueno,b.bname from student s,book b,issue i,membership m where s.sid=m.sid and m.mid=i.mid and b.bid=i.bid);

View created.

SQL> select * from smbi_view;

```
SNAME    SID    ISSUENO    BNAME
---------- ------- ----------    ---------------
deepu       1     3000          se
lavanya     2      3001         stm
ujjwala     3      3002         ppl
kiran       4      3003         cn
kamala      5      3004         os
ratnam      7      3005         uml
mani        8      3006         co
ravi        9      3007         android
rahul       10     3008         www
ravi        9     3009          c
vijay       6     3010          ppl

SNAME  SID        ISSUENO    BNAME
---------- ---------- ----------       ---------------
kamala    5           3012           se

12 rows selected.
```

j) Create a view which lists the daily issues-date wise for the last one week

SQL> create view did as(select i.issueno,i.issuedate from issue i
where issuedate between '20-jul-2015' and '31-jul-2015');

View created.

SQL> select * from did;

```
ISSUENO   ISSUEDATE
---------- ----------------------
3000          22-JUL-15
3004          29-JUL-15
3012          29-JUL-15
```

Experiment:3

Database Schema for a Employee-pay scenario

employee(**emp_id : integer**, emp_name: string)

department(**dept_id: integer**, dept_name:string)

paydetails(**emp_id : integer**, **dept_id: integer**, basic: integer, deductions: integer, additions: integer, DOJ: date)

payroll**(emp_id : integer**, pay_date: date)

For the above schema, perform the following—

a) Create the tables with the appropriate integrity constraints
b) Insert around 10 records in each of the tables
c) List the employee details department wise
d) List all the employee names who joined after particular date
e) List the details of employees whose basic salary is between 10,000 and 20,000
f) Give a count of how many employees are working in each department
g) Give a names of the employees whose netsalary>10,000
h) List the details for an employee_id=5
i) Create a view which lists out the emp_name, department, basic, dedeuctions, netsalary
j) Create a view which lists the emp_name and his netsalary

a) SQL> create table employee(eid number(2) primary key, ename varchar2(15) not null unique);

SQL> desc employee;

Name	Null?	Type
EID	NOT NULL	NUMBER(2)
ENAME	NOT NULL	VARCHAR2(15)

SQL> insert into employee values(10,'kiran');

SQL> select * from employee;

EID	ENAME

1	deepu
2	naresh
3	ujj
4	lavanya
5	pavan
6	paritosh
7	harsha
8	swaga
9	venkat
10	kiran

SQL> create table department(depid number(4) primary key,

2 dept_name varchar2(10));

SQL> desc department;

Name	Null?	Type
DEPID	NOT NULL	NUMBER(4)
DEPT_NAME		VARCHAR2(10)

SQL> select * from department;

DEPID	DEPT_NAME
100	cse
101	it
102	ece
103	ce
104	mec

105 chem

106 eee

107 ins

108 hs

109 biology

110 humanities

SQL> **create table paydetails(**

2 eid number(2) references employee(eid) primary key,

3 depid number(3) references department(depid),

4 deductions number(4) not null,

5 additions number(5) not null

Doj date);

L> select * from paydetails;

EID	DEPID	BASIC	DEDUCTIONS	ADDITIONS	DOJ
2	101	7000	300	405	12-JUN-14
3	102	7000	300	405	12-JUN-14
1	101	6000	202	400	12-JAN-14
4	103	6000	305	400	12-FEB-14
5	104	6005	355	300	12-MAR-15
6	105	9005	111	100	31-JUL-15
7	107	8005	177	154	29-JUL-15
8	107	5305	127	124	01-APR-15
9	109	3305	117	114	11-APR-15
10	110	3505	157	164	11-AUG-14

SQL> create table payroll(eid references employee(eid) primary key,

2 paydate date);

SQL> desc payroll;

Name	Null?	Type
EID	NOT NULL	NUMBER(2)
PAYDATE		DATE

SQL> select * from payroll;

EID	PAYDATE
1	30-JAN-15
2	30-JAN-15
3	27-FEB-15
4	27-MAR-15
5	07-JUN-15
6	17-JAN-15
7	07-JUL-15
8	08-MAR-15
9	08-MAR-15
10	8-MAR-15

c) List the employee details department wise

```
SQL> select e.eid,e.ename,d.depid,d.dept_name
  2  from employee e,department d,paydetails p
  3  where e.eid=p.eid and d.depid=p.depid;
```

EID	ENAME	DEPID	DEPT_NAME
1	deepu	101	it

2	naresh	101	it
3	ujj	102	ece
4	lavanya	103	ce
5	pavan	104	mec
6	paritosh	105	chem
8	swaga	107	ins
7	harsha	107	ins
9	venkat	109	biology
10	iran	110	humanities

d) List all the employee names who joined after particular date

```
SQL> select e.eid,e.ename,p.doj,d.dept_name
  2  from employee e,department d,paydetails p
  3  where e.eid=p.eid and d.depid=p.depid and p.doj>'20-jan-2014';
```

EID	ENAME	DOJ	DEPT_NAME
2	naresh	12-JUN-14	it
3	ujj	12-JUN-14	ece
4	lavanya	12-FEB-14	ce
5	pavan	12-MAR-15	mec
6	paritosh	31-JUL-15	chem
8	swaga	01-APR-15	ins
7	harsha	29-JUL-15	ins
9	venkat	11-APR-15	biology
10	iran	11-AUG-14	humanities

e) List the details of employees whose basic salary is between 10,000 and 20,000

```
SQL> select e.eid,e.ename,d.dept_name,d.depid,p.basic
  2  from employee e,department d,paydetails p
  3  where e.eid=p.eid and d.depid=p.depid and p.basic between 2000 and 7000;
```

