Your need, Our Potential

SQL QUERIES
&
PL/SQL PROGRAMMING

Powered by C.Sivanantham

C.Sivanantham

PART – I

SQL QUERIES

C.Sivanantham

CREATE A TABLE

Syntax

Create table <table name> (column definition 1,column definition 2, ... ,);

Example

```
SQL> create table bab(name varchar2(10), eng number(3), tam number(3));
Table created.
```

TO VIEW THE TABLE STRUCTURE

Syntax

Desc <table name>

Example

```
SQL> desc bab;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 NAME                                               VARCHAR2(10)
 ENG                                                NUMBER(3)
 TAM                                                NUMBER(3)
```

ALTER A TABLE

Syntax

Alter table <table name> modify (column definition...);

Alter table <table name> add (column definition...);

Example

```
SQL> alter table bab modify (name varchar2(15));

Table altered.

SQL> alter table bab add(mat number(3));

Table altered.

SQL> desc bab;
 Name                                      Null?    Type
```

NAME	VARCHAR2(15)
ENG	NUMBER(3)
TAM	NUMBER(3)
MAT	NUMBER(3)

TRUNCATE A TABLE

Description

The truncate command deletes all the records from the table.

Syntax

Truncate table <table name>;

Example

SQL> truncate table bab;

Table truncated.

SQL> truncate table bab reuse storage;

{Oracle internally reclaims the space of the deleted rows for future use.}

Table truncated.

TO DROP A TABLE

Description

To delete a table.

Syntax

Drop table <table name>;

Example

SQL> drop table bab;

Table dropped.

INSERT COMMAND

Syntax

Insert into <table name> values (a list of data values);

```
SQL> insert into bab values('babu',67,89,90);

1 row created.

SQL> insert into bab values('&name',&english,&tamil,&maths);
Enter value for name: ddl
Enter value for english: 67
Enter value for tamil: 45
Enter value for maths: 67
old   1: insert into bab values('&name',&english,&tamil,&maths)
new   1: insert into bab values('ddl',67,45,67)

1 row created.

SQL> insert into bab(name,eng) values('jack',100);

1 row created.

SQL> insert into bab values('raj',null,90,null);

1 row created.

SQL> select * from bab;

NAME                 ENG       TAM       MAT
---------------- ---------- ---------- ----------
babu                  67        89        90
ddl                   67        45        67
sandhya               45        67        89
mani                  67        89        56
kavi                  56        78        90
jack                  100
raj                   90

7 rows selected.

SQL> alter table bab add(odate date);
Table altered.

SQL> desc bab;
```

```
Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 NAME                                       VARCHAR2(15)
 ENG                                       NUMBER(3)
 TAM                                       NUMBER(3)
 MAT                                       NUMBER(3)
 ODATE                                      DATE
```

INSERT DATE VALUES

```
SQL> insert into bab values ('ram',45,56,78,'10-jan-98');

1 row created.
```

SELECT COMMAND

Syntax

Select column_name ..... from <tale name>;

Example

```
SQL> select name,eng from bab;

NAME                ENG
--------------- ----------
babu                 67
ddl                 67
sandhya               45
mani                 67
kavi                56
jack                100
raj
ram                  45

8 rows selected.
```

SELECTING DISTINCT ROWS

Example

```
SQL> select distinct eng from bab;

 ENG
```

45
56
67
100

SELECT WITH WHERE CLAUSE

Syntax

Select columns.....from table where condition [order by];

Example

SQL> select * from bab where eng = 67 order by name;

NAME	ENG	TAM	MAT	ODATE
babu	67	89	90	
ddl	67	45	67	
mani	67	89	56	

DISPLAY VALUES ONLY MORE THAN 1

Example

SQL> select * from bab;

NAME	ENG	TAM	MAT	ODATE
babu	200	100	90	12-MAR-98
ddl	67	45	67	12-MAR-99
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96
kavi	56	78	90	12-MAR-98
ram	45	56	89	10-JAN-98

6 rows selected.

SQL>select name ,mat from bab where mat in(select mat from bab
group by mat having count(mat) > 1);

```
NAME                MAT
--------------- ----------
sandhya              89
ram                 89
babu                 90
kavi                90

4 rows selected.
```

DISPLAY SAME MATHS MARK MORE THAN 1

Example

```
SQL> select mat from bab group by mat having count(*) > 1;

MAT
----------
      89
      90
2 rows selected.
```

DISPLAYING HIGHEST MARK(ENG) IN TABLE BAB.

Example1

```
SQL>select * from bab where eng=(select
 (eng) from bab);

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
```

Example2

```
SQL>select * from bab where mat>(select avg(tam) from bab);

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
sandhya              45        67        89 12-MAY-99
kavi                56        78        90 12-MAR-98
```

DISPLAYING HIGHEST MARK (ENG) TO NO. OF PERSONS.

Example

```
SQL> select name,eng from bab a where &no >(select count(distinct eng) from
    bab  where eng>=a.eng);
```

```
SQL> /
Enter value for no: 3

NAME                ENG
--------------- ----------
babu                200
ddl                 67
mani                67

3 rows selected.
```

DISPLAY VALUES OF SUM(MAT) GREATER THAN MAX(TAM)

Example

```
SQL> select sum(mat) from bab having sum(mat)>(select max(tam)
    from bab) group by mat;
```

```
 SUM(MAT)
----------
       178
       180

2 rows selected.
```

UPDATE COMMAND

Syntax

Update <table name> set field = value,.....where condition;

Example
SQL> update bab set eng = 200 where name = 'babu';

1 row updated.

DELETE COMMAND

Syntax

Delete from <table name> where conditions;

Example1

SQL> delete from bab where eng = 100;

1 row deleted.

DELETE DUPLICATE ROWS

Example2

SQL> delete from bab a where rowid > (select min(rowid) from bab b where a.mat = b.mat);

2 rows deleted.

SQL> select * from bab;

NAME	ENG	TAM	MAT	ODATE
babu	200	100	90	12-MAR-98
ddl	67	45	67	12-MAR-99
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96

4 rows selected.

COMMIT

Description

Changes can made permanent to the database.

Syntax

Commit;

Commit complete.

SAVEPOINT
Description
Savepoints are like markers to divided a very lengthy transaction to smaller ones.
Syntax
savepoint savepoint_id;

Example
SQL> select * from bab;

NAME	ENG	TAM	MAT	ODATE
babu	200	89	90	12-MAR-98
ddl	67	45	67	12-MAR-99
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96
kavi	56	78	90	12-MAR-98
ram	45	56	78	10-JAN-98

6 rows selected.

SQL> update bab set tam = 100 where eng = 200;
1 row updated.

SQL> select * from bab;

NAME	ENG	TAM	MAT	ODATE
babu	200	100	90	12-MAR-98
ddl	67	45	67	12-MAR-99
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96
kavi	56	78	90	12-MAR-98
ram	45	56	78	10-JAN-98

6 rows selected.

```
SQL> savepoint u;

Savepoint created.

SQL>  delete from bab where tam = 78;

1 row deleted.

SQL> update bab set eng = 300 where name = 'ram';

1 row updated.

SQL> select * from bab;

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
ddl                67        45        67 12-MAR-99
sandhya              45        67        89 12-MAY-99
mani                67        89        56 12-AUG-96
ram                300        56        78 10-JAN-98

SQL> rollback to savepoint u;

Rollback complete.

SQL> select * from bab;

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
ddl                67        45        67 12-MAR-99
sandhya              45        67        89 12-MAY-99
mani                67        89        56 12-AUG-96
kavi                56        78        90 12-MAR-98
ram                 45        56        78 10-JAN-98

6 rows selected.

SQL> rollback;
```

Rollback complete.

SQL> select * from bab;

NAME	ENG	TAM	MAT	ODATE
babu	200	89	90	12-MAR-98
ddl	67	45	67	12-MAR-99
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96
kavi	56	78	90	12-MAR-98
ram	45	56	78	10-JAN-98

6 rows selected.

GRANT PRIVILEGE COMMAND

Description

User wants to share an object with other, the appropriate privileges can be granted on that particular object to others.

Syntax

Grant privileges on <object-name> to <username>;

Example

SQL> grant select,update on bab to system;

Grant succeeded.

SQL> grant update(eng,tam) on stud to system;

Grant succeeded.

SQL>grant all on bab to system;

Grant succeeded.

REVOKE PRIVILEGE COMMAND

Description

To withdraw the privilege that has been granted to a user.

Syntax

Revoke privileges on <object-name> from <username>.

Example

SQL> revoke select,update on bab from system;

Revoke succeeded.

SQL>revoke all on bab from system;

Revoke succeeded.

SELECT STATEMENT TO CREATE A TABLE

Description

Create a table and copy the records into it with a single statement.

Syntax

Create table <new_table_name> as select column_name from <existing_table_name>.

SQL> create table frook as select * from bab;

Table created.

SQL> select * from frook;

NAME	ENG	TAM	MAT	ODATE
babu	200	89	90	12-MAR-98
ddl	67	45	67	12-MAR-99
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96
kavi	56	78	90	12-MAR-98
ram	45	56	78	10-JAN-98

6 rows selected.

Example2
SQL> create table rajesh(stuname,english) as select name,eng from bab;

Table created.

SQL> select * from rajesh;

```
STUNAME         ENGLISH
--------------- ----------
babu                 200
ddl                  67
sandhya               45
mani                 67
kavi                 56
ram                  45
```

6 rows selected.

SQL> desc rajesh;

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 STUNAME                                            VARCHAR2(15)
 ENGLISH                                            NUMBER(3)
```

Example3
SQL> create table siva as select * from bab where eng = 67;

Table created.

SQL> select * from siva;

```
NAME                 ENG        TAM        MAT ODATE
--------------- ---------- ---------- ---------- ---------
ddl                  67         45         67 12-MAR-99
mani                 67         89         56 12-AUG-96
```

Example4
Note

(Copy table structure only.)

SQL> create table ram as select * from bab where 1=2;

Table created.

SQL> select * from ram;

no rows selected

SQL> desc ram;

Name	Null?	Type
NAME		VARCHAR2(15)
ENG		NUMBER(3)
TAM		NUMBER(3)
MAT		NUMBER(3)
ODATE		DATE

SELECT COMMAND TO INSERT RECORDS

Description

Inserting records from one table into another table. The structure of the two tables must be same.

Syntax

Insert into <table_table> (select column_names from <existing_table_name>);

Example

SQL> create table jack(name varchar2(10),eng number(3),tam number(3),mat number(3),odate date);

Table created.

SQL> insert into jack (select * from bab);

6 rows created.

SQL> select * from jack;

```
NAME            ENG       TAM       MAT ODATE
---------- ---------- ---------- ---------- ---------
babu             200        89        90 12-MAR-98
ddl               67        45        67 12-MAR-99
sandhya           45        67        89 12-MAY-99
mani              67        89        56 12-AUG-96
kavi              56        78        90 12-MAR-98
ram               45        56        78 10-JAN-98

6 rows selected.
```

SELECT COMMAND FOR COLUMN ALIASES

Description

Used to temporarily change a column name.

Syntax

Select column_name <alias_name> from table_name;

Example

```
SQL> select name student_name,eng english_mark from bab;

STUDENT_NAME    ENGLISH_MARK
--------------- ------------
babu                 200
ddl                  67
sandhya                45
mani                 67
kavi                 56
ram                  45

6 rows selected.
```

ARITHMETIC OPERATORS

Description

The arithmetic operators are addition(+),subtraction(-),multiplication(*),and division(/).

Example1
SQL> select name,eng+tam total_et from bab where mat = 90;

```
NAME            TOTAL_ET
--------------- ----------
babu                 289
kavi                 134
```

Example2
Note: {value+null=null)
SQL> select * from second;

```
   ROLLNO       TAM       ENG
---------- ---------- ----------
      100        65        78
      101        63
```

SQL> select rollno,tam+eng totalet from second;

```
   ROLLNO    TOTALET
---------- ----------
      100       143
      101
```

Example3
SQL> select name,eng*(tam-mat) total from bab where name = 'ram';

```
NAME              TOTAL
--------------- ----------
ram                -990
```

Example4
SQL> select * from bab where eng>mat*2;

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200        89        90 12-MAR-98
```

COMPARISON OPERATORS

Description

The comparison operators are =,!=,<,>,<=,>=,
BETWEEN(to check between any two values),
IN(to match with any of the values in the list),
LIKE(to match a character pattern),
IS NULL(to check whether it is null).
NOT BETWEEN and NOT LIKE.

Example1

```
SQL> select * from bab where eng>mat/2;

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200        89        90 12-MAR-98
ddl                 67        45        67 12-MAR-99
sandhya             45        67        89 12-MAY-99
mani                67        89        56 12-AUG-96
kavi                56        78        90 12-MAR-98
ram                 45        56        78 10-JAN-98

6 rows selected.
```

Example2

```
SQL> select name from bab where eng in(67,89,200,48);

NAME
---------------
babu
ddl
mani
```

Example3

```
SQL> select * from bab where not (eng = 67 or eng = 45);

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200        89        90 12-MAR-98
```

C.Sivanantham

```
kavi                 56        78        90 12-MAR-98
```

Example4

```
SQL> select * from bab where eng > 100;

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200        89        90 12-MAR-98
```

Example5

```
SQL> select name from bab where eng between 44 and 60;

NAME
---------------
sandhya
kavi
ram
```

Example6

```
SQL> select eng from bab where name like 'b%';

     ENG
----------
     200
```

Example7

```
SQL> select * from bab where name like 'k_v%';

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
kavi                56        78        90 12-MAR-98
```

LOGICAL OPERATORS

Description

The logical operator are AND,NOT and OR.

Example

```
SQL> select * from bab where eng = 67 and mat = 67;

NAME                ENG       TAM       MAT ODATE
```

--- --- --- --- ---
ddl 67 45 67 12-MAR-99

DATE FUNCTIONS

ADD_MONTHS

Syntax

add_months(d,n),where d is the date and n is the number of months.

Example1

```
SQL> select odate,add_months(odate,2) from bab;

ODATE     ADD_MONTH
--------- ---------
12-MAR-98 12-MAY-98
12-MAR-99 12-MAY-99
12-MAY-99 12-JUL-99
12-AUG-96 12-OCT-96
12-MAR-98 12-MAY-98
10-JAN-98 10-MAR-98

6 rows selected.
```

Example2

```
SQL> select sysdate, add_months(sysdate,-5) from dual;

SYSDATE   ADD_MONTH
--------- ---------
05-FEB-06 05-SEP-05

SQL> select sysdate,sysdate+5 from dual;

SYSDATE   SYSDATE+5
--------- ---------
05-FEB-06 10-FEB-06

SQL> select sysdate,sysdate-4 from dual;

SYSDATE   SYSDATE-4
```

--------- ---------
05-FEB-06 01-FEB-06

LAST_DAY
Description
Returns the last day of the month.
Syntax
last_day(d)

Example
SQL> select sysdate,last_day(odate) from bab where eng > 65;

SYSDATE LAST_DAY(
--------- ---------
26-JAN-06 31-MAR-98
26-JAN-06 31-MAR-99
26-JAN-06 31-AUG-96

MONTH_BETWEEN
Description
To find out the number of months between two dates.
Syntax
months_between(d1,d2) where d1,d2 are dates.