EID	ENAME	DEPT_NAME	DEPID	BASIC
2	naresh	it	101	7000
1	deepu	it	101	6000
3	ujj	ece	102	7000
4	lavanya	ce	103	6000
5	pavan	mec	104	6005
8	swaga	ins	107	5305
9	venkat	biology	109	3305
10	kiran	humanities	110	3505

f) Give a count of how many employees are working in each department

```
SQL> select count(eid),depid from paydetails p group by(depid);
```

```
COUNT(EID)          DEPID
---------- -------------------------
         1                  102
         1                  110
         2                  101
         2                  107
         1                  104
         1                  105
         1                  109
         1                  103
```

g) Give a names of the employees whose netsalary>10,000

```
SQL> select e.ename from employee e where e.eid in(
  2  select e.eid from paydetails p where p.eid=e.eid and (p.basic-p.deductions)
>7000);

ENAME
---------------
paritosh
harsha
```

h) List the details for an employee_id=5

```
SQL> select * from employee where eid=5;

      EID ENAME
---------- ---------------
         5  pavan
```

i) Create a view which lists out the emp_name, department, basic, dedeuctions, netsalary

```
SQL> create view edp_view as(select e.ename,e.eid,d.depid,p.basic,p.deductions,(
p.basic-p.deductions) netsalary from
  2  employee e,department d,paydetails p
  3  where e.eid=p.eid and d.depid=p.depid);

View created.
```

SQL> select * from edp_view;

ENAME	EID	DEPID	BASIC	DEDUCTIONS	NETSALARY
naresh	2	101	7000	300	6700
ujj	3	102	7000	300	6700
deepu	1	101	6000	202	5798
lavanya	4	103	6000	305	5695
pavan	5	104	6005	355	5650
paritosh	6	105	9005	111	8894
harsha	7	107	8005	177	7828
swaga	8	107	5305	127	5178
venkat	9	109	3305	117	3188
kiran	10	110	3505	157	3348

SQL> select * from enet_view;

EID	ENAME	NETSALARY
2	naresh	6700
3	ujj	6700
1	deepu	5798
4	lavanya	5695
5	pavan	5650
6	paritosh	8894
7	harsha	7828
8	swaga	5178

9	venkat	3188
10	kiran	3348

Experiment: 4

Database Schema for a Video Library scenario

Customer(**cust_no**: integer,cust_name: string)

Membership(**Mem_no**: integer, cust_no: integer)

Cassette(**cass_no:**integer, cass_name:string, Language: String)

Iss_rec(iss_no: integer, iss_date: date, mem_no: integer, cass_no: integer)

For the above schema, perform the following—

a) Create the tables with the appropriate integrity constraints
b) Insert around 10 records in each of the tables
c) List all the customer names with their membership numbers
d) List all the issues for the current date with the customer names and cassette names
e) List the details of the customer who has borrowed the cassette whose title is " The Legend"
f) Give a count of how many cassettes have been borrowed by each customer
g) Give a list of book which has been taken by the student with mem_no as 5
h) List the cassettes issues for today
i) Create a view which lists outs the iss_no, iss_date, cust_name, cass_name
j) Create a view which lists issues-date wise for the last one week

AIM: Create the tables with the appropriate integrity constraints

Insert around 10 records in each of the tables

SQL>create table customer(cust_no number(5) primary key,cust_name varchar2(20));

SQL>desc customer;

Name	Null?	Type
CUST_NO	NOT NULL	NUMBER(5)
CUST_NAME		VARCHAR2(20)

SQL>insert into customer values(&cust_no,'&cust_name');

SQL>select * from customer;

CUST_NO	CUST_NAME

…….

50	scott
51	pandey
52	varshney
53	naidu
54	bhimbra

SQL>create table membership(mem_no number(5) primary key,cust_no number(5) references customer(cust_no));

SQL>dsec membership;

Name Null? Type

……...

MEM_NO NOT NULL NUMBER(5)

CUST_NO NUMBER(5)

SQL>insert into memshipvalues(&mem_no,&cust_no);

SQL>select * from memship;

MEM_NO CUST_NO

……

920 50

981 51

897 52

820 53

928 54

SQL>create table cassette(cass_no number(5) primary key,

Cass_namevarchar2(15),language varchar2(15));

SQL>desc cassette;

Name Null? Type

……..

CASS_NO	NOT NULL	NUMBER(5)
CASS_NAME		VARCHAR2(15)
LANGUAGE		VARCHAR2(15)

SQL>insert into cassette values(&cass_no,'&cass_name','&language');

SQL>select * from cassette;

CASS_NO	CASS_NAME	LANGUAGE
1	tagore	telugu
2	the lion king	English
3	anniyan	tamil
4	indra	telugu
5	lord of rings	English

SQL>create table issu_rec(iss_no number(5) primary key,iss_datedate,mem_no number(5)references memship(mem_no),cass_no number(5) references cassette(cass_no));

SQL>desc issu_rec;

Name	Null?	Type
ISS_NO	NOT NULL	NUMBER(5)
ISS_DATE		DATE
MEM_NO		NUMBER(5)
CASS_NO		NUMBER(5)

SQL>select * from issu_rec;

ISS_NO	ISS_DATE	MEM_NO	CASS_NO
22	07-JAN-06	920	1

23	10-JAN-00	981	2
26	10-JAN-06	897	5
3	01-JAN-06	820	4
34	31-DEC-05	928	3

c) List all the customer names with their membership numbers

SQL>select c.custname,m.memno from customer1 c,membership1 m where c.custno=m.custno;

CUSTNAME	MEMNO
……..	……..
NIKHIL	51
VIVEK	52
SHRAVAN	58
VAMSI	57
SHIVA	56

d)List all the issues for the current date with the customer names and cassette names

SQL>select i.issno,c.custname,cc.cassettename from customer1 c,membership1 m,cassette cc,issrec1 I where i.issdate=to_char(sysdate) and c.custno=m.custno and i.cassno=cc.cassno and i.memno=m.memno;

OutPut:

no rows selected.