Example1
SQL> select months_between(sysdate,odate) from bab;

MONTHS_BETWEEN(SYSDATE,ODATE)

94.4794015
82.4794015
80.4794015
113.479402
94.4794015
96.5439176

6 rows selected.

Example2

```
SQL> select sysdate, add_months(sysdate,-5) from dual;

SYSDATE   ADD_MONTH
--------- ---------
05-FEB-06 05-SEP-05

SQL> select months_between('12-aug-02','12-jan-02') from dual;

MONTHS_BETWEEN('12-AUG-02','12-JAN-02')
---------------------------------------
                                      7
SQL> select months_between('12-aug-02','18-july-02') from dual;

MONTHS_BETWEEN('12-AUG-02','18-JULY-02')
----------------------------------------
                             .806451613

SQL> select months_between('12-aug-02','10-july-02') from dual;

MONTHS_BETWEEN('12-AUG-02','10-JULY-02')
----------------------------------------
                            1.06451613
```

FOLLOWING QUERY IS CALCULATED DAYS,MONTHS,YEARS

```
SQL>select (to_date('31-jan-06')-to_date('30-nov-05'))
Days, floor(MONTHS_BETWEEN('31-jan-06','30-nov-05'))
months, floor(months_between('31-jan-06','30-nov-
05')/12) years from dual;

DAYS     MONTHS      YEARS
---------- ---------- ----------
      62         2         0
```

FOLLOWING QUERY IS CALCULATED THE MONTH FOR 30 & 31 DAYS.

Example

```
SQL> select (to_date('31-jan-06')-to_date('30-nov-05')) days,
        floor((to_date('31-jan-06')-to_date('30-nov-05'))/30) months,
        floor((to_date('31-jan-06')-to_date('30-nov-05'))/365) years
        from dual;

DAYS          MONTHS          YEARS
---------- ---------- ----------
        62          2          0
```

```
SQL>  select (to_date('26-jan-06')-to_date('15-oct-03')) days,
        floor((to_date('26-jan-06')-to_date('15-oct-03'))/31) months,
       floor((to_date('26-jan-06')-to_date('15-oct-03'))/365) years
       from dual;

 DAYS         MONTHS     YEARS
---------- ---------- ----------
  834           26          2
```

ROUND

Description

Returns the date which is rounded(nearest year) to the unit specified by the format model.

Syntax

round(d,[fmt]) where d is date and fmt is format model.

Example1

```
SQL> select odate,round(odate,'year') from bab where eng > 64;

ODATE     ROUND(ODA
--------- ---------
12-MAR-98 01-JAN-98
12-MAR-99 01-JAN-99
```

C.Sivanantham

12-AUG-96 01-JAN-97

Example2
SQL> select odate, round(odate,'month') from bab where eng > 64;

ODATE ROUND(ODA
--------- ---------
12-MAR-98 01-MAR-98
12-MAR-99 01-MAR-99
12-AUG-96 01-AUG-96

Example3 (day to be rounded to the nearest Sunday)
SQL> select odate,round(odate,'day') from bab where eng > 64;

ODATE ROUND(ODA
--------- ---------
12-MAR-98 15-MAR-98
12-MAR-99 14-MAR-99
12-AUG-96 11-AUG-96

Example4 (it is rounded to the day)
SQL> select odate,round(odate) from bab where eng > 64;

ODATE ROUND(ODA
--------- ---------
12-MAR-98 12-MAR-98
12-MAR-99 12-MAR-99
12-AUG-96 12-AUG-96

NEXT_DAY (display next Tuesday date)
Syntax
next_day(d,day) where d is date and day implies any weekday.
Example
SQL> select sysdate,next_day (sysdate,'tuesday') from dual;

SYSDATE NEXT_DAY(
--------- ---------

26-JAN-06 31-JAN-06

TRUNCATE
Syntax
trunc(d,[fmt]) fmt is neglected, then date is converted to the nearest day.

Example
SYSDATE TRUNC(SYS
--------- ---------
26-JAN-06 01-JAN-06

SQL> select sysdate,trunc(sysdate,'month') from dual;

SYSDATE TRUNC(SYS
--------- ---------
26-JAN-06 01-JAN-06

SQL> select sysdate,trunc(sysdate,'day') from dual;

SYSDATE TRUNC(SYS
--------- ---------
26-JAN-06 22-JAN-06 (display to the nearest Sunday)

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

SYSDATE TRUNC(SYS
--------- ---------
26-JAN-06 26-JAN-06 (it is rounded to the nearest day i.e. the sysdate)

GREATEST & LEAST
Description
Returns the latest date present in the argument.
Syntax
greatest(d1,d2....).where d1 and d2 are dates.

Example
SQL> select odate,sysdate,greatest(odate,sysdate) from bab where eng = 200;

ODATE SYSDATE GREATEST(
--------- --------- ---------

12-MAR-98 26-JAN-06 26-JAN-06
Example2
SQL> select greatest(10,70,32,12) from dual;

```
GREATEST(10,70,32,12)
---------------------
                   70
```

SQL> select least(10,70,32,12) from dual;

```
LEAST(10,70,32,12)
------------------
                10
```

SQL> select greatest('a','j','z') from dual;

```
G
-
z
```

SQL> select least('babu','ddl','sandhya') from dual;

```
LEAS
----
babu
```

NEW_TIME
Description
Returns the time and date of a date column or literal date in other time zones.
Syntax
new_time(date,’this’,’other’);

Example
SQL> select new_time('13-feb-99','est','yst') from dual;

```
NEW_TIME(
---------
12-FEB-99 (which is the date in the time sone ‘yst’)
```

CHARACTER FUNCTIONS

Description

Character functions accept character input and return either character or number values.

FUNCTION	INPUT	OUTPUT
Initcap(char)	Select initcap('hello') from dual;	Hello
Lower(char)	Select lower('FUN') from dual;	fun
Upper(char)	Select upper('sun') from dual;	SUN
Ltrim(char,set)	select ltrim('xyzadams','xyz') from dual;	adams
Rtrim(char,set)	select rtrim('xyzadams','ams') from dual;	xyzad
Translate(char, from, to)	select translate('jack','j','b') from dual;	back
Replace(char, searchstring, [rep string])	select replace('jack and jue','j','bl') from dual;	black and blue
(char,m,n)	select substr('abcdefg',3,2) from dual;	cd

NOTE

Soundex,which is also a character function compares words that are spelled differently,but sound alike.

Example

```
SQL> select * from bab where soundex(name)=soundex('baabu');

NAME                 ENG       TAM       MAT ODATE
-------------- ---------- ---------- ---------- ---------
babu                 200        89        90 12-MAR-98
```

CHARACTER FUNCTION

Example

```
SQL> select chr(112) from dual;

C
-
p
```

LPAD and RPAD and LENGTH

Example

```
SQL> select lpad('babu',10,'=') from dual;

LPAD('BABU
----------
======babu

SQL> select rpad('babu',10,'=') from dual;

RPAD('BABU
----------
babu======

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

LENGTH('BABU')
--------------
             4
```

DECODE

Description

DECODE function does a value by value replacement.

Syntax

select decode(<value,if1,then1,if2,then2,...>) from <table_name>;

Example1

```
SQL> select eng,decode(name,'babu','baabu') changename,tam from bab where
eng = 200;

       ENG CHANG       TAM
---------- ----- ----------
       200 baabu        89
```

Example2

```
SQL>
  1  select distinct
  2  DECODE(name,'babu','mr.k.babu','ddl','dhanlakshmi',name)
```

```
  3* from bab
  4  /

DECODE(NAME,'BA
---------------
dhanlakshmi
kavi
mani
mmm
mr.k.babu
sandhya

6 rows selected.
```

Example3

```
SQL>
  1  select distinct
  2  DECODE(name,'babu','mr.k.babu','ddl','dhanlakshmi')
  3* from bab
  4  /

DECODE(NAME
-----------
dhanlakshmi
mr.k.babu
```

CONCATENATION (||) OPERATOR

Description

Used to merge two or more strings.

Example

```
SQL> select ('The name is '||name||'   english mark is '||eng||'   maths mark is
'||mat) Record from bab where tam = 89;

RECORD
--------------------------------------------------------------------------------
The name is babu   english mark is 200   maths mark is 90
The name is mani   english mark is 67   maths mark is 56
```

NUMERIC FUNCTIONS

FUNCTION	INPUT	OUTPUT
Abs	Select abs(-15) from dual	15
Celi(n)	select ceil(44.778) from dual	45
Cos(n)	Select cos(180) from dual	-.5984601
Cosh(n)	Select cosh(0) from dual	1
Exp(n)	select exp(4) from dual	54.59815
Floor(n)	select floor(100.2) from dual	100
Power(m,n)	select power(4,2) from dual	16
Mod(m,n)	select mod(10,3) from dual	1
Round(m,n)	select round(100.256,2) from dual	100.26
Trunc(m,n)	select trunc(100.256,2) from dual	100.25
Sqrt(n)	Select sqrt(25) from dual	5

Example

```
SQL> select ln(2) from dual;
    LN(2)
----------
.693147181
SQL> select mod(45,6) from dual;
 MOD(45,6)
----------
         3
SQL> select sign(-32) from dual;
 SIGN(-32)
----------
        -1
SQL> select sign(-65) from dual;
```

CONVERSION FUNCTIONS

Description

Conversion functions convert a value from one data type to another.

Default date format is dd-mon-yy.

dd------16

dy------sat

day-----Saturday

mm------03

mon-----mar

month---march

fmmonth-march(fullmonth)

yy------02

yyy-----002

yyyy----2002

ddth----16th or 3rd or 2nd or 1st

Syntax

1. To_char()
2. To_date()
3. To_number()

Example

```
SQL> select to_char(sysdate,'ddth "of" fmmonth yyyy') from dual;

TO_CHAR(SYSDATE,'DDTH"
----------------------
27th of january 2006

SQL> select to_date('january 27 1999','month-dd-yyyy') from dual;

TO_DATE('
---------
27-JAN-99

SQL> select round(to_date('27-jan-99'),'year') from dual;

ROUND(TO_
---------
01-JAN-99
```

```
SQL> select to_number('100') from dual;

TO_NUMBER('100')
----------------
             100
```

MISCELLANEOUS FUNCTIONS

UID

Description

The function returns the integer value corresponding to the user currently logged in.

Example

```
SQL> select uid from dual;

       UID
----------
        66
```

USER

Description

Returns the login's user name, which is in varchar2 datatype.

Example

```
SQL> select user from dual;

USER
------------------------------
SCOTT
```

NULL VALUE (NVL)

Description

The null value function is used in case where we want to consider null values as zero.

Syntax (null values and zeroes are not equivalent)

nvl(exp1,exp2) if exp1 is null, nvl will return exp2.

Example1

```
SQL> select * from first;

   ROLLNO       ENG NAM
---------- ---------- ----------
      100        500 babu
      101           ddl

SQL> select nam,nvl(eng,100) from first;

NAM        NVL(ENG,100)
---------- ------------
babu               500
ddl                100

SQL> select * from first;
   ROLLNO       ENG NAM
---------- ---------- ----------
      100     500   babu
     101              ddl
```

Example2

```
SQL> select * from first;

   ROLLNO       ENG NAM
---------- ---------- ----------
      100        500 babu
      101           ddl
      105           san

SQL> select nam,nvl(eng,100) from first;

         NAM       NVL(ENG,100)
    ---------- ------------
         babu         500
         ddl          100
         san          100

SQL> select * from first;
```

ROLLNO	ENG	NAM
100	500	babu
101		ddl
105		sans

VSIZE
Description
It returns the number of bytes in the expression.
Syntax
vsize(expr).

Example
SQL> select vsize('babu') from dual;

VSIZE('BABU')

4

GROUP FUNCTIONS

AVG FUNCTION
Example
SQL> select avg(eng) from bab where tam = 89;

AVG(ENG)

133.5

MIN FUNCTION
Example
SQL> select min(eng) from bab where tam = 89;

MIN(ENG)

67

MAX FUNCTION

Example
SQL> select max(eng) from bab where tam = 89;

```
  MAX(ENG)
----------
       200
```

SUM FUNCTION
Example
SQL> select sum(eng+tam) totalet from bab;

```
   TOTALET
----------
       904
```

COUNT FUNCTION
Example1
SQL> select count(*) from bab;

```
  COUNT(*)
---------- 6
```

Example2
SQL> update bab set mat = null where tam = 78;

1 row updated.

SQL> select * from bab;

NAME	ENG	TAM	MAT	ODATE
babu	200	89	90	12-MAR-98
ddl	67	45	67	12-MAR-99
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96
kavi	56	78		12-MAR-98
ram	45	56	78	10-JAN-98

6 rows selected.

SQL> select count(mat) from bab;

```
COUNT(MAT)
----------
         5

SQL> select count(distinct tam) from bab;

COUNT(DISTINCTTAM)
------------------
                 5
```

GROUP BY CLAUSE

Example

```
SQL> select eng,tam from bab;

       ENG        TAM
---------- ----------
       200         89
        67         56
        45         67
        67         56
        56         78
        45         56

6 rows selected.

SQL> select tam,max(eng) from bab group by tam;

       TAM   MAX(ENG)
---------- ----------
        56         67
        67         45
        78         56
        89        200

SQL> select tam,sum(eng) from bab group by tam;

       TAM   SUM(ENG)
---------- ----------
        56        179
```

67 45
78 56
89 200

HAVING CLAUSE
Example
SQL> select tam,max(eng) from bab group by tam having tam not in (77.45.56);

TAM MAX(ENG)
--- ---
67 45
78 56
89 200

STDDEV
Description
Gives the standard deviation of a norm of values.

Example
SQL> select tam,stddev(eng) from bab group by tam;

TAM STDDEV(ENG)
--- ---
56 12.7017059
67 0
78 0
89 0

VARIANCE
Description
Gives the variance of a norm of values.

Example
SQL> select tam,variance(eng) from bab group by tam;

TAM VARIANCE(ENG)
--- ---
56 161.333333
67 0
78 0

89 0

SET OPERATORS

SQL> select * from bab;

NAME	ENG	TAM	MAT	ODATE
babu	200	89	90	12-MAR-98
ddl	67	45	67	12-MAR-99
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96
kavi	56	78	90	12-MAR-98
ram	45	56	78	10-JAN-98

6 rows selected.

SQL> select * from stud;

ROLLNO	NAME	ENG	TAM	MAT
500	babu	250	278	300
501	dhana	200	89	90
503	ddl	200	89	90
504	sandhya	67	89	45
569	aaaa	67	34	100
100	bbbbb	89	45	65

6 rows selected.

UNION

Description

Returns all distinct rows selected by both queries.

Example

SQL> select name from bab union select name from stud;

NAME

---

aaaa

babu
bbbbb
ddl
dhana
kavi
mani
ram
sandhya

9 rows selected.

UNION ALL
Description
Returns all rows selected by either query including duplicates.

Example
SQL> select name from bab union all select name from stud;

NAME
---
babu
ddl
sandhya
mani
kavi
ram
babu
dhana
ddl
sandhya
aaaa
bbbbb

12 rows selected.