e) List the details of the customer who has borrowed the cassette whose title is " The Legend"
SQL> select c.cid,c.cname from customer c, videpmem m, videoissue I,cassette t where c.cid=m.cid and m.memid=i.memid and i.cassid=t.cassid and t.cassname='The Legend'

cid	cname
……..……..	
1	The Legend

f) Give a count of how many cassettes have been borrowed by each customer

SQL>select v.memid,count(v.memid) from customer c, videomem m,cassette c, videoissue v where c.cid=m.cid and m.memid=v.memid and c.cassid=v.cassid group by(v.memid);

MEMID	COUNT(V.MEMID)

```
--------------------------------------------------
100                             1
108                             1
102                             2
110                             1
101                             1
107                             1
104                             2
109                             1
106                             1

9 rows selected.
```

g) Give a list of book which has been taken by the student with mem_no as 5

SQL> select c.cassid,c.cassname,c.language,v.issueno,v.issuedate from customer c, videomem m,cassette c, videoissue v where c.cid=m.cid and m.memid=v.memid and c.cassid=v.cassis and m.memid=102;

```
CASSID   CASSNAME   LANGUAGE    ISSUENO  ISSUEDATE
-------------------------------------------------------------------------------
102         os              telugu              2002         30-JUL-15
102         os              telugu              2003         04-AUG-15
```

h) List the cassettes issues for today

SQL> select c.cassid,c.cassname,c.language,v.issueno,v.issuedate from customer c, videomem m,cassette c, videoissue v where c.cid=m.cid and m.memid=v.memid and c.cassid=v.cassis and v.issuedate=to_char(sysdate);

```
CASSID   CASSNAME   LANGUAGE    ISSUENO  ISSUEDATE
-------------------------------------------------------------------------------
1006         java             telugu              2006         09-DEC-15
```

i**) Create a view which lists outs the iss_no, iss_date, cust_name, cass_name**

SQL> create view cmav as(select c.cassid,c.cassname,c.language,v.issueno,v.issuedate from customer c, videomem m,cassette c, videoissue v where c.cid=m.cid and m.memid=v.memid and a.cassid=v.cassis);
SQL> select * from cmav;

```
CNAME  CASSNAME   ISSUENO  ISSUEDATE
-------------------------------------------------------------------------------
hari        se              2002         30-JUL-15
pavan       spm             2003         04-AUG-15
damani      os              2004         30-AUG-15
damani      java            2002         30-JUL-15
datri       c               2001         30-AUG-15
suma        ds              2002         22-JUL-15
```

kamala oops 2002 23-AUG-15

7 records selected.

j) Create a view which lists issues-date wise for the last one week

SQL> create view cmav as(select v.issueno,v.issuedate, a.cassid, c.cassname from customer c, videomem m,cassette a, videoissue v where c.cid=m.cid and m.memid=v.memid and a.cassid=v.cassis and v.issuedate between '10-jul-2014' and '18-aug-2015');

ISSUENO ISSUEDATE CASSID CASSNAME

2002 30-JUL-15 1000 se
2003 04-AUG-15 1001 spm
2004 30-AUG-15 1002 se
2002 30-JUL-15 1003 os
2001 30-AUG-15 1004 os
2002 22-JUL-15 1005 os
2002 23-AUG-15 1006 os

Experiment:5

Database Schema for a student-Lab scenario

Student(**stud_no: integer**, stud_name: string, **class: string**)

Class(**class: string, descrip: string**)

Lab(**mach_no: integer**, Lab_no: integer, description: String)

Allotment(**Stud_no: Integer, mach_no: integer, dayof week: string)**

For the above schema, perform the following—

a) Create the tables with the appropriate integrity constraints.
b) Insert around 10 records in each of the tables.
c) List all the machine allotments with the student names, lab and machine numbers.
d) List the total number of lab allotments day wise.
e) Give a count of how many machines have been allocated to the 'CSIT' class.
f) Give a machine allotment details of the stud_no 5 with his personal and class details.
g) Count for how many machines have been allocated in **Lab_no 1** for the day of the week as "Monday".
h) How many students class wise have allocated machines in the labs.
i) Create a view which lists out the stud_no, stud_name, mach_no, lab_no, dayofweek.
j) Create a view which lists the machine allotment details for "Thursday".

SQL> create table class(cname varchar2(10) primary key, cdesc varchar2(20));

SQL> create table studen(sid number(3) primary key, sname varchar2(10),cname varc

har2(10) references class(cname));

SQL> create table lab(macid number(5) primary key,labid varchar2(10),ldesc varch

ar2(10));

SQL> create table allotment(sid number(3) references studen(sid),macid number(5)

references lab(macid),day varchar2(10));

SQL> desc class;

Name	Null?	Type
CNAME	NOT NULL	VARCHAR2(10)
CDESC		VARCHAR2(20)

SQL> desc studen;

Name	Null?	Type
SID	NOT NULL	NUMBER(3)
SNAME		VARCHAR2(10)
CNAME		VARCHAR2(10)

SQL> desc lab;

Name	Null?	Type
MACID	NOT NULL	NUMBER(5)
LABID		VARCHAR2(10)
LDESC		VARCHAR2(10)

SQL> desc allotment;

Name	Null?	Type
SID		NUMBER(3)
MACID		NUMBER(5)
DAY		VARCHAR2(10)

SQL> select * from class;

CNAME	CDESC
1a	cse
1b	it
1c	ece
1d	cse
2d	chem
2c	civil
2b	eee
2a	cse
3a	it

```
3b        civil
3c        pharma

CNAME      CDESC
---------- --------------------
3d        bpharma
4a        eie
4b        cse
4c        cse
4d        ese

16 rows selected.
```

SQL> select * from studen;

```
      SID SNAME  CNAME
---------- ---------- ----------
      100     hari       1a
      101     mani     1b
      102     pari       1c
      103     pandu    1a
      104     pinky     1c
      105     paritosh 1d
      106     harish     2d
      107     cutie       2c
      108    deepu      2b
      109    ujj          2a
      110    lav          4a

      SID SNAME    CNAME
---------- ---------- ----------
      111  madhu       2a
      112  kari          2c
```