INTERSECT
Description
Returns only rows that are common to both the queries.

Example
SQL> select name from bab intersect select name from stud;

NAME
---
babu
ddl
sandhya

MINUS
Description
Returns all distinct rows selected only by the first query and not by the second.

Example
SQL> select name from bab minus select name from stud;

NAME
---
kavi
mani
ram

SIMPLE JOIN

EQUI JOIN
Description
A join, which is based on equalities, is called an equi-join.

Example
SQL> select rollno,bab.name,odate from bab,stud where bab.name=stud.name;

ROLLNO	NAME	ODATE
500	babu	12-MAR-98
503	ddl	12-MAR-99
504	sandhya	12-MAY-99

NON EQUI JOIN

Description

Specifies the relationship between columns belonging to different tables by making use of the relational operators (<, <=,>,>=, =).

Example

SQL> select rollno,odate,bab.mat from bab,stud where ((stud.tam>bab.mat) and (bab.name=stud.name));

```
   ROLLNO   ODATE          MAT
   -------  ---------    ----------
    500     12-MAR-98       90
    503     12-MAR-99       67
```

SELF JOIN

Description

Joining of a table itself is known as self join.

Example

SQL> select * from bab;

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200        89        90 12-MAR-98
ddl                 67         45        67 12-MAR-99
sandhya             45         67        89 12-MAY-99
mani                67         89        56 12-AUG-96
kavi                56         78        90 12-MAR-98
ram                 45         56        78 10-JAN-98
```

6 rows selected.

SQL> select a.eng,b.tam,a.mat from bab a,bab b where a.eng = b.tam;

```
     ENG       TAM       MAT
---------- ---------- ----------
      45        45        89
```

45	45	78
56	56	90
67	67	67
67	67	56

OUTER JOIN

Description

Returns those rows from one table that do not match any row from the other table.

Example

```
SQL> select a.eng,rollno,a.name from bab a, stud where a.name = stud.name(+);
```

ENG	ROLLNO	NAME
200	500	babu
67	503	ddl
56		kavi
67		mani
45		ram
45	504	sandhya

6 rows selected.

SUBQUERIES

Description

Nesting of queries, one within the other, is terned as a subquery.

Example

```
SQL>  select * from bab where name=(select name from stud where eng =250);
```

NAME	ENG	TAM	MAT	ODATE
babu	200	89	90	12-MAR-98

SUBQUERIES RETURN SEVERAL VALUES

Description

Subqueries can also return more than one value. we use ANY,ALL,IN or NOT IN.

Example1

SQL> select * from stud where eng < any(select tam from bab where mat between 10 and 70);

ROLLNO	NAME	ENG	TAM	MAT
504	sandhya	67	89	45
569	aaaa	67	34	100

Example2

SQL>select * from stud where eng > all(select tam from bab where mat < 100);

ROLLNO	NAME	ENG	TAM	MAT
500	babu	250	278	300
501	dhana	200	89	90
503	ddl	200	89	90

MULTIPLE SUBQUERIES

Example1

SQL> select name,eng,odate from bab where name=(select name from stud where rollno=(select rollno from student where mat = 300));

NAME	ENG	ODATE
babu	200	12-MAR-98

CORRELATED SUBQUERIES

Example

SQL> select distinct(a.eng) from bab a where 3 >= (select count(name) from bab where a.mat = mat);

ENG
45
56

INTEGRITY CONSTRAINTS

1. Domain integrity constraints
   - (i) Not null constraint
   - (ii) Check constraint

2. Entity integrity constraints
   - (iii) Unique constraints
   - (iv) Primary Key constraints

3. Referential integrity constraints
   - (v) Foreign Key constraints

NOT NULL CONSTRAINTS

Description

Zero and Null are not equivalent. One null is not equivalent to another null.

Example

```
SQL> create table first (a number(3),b number(3) constraint b1 not null,
c number(3) not null);

Table created.

SQL> alter table first modify a not null;

Table altered.
```

Example2

```
SQL> create table second(a number(3),b number(3));

Table created.

SQL> insert into second values(101,null);

1 row created.
```

SQL> alter table second modify b not null;

ERROR at line 1:
ORA-02296: cannot enable (SCOTT.) - null values found

CHECK CONSTRAINTS

Example1
SQL>create table third (a number(3),b number(3) constraint bc check(b<50));

Table created.

SQL> insert into third values(78,56);
insert into third values(78,56)
*
ERROR at line 1:
ORA-02290: check constraint (SCOTT.BC) violated

Example2
SQL> alter table third add constraint ac check(a in(45,65));

Table altered.

SQL> insert into third values(48,33);

ERROR at line 1:
ORA-02290: check constraint (SCOTT.AC) violated

UNIQUE CONSTRAINTS

Example1
SQL> create table four(a number(3),b number(3) constraint bc1
check(b<90),constraint bu unique(b));

Table created.

Example2
SQL> alter table four add constraint bc2 unique(a);

Table altered.

PIMARY KEY CONSTRAINT

Description

Primary key does not allow null values. A table can have only one primary key. Primary key cannot be defined in an alter table command when the table contains rows having null values.

Example1

SQL> create table five(a number(3) constraint ap primary key,b number(3),c number(3) not null);

Table created.

Example2

SQL> create table six(a number(3),b number(3), constraint ap1 primary key(a));

Table created.

Example3

SQL> alter table first add constraint ap2 primary key(b);

Table altered.

REFERENTIAL INTEGRITY CONSTRAINTS

Description

The column in the parent table as a primary key and the same column in the child table as a foreign key referring to the corresponding parent entry.

Example1

SQL> create table seven(a number(3) constraint af references five(a), b number(3) not null);

(or)

SQL> create table seven(a number(3), foreign key(a) references five(a), b number(3) not null);

Table created.

Example2
SQL> alter table four add constraint af1 foreign key(a) references six(a);

Table altered.

Example3
SQL> delete from five where a=100;

ERROR at line 1:
ORA-02292: integrity constraint (SCOTT.AF) violated - child record found
NOTE
First delete child record and then delete parent record.

ON DELETE CASCADE CLAUSE
Description
Use on delete cascade clause to delete parent record automatically removes child record.

Example
SQL> create table nine(a number(3),b number(3), constraint babu primary key(a));

Table created.

SQL> create table ten(a number(3),c number(3));

Table created.

SQL> alter table ten add constraint babu1 foreign key(a) references nine(a) on delete cascade;

Table altered.
SQL> insert into nine values(1,78);

1 row created.

SQL> insert into ten values(1,67);

1 row created.

SQL> delete from ten where a=1;

1 row deleted.

DEFERRABLE CONSTRAINT
Description

Three conditions
1. Deferrable initially immediate=constraint violation at the time of insert.
2. Deferrable initially deferred=constraint violation at the time of commit.

Syntax1 :

Alter table <table name> add constraint <constraint_name> foreign key <column_name> references <table name> deferrable initially deferred;

Syntax2(Enables all constraint)
Set constraint all immediate;

Syntax3(Diable all constraint)

Set constraint all deferred;

Syntax4
Alter table <table_name> drop constraint constraint_name;

Example
SQL> alter table four drop constraint bu;

Table altered.

TYPES OF LOCKS

1. Row Level Locks
2. Table Level Locks

ROW LEVEL LOCKS

Description

This command used to lock the rows that would be updated later. Other users cannot update that particular row till the lock is released.

Syntax

Select ..for update clause

Example

```
SQL> select * from bab where name = 'ram' for update of eng,tam;

NAME                 ENG       TAM       MAT ODATE
---------------- ---------- ---------- ---------- ---------
ram                  45        56        78 10-JAN-98

SQL> update bab set eng = 75 where name = 'ram';

1 row updated.
```

TABLE LEVEL LOCKS

1. Share lock
2. Share update lock
3. Exclusive lock

Syntax

Lock table <table_name> in <share or share update or exclusive mode>;

SHARE LOCK

Description

The table allowing other users to only query but not insert,update or delete rows in a table.

Example

```
SQL> lock table bab in share mode;

Table(s) Locked.
```

SHARE UPDATE LOCK

Description

It locks rows that are to be updated in a table. It permits other users to concurrently query, insert, update or even lock other rows in the same table.

Example

SQL> lock table bab in share update mode;

Table(s) Locked.

EXCLUSIVE LOCK

Description

It allows the other user to only query but not insert, delete or update rows in a table. It is similar to a share lock but one user can place an exclusive lock on a table at a time.

Example

SQL> lock table bab in exclusive mode;

Table(s) Locked.

RELEASE TABLE LOCK

Description

Locks can be released by issuing either rollback or commit.

Example

SQL> rollback;

Rollback complete.

NOWAIT

Description

If another user tries to violate the above restrictions by trying to lock the table, then he will be made to wait indefinitely. This delay could be avoided by appending a 'nowait' clause in the lock table command.

Example

SQL> lock table bab in exclusive mode nowait;

Table(s) Locked.

DEADLOCK

Description

A deadlock occurs when two user have a lock, each on a separate object, the first person is wait for second person to release lock, but second person is not release, at that we use deadlock.

User performing the following task	Locks mode permitted by another user		
	share	share update	exclusive
	yes	no	no
	no	yes	no
	no	no	no
	yes	yes	yes

TABLE PARTITIONS

Description

A single logical table can be split into a number of physically separate pieces based on ranges of key values. Each of the parts of the table is called a partition.

Syntax

Create table <table_name> (column name data type, column name date type,..) partition by range(column name) (partition <partition name> values less than <value>, partition <partition name> values less than <value>);

Example

SQL> create table mani(eng number(3),tam number(3),mat number(3)) partition by range(eng) (partition m1 values less than (40),partition m2 values less than (90));

Table created.

SQL> select * from mani;

```
     ENG       TAM       MAT
---------- ---------- ----------
      34        56        78
      12        76        54
      78        56        89
      40        65        43
```

SQL> select * from mani partition(m1);

```
     ENG       TAM       MAT
---------- ---------- ----------
      34        56        78
      12        76        54
```

SQL> select * from mani partition(m2);

```
     ENG       TAM       MAT
---------- ---------- ----------
      78        56        89
      40        65        43
```

SQL> insert into mani values(99,56,45);
insert into mani values(99,56,45)
*
ERROR at line 1:
ORA-14400: inserted partition key does not map to any partition

RENAMING PARTITIONS

Syntax

Alter table <tablename> Rename partition <old partition name> to <new partition name>;

Example

Alter table mani rename partition m2 to m100;

MOVING PARTITIONS

Description

Move a partition from a most active tablespace to a different tablespace in order to balace I/O operations. Oracle creates the table in the default 'system' tablespace.
Example
SQL> alter table mani move partition m1 tablespace system;

Table altered.

ADDING PARTITIONS
Description
To add a new partition after the existing last partition .
Example
SQL> alter table mani add partition m3 values less than(99);

Table altered.

SQL> insert into mani values(98,67,89);

1 row created.

SQL> select * from mani partition(m3);

```
       ENG        TAM        MAT
---------- ---------- ----------
        98         67         89
```

SPLITTING PARTITIONS
Description
To split a partition into two.

Example
SQL>alter table mani split partition m1 at(20) into(partition m10,partition m11);

Table altered.

SQL> select * from mani partition(m10);

```
       ENG        TAM        MAT
---------- ---------- ----------
        12         76         54
```

```
SQL> select * from mani partition(m11);

       ENG        TAM        MAT
---------- ---------- ----------
        34         56         78

SQL> select * from mani partition(m1);
                      *
ERROR at line 1:
ORA-02149: Specified partition does not exist
```

TRUNCATING PARTITION

Description

It is used to truncate the values in the partition without disturbing other partition values. Structure can't effect when the partition truncated.

Syntax

Alter table <table name> TRUNCATE partition <partition name>;

Example

Alter table mani truncate partition m10;

DROPPING PARTITIONS

Description

To drop a specified partition.

Example

```
SQL> select * from mani;

       ENG        TAM        MAT
---------- ---------- ----------
        34         56         78
        12         76         54
        78         56         89
        40         65         43
        98         67         89

SQL> alter table mani drop partition m10;

Table altered.
```

SQL> alter table mani drop partition m11;

Table altered.

SQL> select * from mani;

ENG	TAM	MAT
78	56	89
40	65	43
98	67	89

EXCHANGING TABLE PARTITIONS(The two table structure must be same)

SQL> select * from mani partition(m2);

ENG	TAM	MAT
78	56	89
40	65	43

SQL> select * from ram;

ENG	TAM	MAT
34	67	78
67	34	89

SQL> alter table mani exchange partition m2 with table ram;

Table altered.

SQL> select * from ram;

ENG	TAM	MAT
78	56	89
40	65	43

SQL> select * from mani partition(m2);

```
       ENG        TAM        MAT
---------- ---------- ----------
        34         67         78
        67         34         89
```

SYNONYM

Description

A synonym is a database object, which is used as an alias(alternative name) for a table, view or sequence.

Syntax

create [public] synonym <synonym_name> for <table_name>;

Example

SQL> create synonym mani for bab;

Synonym created.

SQL> select * from mani;

```
NAME                 ENG        TAM        MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                 200         89         90 12-MAR-98
ddl                   67         45         67 12-MAR-99
sandhya               45         67         89 12-MAY-99
mani                  67         89         56 12-AUG-96
kavi                  56         78         90 12-MAR-98
ram                   45         56         78 10-JAN-98
```

6 rows selected.

SQL> grant all on mani to system;

Grant succeeded.

SQL> drop synonym mani;

Synonym dropped.

SEQUENCES

Description

A sequence is a database object, which can generate unique, sequential integer values. Generate primary key or unique key values automatically. nextval is initial value, currval is value of sequence. sequence name must be in caps.

Syntax

create sequence <seq_name> [increment by n] [start with n] [maxvalue n]

Example

```
SQL> create sequence frook
  2 increment by 1
  3 start with 1
  4 maxvalue 10
  5 minvalue 1
  6 cycle
  7 cache 4;

Sequence created.
SQL>insert into bab(name,eng,tam,mat) values('babu'||frook.nextval,50,34,100);

1 row created.

SQL> select frook.currval from dual;

   CURRVAL
----------
         2
SQL> select * from bab;


NAME                 ENG        TAM        MAT ODATE
---------------- ---------- ---------- ---------- ---------
babu                  200         89         90 12-MAR-98
ddl                    67         45         67 12-MAR-99
sandhya                45         67         89 12-MAY-99
mani                   67         89         56 12-AUG-96
kavi                   56         78         90 12-MAR-98
```

```
babu2                50       34       100
ram                 45       56       78 10-JAN-98

7 rows selected.