SQL> select * from lab;

```
    MACID LABID      LDESC
---------- ---------- ----------
        1           lab-1      ibm
        2           lab-2      oracle
        3           lab-3      mic
        4           lab-4      pg
        5           lab-1      ibm
        6           lab-2      oracle
        7           lab-3      mic
        8            lab-4      pg
```

9	cc1	first
10	cc2	first

SQL> select * from allotment;

SID	MACID	DAY
100	1	sat
101	2	mon
103	4	tue
103	4	tue
105	5	wed
106	6	thu
107	7	fri
108	8	sat
108	8	sat
109	5	sat
110	2	tue

c) List all the machine allotments with the student names, lab and machine numbers

```
SQL> select l.macid,s.sname,l.labid,s.sid,c.cname
  2  from class c,studen s,lab l,allotment a
  3  where c.cname=s.cname and s.sid=a.sid and l.macid=a.macid;
```

MACID	SNAME	LABID	SID	CNAME
1	hari	lab-1	100	1a
2	lav	lab-2	110	4a
2	mani	lab-2	101	1b
4	pandu	lab-4	103	1a
4	pandu	lab-4	103	1a
5	ujj	lab-1	109	2a
5	paritosh	lab-1	105	1d
6	harish	lab-2	106	2d
7	cutie	lab-3	107	2c
8	deepu	lab-4	108	2b
8	deepu	lab-4	108	2b

11 rows selected.

d) List the total number of lab allotments day wise

```
SQL> select a.macid,l.labid,l.ldesc,a.day from lab l,allotment a where
  2  l.macid=a.macid;
```

MACID	LABID	LDESC	DAY

```
---------- ---------- ---------- --------------------
         1          lab-1      ibm            sat
         2          lab-2      oracle         mon
         4           lab-4      pg             tue
         4          lab-4      pg             tue
         5          lab-1      ibm            wed
         6          lab-2      oracle         thu
         7          lab-3      mic            fri
         8          lab-4      pg             sat
         8          lab-4      pg             sat
         5          lab-1      ibm            sat
         2          lab-2      oracle         tue
```

e) Give a count of how many machines have been allocated to the 'CSIT' class

```
SQL> select count(macid) from allotment where sid in(select sid from studen wher
e cname='4a');

COUNT(MACID)
------------
           1
```

f) Give a machine allotment details of the stud_no 5 with his personal and class details

```
SQL> select a.macid,a.sid,s.sname,c.cname,c.cdesc
  2  from class c,studen s,lab l,allotment a
  3  where c.cname=s.cname and s.sid=a.sid and l.macid=a.macid and s.sid='101'

    MACID     SID    SNAME  CNAME   CDESC
---------- ---------- ---------- ---------- --------------------
         2          101      mani       1b          it
```

g) Count for how many machines have been allocatedin Lab_no 1 for the day of the week as "Monday"

```
SQL> select count(a.macid) from allotment a,lab l where l.macid=a.macid and a.da
y='sat'and l.labid='lab-1';

COUNT(A.MACID)
--------------------------
             2
```

h) How many students class wise have allocated machines in the labs

```
SQL> select count(sid) allocated_students_in_lab
  2  from studen s where sid in(select sid from allotment)group by(cname);
ALLOCATED_STUDENTS_IN_LAB
```

1
2
1
1
1
1
1

i) Create a view which lists out the stud_no, stud_name, mach_no, lab_no, dayofweek

```
SQL> create view sl as(select s.sid,s.sname,l.macid,l.labid,a.da
  2  from studen s,lab l,allotment a
  3  where s.sid=a.sid and l.macid=a.macid);
```

View created.

SQL> select * from sl;

SID	SNAME	MACID	LABID	DAY
100	hari	1	lab-1	sat
101	mani	2	lab-2	mon
103	pandu	4	lab-4	tue
103	pandu	4	lab-4	tue
105	paritosh	5	lab-1	wed
106	harish	6	lab-2	thu
107	cutie	7	lab-3	fri
108	deepu	8	lab-4	sat
108	deepu	8	lab-4	sat
109	ujj	5	lab-1	sat
110	lav	2	lab-2	tue

11 rows selected.

j) Create a view which lists the machine allotment details for "Thursday".

SQL> create view st as(select a.macid,l.labid,a.day,s.sid from studen s,lab l,al

lotment a where s.sid=a.sid and l.macid=a.macid and a.day='thu');

View created.

SQL> select * from st;

MACID	LABID	DAY	SID
6	lab-2	thu	106

PL/SQL INTRODUCTON

- PL/SQL is Oracle's *procedural* language extension to SQL

  PL/SQL allows sending an entire block of statements to the database at one time.
  PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.

- PL/SQL blocks contain three sections
  - Declare section
  - Executable section and
  - Exception-handling section.
- The executable section is the only mandatory section of the block.
- Both the declaration and exception-handling sections are optional.
- PL/SQL block has the following structure:

DECLARE

Declaration statements

BEGIN

Executable statements

EXCETION

Exception-handling statements

END ;

Declaration section:

- The *declaration section* is the first section of the PL/SQL block.
- It contains definitions of PL/SQL identifiers such as variables, constants, cursors and so on.
- Example

```
DECLARE
    v_first_name VARCHAR2(35) ;
```

```
    v_last_name VARCHAR2(35) ;
    v_counter NUMBER := 0 ;
```

executable section

- The executable section is the next section of the PL/SQL block.
- This section contains executable statements that allow you to manipulate the variables that have been declared in the declaration section.

```
begin
    SELECT first_name, last_name
        INTO v_first_name, v_last_name
        FROM student
        WHERE student_id = 123 ;
    DBMS_OUTPUT.PUT_LINE
    ('Student name :' || v_first_name ||'  '|| v_last_name);
END;
```

Exception handling section:

- The *exception-handling section* is the last section of the PL/SQL block.
- This section contains statements that are executed when a runtime error occurs within a block.
- Runtime errors occur while the program is running and cannot be detected by the PL/SQL compiler.