SQL> alter sequence frook maxvalue 15;

Sequence altered.
```

DROPING A SEQUENCE

Syntax

Drop sequence <sequencename>;

Example

drop sequence frook;

TO DISPLAY SEQUENCES

Example1

SQL>select * from user_sequences; (OR) select * from user_sequences where sequence_name = '<sequence name'>;

```
SEQUENCE_NAME                    MIN_VALUE   MAX_VALUE INCREMENT_BY C O
CACHE_SIZE
------------------------------ ---------- ---------- ------------ - - ----------
LAST_NUMBER
-----------
FROOK                                 1         15          1 Y N         4
         3
```

Example2

SQL>select * from all_sequences;

```
SEQUENCE_OWNER                 SEQUENCE_NAME                  MIN_VALUE
------------------------------ ------------------------------ ----------
 MAX_VALUE INCREMENT_BY C O CACHE_SIZE LAST_NUMBER
---------- ------------ - - ---------- -----------
MDSYS                          SAMPLE_SEQ                             1
1.0000E+27           1 N N         20           1

MDSYS                          SDO_IDX_TAB_SEQUENCE                   1
1.0000E+27           1 Y N         20           1
```

SCOTT FROOK 1
15 1 Y N 4 3

VIEW
Description
A view is a tailored presentation of the data contained in one or more tables.
Syntax
Create [or replace] [[no] [force]] view <view_name> [column alias name...] as
<query> [with [check option] [read only] [constraint]];

Example1
SQL> create view ddl as select * from bab where eng > 60;

View created.

SQL> select * from ddl;

NAME	ENG	TAM	MAT	ODATE
babu	200	89	90	12-MAR-98
ddl	67	45	67	12-MAR-99
mani	67	89	56	12-AUG-96

SQL> grant all on ddl to system;

Grant succeeded.

Example2
SQL> update ddl set tam = 100 where name = 'babu';

1 row updated.

SQL> select * from ddl;

NAME	ENG	TAM	MAT	ODATE
babu	200	100	90	12-MAR-98
ddl	67	45	67	12-MAR-99

```
mani                 67        89        56 12-AUG-96
```

Example3

```
SQL> create or replace view raju as select * from bab where tam < 50
    with check option constraint vt;

View created.

SQL> select * from raju;

NAME                 ENG       TAM       MAT ODATE
-------------- ---------- ---------- ---------- ---------
ddl                  67        45        67 12-MAR-99
babu2                 50        34       100

SQL> update raju set tam = 55 where eng = 67;

ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Example4

```
SQL> create or replace view ramesh as select * from bab with read only;

View created.

SQL> select * from ramesh;

NAME                 ENG       TAM       MAT ODATE
-------------- ---------- ---------- ---------- ---------
babu                 200       100        90 12-MAR-98
ddl                  67        45        67 12-MAR-99
sandhya               45        67        89 12-MAY-99
mani                  67        89        56 12-AUG-96
kavi                 56        78        90 12-MAR-98
babu2                 50        34       100
ram                  45        56        78 10-JAN-98

7 rows selected.

SQL> update ramesh set eng = 100 where tam = 78;
            *
```

ERROR at line 1:
ORA-01733: virtual column not allowed here

CREATING VIEWS WITH ERRORS
Description
To create a view with errors, include the FORCE option in the CREATE VIEW command

Example
SQL> create force view jack as select * from sasi;

Warning: View created with compilation errors.
NOTE:
The table sasi is created and insert values, the values is automatically updated in the view table jack.

SQL> create table sasi(a number(3),b number(3));

Table created.

SQL> select * from sasi;

A	B
12	34
67	57
56	87

SQL> select * from jack;

A	B
12	34
67	57
56	87

SQL> alter view jack compile;

View altered.

DML STATEMENTS AND JOIN VIEWS

Description

Joining of tables is also possible in a view.Any UPDATE,INSERT, or DELETE statement on a join view can modify only one underlying base table.

Example

```
SQL> create view jegan as select a.name,rollno,a.eng from bab a,stud
where a.name=stud.name;

View created.

SQL> select * from jegan;

NAME            ROLLNO       ENG
--------------- ---------- ----------
babu               500       200
ddl               503        67
sandhya             504        45

SQL> update jegan set eng = 300 where name = 'ddl';
            *
ERROR at line 1:
ORA-01779: cannot modify a column which maps to a non key-preserved table
```

NOTE

The above shows error because the value contains in view jegan from two tables. Only one base table values can possible to update, delete or insert.

FUNCTIONS IN VIEW

Example

```
SQL> select * from bab;

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
ddl                67        45        67 12-MAR-99
sandhya              45        67        89 12-MAY-99
mani                67        89        56 12-AUG-96
kavi                56        78        90 12-MAR-98
```

```
ram                45        56        89 10-JAN-98

6 rows selected.

SQL> create view arun as select mat,sum(eng+tam) totengtam from bab
    group by mat;

View created.

SQL> select * from arun;

      MAT      TOTENGTAM
       -------    ----------
       56       156
       67       112
       89       213
       90       434
```

PARTITION VIEW

Description

Work only no: of columns and column data types are same in all used tables.

Syntax

```
Create view <viewname> as
select * from <tablename1>
union all
select * from <tablename2>
union all
select * from <tablename3>
```

Example

```
SQL> create view anu1 as
  2  select * from a
  3  union all
  4  select * from b
  5  union all
  6  select * from usha1;

View created.
```

SQL> select * from anu1;

```
         A
----------
        54
        54
        89
        90
        99
       900
```

7 rows selected.

DROPPING VIEWS
Example
SQL> drop view anu1;

View dropped.

INDEX
Description
We can create indexes explicitly to speed up SQL statement execution on a table.
Example
SQL> create index raj on bab(eng);

Index created.

SQL> drop index raj;

Index dropped.

UNIQUE INDEXES
Description
Indexes can be unique or non-unique. Unique indexes guarantee that no two rows of a table have duplicate values in the columns that define the index.

Example

```
SQL> create unique index venkat on bab(eng);
                         *
ERROR at line 1:
ORA-01452: cannot CREATE UNIQUE INDEX; duplicate keys found

SQL> create unique index venkat on bab(name);

Index created.
```

COMPOSITE INDEXES

Description

A composite index(also called a concatenated index) is an index created on multiple columns of a table.

Example

```
SQL> create index mono on bab(name,mat);

Index created.
```

REVERSE KEY INDEXES

Description

Reverse each byte of the column being indexed while keeping the column order.

We cannot rebuild a normal index as a reverse key index.

Normal_index	Reverse_index
A100001	10001A

Example

```
SQL> create index jack on bab(eng) reverse;

Index created.

SQL> alter index jack rebuild noreverse;

Index altered.
```

BITMAP INDEXES

Description

The number of distinct values is small, compared to the number of rows in the table. At that time we use bitmap index.

Example

```
SQL> create bitmap index frook on bab(eng);

Index created.
```

INDEX ORGANIZED TABLES

Description

An index organized table differs from a regular table. Values are always arranged in ascending order based on primary key variable.

Example

```
SQL> create table ganga(eng number(3) primary key, tam
    number(3)) organization index;

SQL> select * from ganga;

       ENG        TAM
---------- ----------
        36         18
        45         32
        78         54
```

PARTITIONING IN INDEX

Description

Like table partitions, index partitions could be in different tablespaces.

Syntax

```
Create index <index name>
on <tablename>(<col1>,<col2>) [global/local]
partition values less than (<value1>) tablespace <tablespacename1>,
partition values less than (<value2>) tablespace <tablespacename2>;
```

NOTE

Tablespacename 1,Tablespacename 2 refer to the names of the tablespaces

which are associated with the partitions. The clause tablespace <tablespacename> can be ignored, as partitions are stored in default user tablespace.

LOCAL INDEXES

Description

The partition key in the partition index should refer to the same rows as that of the underlying table partition.

Example

```
SQL> create table mani(eng number(3),tam number(3),mat number(3)) partition
by range(eng) (partition  m1 values less than (40),partition m2
values less than (90));

Table created.

SQL> create index king on mani(eng) local;

Index created.

SQL>select segment_name, partition_name, segment_type, tablespace_name
from     user_segments where segment_name='king';

no rows selected
```

GLOBAL INDEXES

The keys of a global index may refer to rows stored in more than one underlying partition.