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE
        (' There is no student with student id 123 ');
END;
```

DBMS_OUTPUT.PUT_LINE in your executable section to display any message you

want to the screen.

Syntax for displaying a message:

DBMS_OUTPUT.PUT_LINE(<string>);

in which PUT_LINE is the procedure to generate the output on the screen, and DBMS_OUTPUT is the package to which the PUT_LINE belongs.

DBMS_OUTPUT_PUT_LINE('My age is ' || num_age);

Output:

SET SERVEROUTPUT ON;

SAMPLE PROGRAM:

```
 begin
  Dbms_output.put_line('hello');
  end;
  /
hello
```

program: 2 DISPLAY HI ENAME

```
   declare
    ename varchar2(10):='sid';
    begin
    dbms_output.put_line('hello'|| ename);
   end;
SQL> /
hellosid
```

SAMPLE PROGRAM: PL/SQL FOR Adding two numbers

```
 declare
   a number;
  b number;
  c number;
```

```
  begin
  a:=&a;
  b:=&b;
  c:=a+b;
  Dbms_output.put_line(c);
  end;
 /
Enter value for a: 50
old   6: a:=&a;
new   6: a:=50;
Enter value for b: 60
old   7: b:=&b;
new   7: b:=60;
110
```

Experiment:6

Write a program to find largest number from the given three numbers.

Aim: To find largest number from the given three numbers.

Algorithm:

Step 1: Declare the variable A, B, and C.
Step 2: Store the valid data.
Step 3: Compare variable A with B and A with C
Step 4:If the value stored in variable A is big, it displays "A is Big". (IF conditional statement should be used)
Step 5: Compare variable B with C
Step 6: If the value stored in variable B is big, it displays "B is Big".
Step 7: Other wise it displays "C is Big"

```
declare

a number;

b number;

c number;

begin

a:=&a;

b:=&b;

c:=&c;

if (a>b and a>c) then

Dbms_output.put_line('a is big');

else if(b>a and b>c) then

Dbms_output.put_line('b is big');

else

Dbms_output.put_line('c is big');

end if;

end if;
```

```
end;
/
```

Output:

```
Enter value for a: 50
old   6: a:=&a;
new   6: a:=50;
Enter value for b: 60
old   7: b:=&b;
new   7: b:=60;
Enter value for c: 70
old   7: c:=&c;
new   7: c:=70;
c is big.
```

Experiment:7

Simple programs using loop, while and for iterative control statement.

a) To generate first 10 natural numbers using **loop, while** and **for**.

Algorithm:

Step 1: Declare the variable I.
Step 2: Store the valid data 1 in I.
Step 3: Use **LOOP** statement
Step 4: Display the first value.
Step 5: Increment the value of I by 1 value.
Step 6: check the value up to 10 no. and repeat the loop
Step 7: If condition exceeds the given value 10, the loop will be terminated.

/* using loop statement */

```
 declare
  I number;
  begin
     I: =1;
  loop
     Dbms_output.put_line(I);
     I: =I+1;
     exit when I>10;
  end loop;
  end;
  /
```

```
1
2
3
4
5
6
7
8
9
10
PL/SQL procedure successfully completed.
```

Algorithm: for WHILE loop

Step 1: Declare the variable I.
Step 2: Store the valid data 1 in I.
Step 3: Use **WHILE** statement
Step 4: Check the value of I with value 10.
Step 5: if the value of I reached to 10 the loop will be terminated
Step 6: otherwise display value of I
Step 7: increment the next value of I using I=I+1.

```
/* using while */
 declare
  I number;
  begin
     I: =1;
  while (I<=10)
  loop
     Dbms_output.put_line(I);
     I: =I+1;
  end loop;
  end;
  /
1
2
3
4
5
6
7
8
9
10
```

PL/SQL procedure successfully completed.

Algorithm:
Step 1: Declare the variable I.
Step 2: Store the value 1 in var. I.
Step 3: Use **For**… **LOOP** statement

Step 4: Display the first value of I.
Step 5: Increment the value of I by 1 value.
Step 6: check the value up to 10 no. and repeat the loop
Step 7: if the loop exceeds the value 10 then the loop will be terminated.

```
/* using for loop*/
begin
for I in 1..10
loop
        Dbms_output.put_line(I);
end loop;
end;
1
2
3
4
5
6
7
8
9
10
PL/SQL procedure successfully completed.
```

Experiment:8

Program to check whether given number is Armstrong or not.

Algorithm:

Step 1: Declare the variable N, S, D and DUP.
Step 2: Store the value in var. N and var. DUP..
Step 3: check for the value of N, which is not equal to 0.
Step 4: divide value stored in N by 10 and store it var. D. (D=n%10).
Step 5: the remainder will be multiply 3 times and store it in Var. S.
Step 6: The coefficient will be calculated using FLOOR function. And store it in var. N.
Step 7: repeat the Steps 3, 4, 5, and 6 till loop will be terminated.
Step 8: Check whether the stored value and calculated values are same
Step 9: if both the values are same, then display "The given number is Armstrong"
Step 10: Otherwise display "it is not Armstrong" and terminate the loop.

```
declare
N number;
m number;
s number;
D number;
begin
        N:=&N;
      m:=N;
        s:=0;
while(n<>0)
loop
        D:=n mod 10;
        s:=s+(D*D*D);
        N:=floor(N/10);
end loop;
if (m=s) then
        DBMS_output.put_line('number is armstrong');
else
        DBMS_output.put_line('number is not armstrong');
end if;
end;
```

Test Valid Data Set:

Enter value of n

153

OUTPUT:

number is Armstrong.

.

Experiment:9

Write a program to generate all prime numbers below 100.

```
declare
    Flag Boolean:=FALSE;
    begin
    for i in 2..100 LOOP
      for j in 2..i-1 LOOP
       if mod(i,j)=0 then
          Flag:=TRUE;
          exit;
       end if;
      end loop;
        IF Flag != TRUE THEN
          DBMS_OUTPUT.PUT_LINE(i);
        END IF;
        Flag:=FALSE;
      end loop;
   end;
 /
```

Valid Test Data

OUTPUT:

2
3
5
7
11
99

Experiment:10

Write a program to demonstrate the GOTO statement.

Aim: to demonstrate the GOTO statement

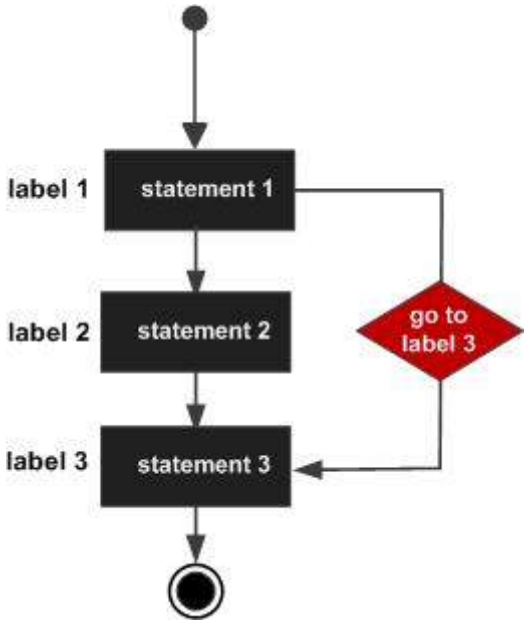
HW/SW requirements:

Processor	:	AMD Athelon ™ 1.67 GH_z
RAM	:	256 MB
Hard Disk	:	40 GB
Software	:	Oracle, PlSQL

A **Goto** Statement In Pl/Sql Programming Language Provides An Unconditional Jump From The Goto To A Labeled Statement In The Same Subprogram.

Note: Use Of Goto Statement Is Highly Discouraged In Any Programming Language Because It Makes Difficult To Trace The Control Flow Of A Program, Making The Program Hard To Understand And Hard To Modify. Any Program That Uses A Goto Can Be Rewritten So That It Doesn't Need The Goto.

Flow Diagram:




```
declare
 i number;
```

```
begin
i:=&i;
If(i>=0) then
 GOTO second_output;
else
  DBMS_output.put_line('This line will never execute.');
end if;
  <<second_output>>
  Dbms_output.put_line('We are here!');
   end;
  /
```

Enter value for i: 0

old 4: i:=&i;

new 4: i:=0;

We are here!

PL/SQL procedure successfully completed.

SQL> set serveroutput on;

SQL> /

Enter value for i: -1

old 4: i:=&i;

new 4: i:=-1;

We are here!

PL/SQL procedure successfully completed.

SQL> set serveroutput on;

SQL> /

Enter value for i: 10

old 4: i:=&i;

new 4: i:=10;

This line will never execute.

We are here!

Experiment:11

Write a program to demonstrate %type and %rowtype attributes

Aim: to demonstrate %type and %rowtype attributes

```
declare
    my_empid emp.empid%type;
   my_ename emp.ename%type;
   my_emprow emp%rowtype;
   no number;
   begin
   no:=&no;
   select empid,ename into my_empid,my_ename from emp where empid=no;
   if(SQL%rowcount=1) then
  Dbms_output.put_line('empid is' || my_empid);
 else
  Dbms_output.put_line('error');
 end if;
   select * into my_emprow from emp where empid=no;
    if(SQL%rowcount=1) then
Dbms_output.put_line('empidis'||my_emprow.empid||'ename is'||my_emprow.ename);
   else
   Dbms_output.put_line('error');
   end if;
   end;
 /
```

Enter value for no: 2
old 7: no:=&no;
new 7: no:=2;
empid is2ename iss
empid is2ename iss

PL/SQL procedure successfully completed.

Experiment:12

Write a program to demonstrate predefined exceptions

An error condition during a program execution is called an exception in PL/SQL. PL/SQL supports programmers to catch such conditions using**EXCEPTION** block in the program and an appropriate action is taken against the error condition. There are two types of exceptions:

- System-defined exceptions
- User-defined exceptions

Aim: to demonstrate predefined exceptions

HW/SW requirements:

Processor	:	AMD Athelon ™ 1.67 GH$_z$
RAM	:	256 MB
Hard Disk	:	40 GB
Software	:	Oracle, PlSQL

```
declare
 v number;
 a number;
 b number;
 begin
 a:=&a;
 b:=&b;
 v:=a/b;
 Dbms_output.put_line('the value of v is'||v);
   exception
    when ZERO_DIVIDE then
 Dbms_output.put_line('b could not be zero');
```

```
 end;
 /
Enter value for a: 4
old   6: a:=&a;
new   6: a:=4;
Enter value for b: 0
old   7: b:=&b;
new   7: b:=0;
b could not be zero
PL/SQL procedure successfully completed.
 set serveroutput on;
 /
Enter value for a: 4
old   6: a:=&a;
new   6: a:=4;
Enter value for b: 2
old   7: b:=&b;
new   7: b:=2;
the value of v is2
PL/SQL procedure successfully completed.
```

Experiment:13

Write a program to demonstrate user defined exceptions

Aim: To demonstrate user defined exceptions

```
declare
  a number;
  b number;
  c number;
  mydivide_zero exception;
  begin
  a:=&a;
  b:=&b;
  if(b=0) then
 raise mydivide_zero;
  else
  c:=a/b;
  Dbms_output.put_line('division is'||c);
  end if;
  exception
 when mydivide_zero then
 Dbms_output.put_line('b could not be zero');
  end;
  /
Enter value for a: 4
```

old 7: a:=&a;

new 7: a:=4;

Enter value for b: 2

old 8: b:=&b;

new 8: b:=2;

division is2

Experiment:14

Create a cursor which displays all details of emp table

```
set serveroutput on;
declare
   cursor my_cur is select empid,ename from emp;
   xempid emp.empid%type;
   xename emp.ename%type;
   begin
   open my_cur;
    loop
    fetch my_cur into xempid,xename;
    Dbms_output.put_line('empid is' || xempid||'ename'||xename);
    exit when my_cur%NOTFOUND;
    end loop;
    close my_cur;
    end;
  /
```

Experiment:15

Create a Cursor which update the salaries of an Employee as follows.

1. if sal<1000then update the salary to 1500.
2. if sal>=1000 and <2000 then update the salary to 2500.
3. if sal>=2000 and <=3000 then update the salary to 4000.

And also count the no.of records have been updated.*/

HW/SW requirements:

Processor	:	AMD Athelon ™ 1.67 GH$_z$
RAM	:	256 MB
Hard Disk	:	40 GB
Software	:	Oracle, PlSQL

```
set serveroutput on;

declare

 cursor my_cur is select empno,sal from emp1;

   xno emp1.empno%type;

   xsal emp1.sal%type;

   c number;

   begin

   open my_cur;

   c:=0;

   loop

  fetch my_cur into xno,xsal;

  if(xsal<3000)then

  update emp1 set sal=100 where empno=xno;

  c:=c+1;
```

```
else if(xsal>=3000 and xsal<6000)then
update emp1 set sal=200 where empno=xno;
c:=c+1;
end if;
end if;
exit when my_cur%NOTFOUND;
end loop;
close my_cur;
Dbms_output.put_line(c||'record have been successfully updated');
end;
/
```

8record have been successfully updated

PL/SQL procedure successfully completed.

SQL> select* from emp;

EMPNO	**SAL**
1	100
2	200
3	200
4	100
5	6444
6	200
7	100

Experiment:16

Create cursor which display all employees whose salary is greater than 50000

```
set serveroutput on;

declare

cursor my_cur is select empno,sal from emp1 where sal>50000;

xno emp1.empno%type;

xsal emp1.sal%type;

begin

open my_cur;

 loop

 fetch my_cur into xno,xsal;

Dbms_output.put_line('empno is' || xno||'salary'||xsal);

exit when my_cur%NOTFOUND;

end loop;

close my_cur;

end;

  /
```

Valid Test Data

Before executing the cursor, the records in emp table as follows

Sql>select * from emp;

A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the calling program.

A subprogram can be created:

- At schema level
- Inside a package
- Inside a PL/SQL block

A schema level subprogram is a **standalone subprogram**. It is created with the CREATE PROCEDURE or CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

A subprogram created inside a package is a **packaged subprogram**. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter 'PL/SQL - Packages'.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms:

- **Functions**: these subprograms return a single value, mainly used to compute and return a value.
- **Procedures**: these subprograms do not return a value directly, mainly used to perform an action.

This chapter is going to cover important aspects of a **PL/SQL procedure** and we will cover **PL/SQL function** in next chapter.

Parts of a PL/SQL Subprogram

Each PL/SQL subprogram has a name, and may have a parameter list. Like anonymous PL/SQL blocks and, the named blocks a subprograms will also have following three parts:

S.N.	Parts & Description
1	**Declarative Part** It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.
2	**Executable Part**

	This is a mandatory part and contains statements that perform the designated action.
3	**Exception-handling** This is again an optional part. It contains the code that handles run-time errors.

Creating a Procedure

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

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

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows modifying an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Example:

```
create or replace procedure greetings
  as
  begin
  Dbms_output.put_line('hello every one');
```

```
   end;
/
```

Procedure created.

SQL> exec greetings;

hello every one

PL/SQL procedure successfully completed.

Deleting a Standalone Procedure

A standalone procedure is deleted with the DROP PROCEDURE statement. Syntax for deleting a procedure is:

```
DROP PROCEDURE procedure-name;
```

So you can drop *greetings* procedure by using the following statement:

Experiment:17

Create procedure to find reverse of number

```
create or replace procedure rproc(n  IN number, rev OUT number) as
declare
x number(9):=0;
r number(3):=0;
sum1 number(10):=0;
rev number(10):=0;
n1 number(10):=0;
begin
n1:=n;
for i in 1..10
loop
if (n1 <> 0) then
r:=mod(n1,10);
sum1:=(sum1*10)+r;
n1:=n1/10;
end if;
end loop;
rev:=sum1;
end;
/

 declare
  t number;
  begin
  rproc(t);
```

```
dbms_output.put_line('the reverse of number'|| t);
end;
/
```

Experiment:18

Create a procedure which updates the salaries of an employees as follows.

1.if sal<3000 then update the salry to 1500.

2.if sal>=3000 and <=6400 then update the salary to 2500.*/

```
create or replace procedure myproc as
  cursor my_cur is select empno,sal from emp2;
  xno emp2.empno%type;
  xsal emp2.sal%type;
  c number;
  begin
  open my_cur;
  c:=0;
  loop
  fetch my_cur into xno,xsal;
  if(xsal<3000) then
  Update emp2 set sal=15 where empno=xno;
  c:=c+1;
  else
  if(xsal>=3000 and xsal<=6400) then
  update emp2 set sal=25 where empno=xno;
  c:=c+1;
  end if;
  end if;
 exit when my_cur%NOTFOUND;
  end loop;
  close my_cur;
  Dbms_output.put_line(c||'records have been successfully updated');
  end;
  /
Procedure created.
SQL> set serveroutput on;
SQL> /
Procedure created.
```

SQL> exec myproc;

8records have been successfully updated

PL/SQL procedure successfully completed.

SQL> select * from emp;

EMPNO	SAL
1	122
2	234
3	434
4	2004
5	1004
6	2104
7	1999

SQL> select * from emp;

EMPNO	SAL
1	1500
2	1500
3	1500
4	2500
5	2500
6	2500
7	2500

7 rows selected.

Experiment: 19

Create a procedure which generate all the prime numbers below the given number and count the no.of prime numbers.

```
create or replace procedure prime_proc(n IN number, tot OUT number) as
 i number;
c number;
j number;
 begin
 i:=1;
 tot:=0;
 while(i<=n)
 loop
 j:=1;
 c:=0;
 while(j<=i)
 loop
 if(mod(i,j)=0) then
 c:=c+1;
 end if;
 j:=j+1;
 end loop;
 if(c=2) then
 dbms_output.put_line(i);
 tot:=tot+1;
 end if;
 i:=i+1;
 end loop;
 end;
 /
Procedure created.
SQL> set serveroutput on;
SQL> /

 declare
```

```
 t number;
 begin
 prime_proc(100,t);
 dbms_output.put_line('the total prime no are'|| t);
 end;
 /
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
the total prime no are15
```

PL/SQL procedure successfully completed.

Procedure created

Experiment:

PL/SQL PROGRAM FOR PALINDROME OF STRING

```
declare
      g varchar2(20);
      r varchar2(20);
      i number(4);
begin
      g:='&g';
      for i in reverse 1.. length(g) loop
            r:=r || substr(g,i,1);

      end loop;
```

```
        dbms_output.put_line('reverse string is ' || r);
        if r=g then
        dbms_output.put_line('String is Palindrome');
        else
        dbms_output.put_line('String is not Palindrome');
        end if;
end;
```

Experiment:20

Create a function to check whether given is palindrome or not

```
declare
s1 varchar2(20);
 s2 varchar2(20);
 begin s1:='&s1';
 select reverse(s1) into s2 from dual;
 if s1=s2 then dbms_output.put_line('given string is palindrome');
else
 dbms_output.put_line('given string is not palindrome');
 end if;
 end;
 /
Number palindrome or not
 declare
   n number(3);
   i number(3);
  sum1 number(3);
  k number(3);
   begin
   sum1:=0;
   n:=&n;
   k:=n;
   while (n>0) loop
   i:=mod(n,10);
   sum1:=(sum1*10)+i;
   n:=trunc(n/10);
   end loop;
   if(k=sum1) then
   dbms_output.put_line('Given Number is a Palindrome Number');
  else
  dbms_output.put_line('Given Number is not a Palindrome Number');
  end if;
```

```
 end;
/
```

Experiment:

/* create function which add two given numbers. (Simple programs) */

```
create or replace function add_fun(a number,b number) return
number as
c number;
begin
c:=a+b;
return c;
end;
/
```

Function created.

/*add_fun specification*/

```
declare
result varchar2(10);
begin
result:=add_fun(10,11);
Dbms_output.put_line('the addition is'||result);
end;
/
```

The sum of 10 and 20 is 30

Pl/sql procedure successfully completed.

Experiment: 21

Create a function to find sum of salaries of all employees working in depart number 10./*function body*/

```
create or replace function count_emp1(sal number)return number as
cursor vin_cur as Select empno,sal from emp1;
xno emp1.empno%type;
xsal emp1.sal%type;
c number;
begin
open vin_cur;
c:=0;
loop
fetch vin_cur into xno,xsal;
if(xsal<sal) then
c:=c+1;
end if;
  exit when vin_cur%notfound;
end loop;
close vin_cur;
return c;
end;
/
```

Function created.

/*function specification*/

```
declare
ne number;
xsal number;
begin
ne:=count_emp1(sal);
Dbms_output.put_line(xsal);
Dbms_output.put_line('there are '||ne||;employees');
end; /
```

OUTPUT

There are 8 employees.

Triggers

triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATES, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes:

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

The syntax for creating a trigger is:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
```

```
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name: Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Experiment:22

Create a trigger before/after update on employee table for each row/statement.

trigger before update

```
create or replace trigger emp_trig before update on emp1 for each row
 begin
if (:old.sal>3500)
then raise_application_error(-20000,'Not allowed to update');
 end if;
end;
```

trigger after update

```
create or replace trigger emp_trig after  update on emp1 for each row
 begin
if (:old.sal>3500)
then raise_application_error(-20000,'Not allowed to update');
 end if;
end;
-create or replace trigger a1 before  insert on emp_tri  for each row
begin
update  budget  set total_budget:= total_budget + :new.salary where dno:= new.dno;
end;
/
```

SQL>insert into employee values('shiva',111,500000);

OUTPUT:

SQL>select * from employee;

0 rows selected

Experiment: 23

Create a trigger before/after delete on employee table for each row/statement

trigger before delete

```
create or replace trigger emp_trig before delete on emp1 for each row
 begin
if (:old.sal>2500)
then raise_application_error(-20000,'Not allowed to update');
 end if;
end;
```

trigger after delete

```
create or replace trigger emp_trig after delete on emp1 for each row
 begin
if (:old.sal>3000)
then raise_application_error(-20000,'Not allowed to update');
 end if;
end;
```

Experiment: 24

Create a trigger before/after insert on employee table for each row/statement

trigger after insert

```
create or replace trigger emp_trig after insert on emp1 for each row
 begin
if (:new.sal>100)
then raise_application_error(-20000,'Not allowed to update');
 end if;
end;
```

A **stored procedure** or in simple a **proc** is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.

A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block.

A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

Procedures: Passing Parameters

We can pass parameters to procedures in three ways.
1) IN-parameters
2) OUT-parameters
3) IN OUT-parameters

A procedure may or may not return any value.

```
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.

The syntax within the brackets [ ] indicate they are optional. By using CREATE OR REPLACE together the procedure is created if no other procedure with the same name exists or the existing procedure is replaced with the current code.

A schema level subprogram is a **standalone subprogram**. It is created with the CREATE PROCEDURE or CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

A subprogram created inside a package is a **packaged subprogram**. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter 'PL/SQL - Packages'.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms:

- **Functions**: these subprograms return a single value, mainly used to compute and return a value.
- **Procedures**: these subprograms do not return a value directly, mainly used to perform an action.

IN An IN parameter lets you pass a value to the subprogram. **It is a read-only parameter**. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. **It is the default mode of parameter passing. Parameters are passed by reference.**	
2	**OUT** An OUT parameter returns a value to the calling program.

	Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. **The actual parameter must be variable and it is passed by value**.
2	**IN OUT** An IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and its value can be read.