Example

```
SQL> create table mani(eng number(3),tam number(3),mat number(3)) partition
by range(eng) (partition  m1 values less than (40),partition m2
values less than (90),partition m3 values less than (maxvalue));

Table created.

SQL> create index ravi on mani(eng) global
  2  partition by range (eng) (partition r1 values less than (36),
  3  partition r2 values less than (maxvalue));
```

Index created.

ALTER INDEX

Syntax

Alter index <index name> rebuild partition <partition name>;

OBJECT IN ORACLE8

1. Abstract data types
2. Object views
3. Varying arrays
4. Nested Tables
5. Object Tables
6. Object Views with REFs

ABSTRACT DATA TYPES

Description

Abstract data types are data types that consist of one or more subtypes.

Example

SQL> create or replace type mark1 as object (eng number(3), tam number(3), mat number(3));

Type created.

IMPLEMENTING AN OBJECT TYPE AS A COLUMN OBJECT

Description

Use abstract type that is created above. Create a table named jack.

Example

SQL> create table jack (rollno number,name varchar2(10),studmark mark1);

Table created.

SQL> desc jack;

Name	Null?	Type
ROLLNO		NUMBER(3)
NAME		VARCHAR2(10)
STUDMARK		MARK1

The same can be viewed through a data dictionary table called USER_TAB_COLUMNS through which the same output as shown below.

```
SQL> select column_name,data_type from user_tab_columns
    where table_name = 'jack';

 column_Name                          Data_Type
 ------------------------------------ -------- ------------
 ROLLNO                               NUMBER(3)
 NAME                                 VARCHAR2(10)
 STUDMARK                             MARK1
```

when the user wants to view the actual values of the MARK1 data type then a query using the USER_TYPE_ATTRS data dictionary table can be used.

```
SQL> select attr_name, length, attr_type_name from user_type_attrs where
    type_name = 'mark1';

 Attr_name          length             Attr_Type_Name
 -----------      ---------------    -------------------------
 ENG                3                  NUMBER(3)
 TAM                3                  NUMBER(3)
 MAT                3                  NUMBER(3)
```

INSERTING RECORDS INTO ABSTRACT DATA TYPES

Example

```
SQL> insert into jack values (501,'babu',mark1(56,78,43));

1 row created.

SQL> select * from jack;

   ROLLNO NAME
---------- ----------
STUDMARK(ENG, TAM, MAT)
-------------------------------------------------------
     501 babu
MARK1(56, 78, 43)
```

SELECTING FROM ABSTRACT DATA TYPES

Example
SQL> select name from jack;

NAME
----
babu

SQL> select eng from jack;

ERROR at line 1:
ORA-00904: invalid column name

SQL> select a.studmark.eng from jack a;

STUDMARK.ENG

        56

UPDATING RECORDS IN ABSTRACT DATA TYPES

Example
SQL> update jack a set a.studmark.eng = 100 where name = 'babu';

1 row updated.

SQL> select * from jack;
   ROLLNO NAME
---------- ----------
STUDMARK(ENG, TAM, MAT)
----
     501 babu
MARK1(56, 78, 43)

     503 ddl
MARK1(45, 78, 32)

DELETING RECORD FROM ABSTRACT DATA TYPES

Example
SQL> delete from jack a where a.studmark.eng = 45;

1 row deleted.

DROPPING OBJECT TYPES

Example

```
SQL> drop type mark1;
drop type mark1
*
ERROR at line 1:
ORA-02303: cannot drop or replace a type with type or table dependents

SQL> drop type mark1 force;

Type dropped.

SQL> select * from jack;

   ROLLNO NAME
---------- ----------
      501 babu
```

INDEXING ABSTRACT DATA TYPE ATTRIBUTES

Example

```
SQL> create index sam on jack(studmark.eng);

Index created.

SQL>  select  owner,index_name,index_type,table_owner,table_name,table_type
from
  2  all_indexes where owner = 'scott';

OWNER                INDEX_NAME        INDEX_TYPE TABLE_OWNER
         TABLE_NAME        TABLE_TYPE
-------  ----------       ----------       -----------       ----------       ----------
SCOTT                SAM            NORMAL     SCOTT              JACK       TABLE
```

OBJECT VIEWS

Description

To implement the Object Oriented concepts in the applications without rebuilding or recreating the entire application. The ability to overlay the Object Oriented structure such as abstract data types on existing tables is requied.

Example
SQL> select * from stud;

```
   ROLLNO NAME            ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
       500 babu            250       278       300
       501 dhana           200        89        90
       503 ddl             200        89        90
       504 sandhya          67        89        45
       569 aaaa             67        34       100
       100 bbbbb            89        45        65
```

6 rows selected.

OBJECT VIEWS is created based on above table.

```
SQL> create or replace type first as object(rollno number(3),name varchar2(10));

Type created.

SQL> create or replace type second as object(eng number(3),tam number(3),var1
first);

Type created.

SQL>  create or replace type third as object(mat number(3),var2 second);

Type created.

SQL>  create or replace view stud_view (mat,var2)
  2   as (select mat,second(eng,tam,
  3   first(rollno,name))
  4   from stud);

View created.

SQL> insert into stud_view values(200,second(100,150,first(508,'frook')));

1 row created.
```

SQL> select * from stud;

ROLLNO	NAME	ENG	TAM	MAT
500	babu	250	278	300
501	dhana	200	89	90
503	ddl	200	89	90
504	sandhya	67	89	45
569	aaaa	67	34	100
100	bbbbb	89	45	65
508	frook	100	150	200

7 rows selected.

VARYING ARRAYS
Description
These help in storing repeating attributes of a record in a single row.

CREATING A VARYING ARRAY
Description
The record could carry a maximum of 5 items. so we have given varray(5).

Example
SQL> create type name as varray(5) of varchar2(10);

SQL> create type eng as varray(5) of number(3);

SQL> create type tam as varray(5) of number(3);

SQL> create table mark1 (rollno number(3),student_name name,english eng,tamil tam);

Table created.

SQL> desc mark1;

Name	Null?	Type

```
ROLLNO                          NUMBER(3)
STUDENT_NAME                         NAME
ENGLISH                         ENG
TAMIL                          TAM
```

```
SQL> select column_name, data_type from user_tab_columns where
    table_name = 'mark1';

Column_name                          Data_type
----------------              --------------------
ROLLNO                 NUMBER
STUDENT_NAME                NAME
ENGLISH                ENG
TAMIL                 TAM
```

```
SQL> select typecode, attributes from user_types where type_name = 'eng';

Typecode                          Attributes
----------                    --------------------
COLLECTION                              0
```

```
SQL> select coll_type, elem_type_owner,elem_type_name,upper_bound,length
    from  user_coll_types where type_name = 'eng';

Coll_type     Elem_type_owner Elem_type_name  Upper_bound        Length
---------     --------------- --------------  -----------      --------
VARRING       ARRAY                         NUMBER                    5               3
```

INSERTING RECORDS INTO VARYING ARRAYS

Example

```
SQL> insert into mark1 values (510,name('babu','ddl','san','jegan','frook'),
  2  eng(45,56,78,34,56),tam(23,45,67,89,90));

1 row created.

SQL> select * from mark1;

   ROLLNO
----------
```

STUDENT_NAME

ENGLISH

TAMIL

510
NAME('babu', 'ddl', 'san', 'jegan', 'frook')
ENG(45, 56, 78, 34, 56)
TAM(23, 45, 67, 89, 90)

SQL> select english from mark1;

ENGLISH

ENG(45, 56, 78, 34, 56)

NESTED TABLES
Description
Varying arrays have a limited number of entries, whereas nested tables have no limit on the number of entries per row. A nested table is a table within a table. A nested table is a table within a table.

Example
SQL> create type first as object (name varchar2(10),eng number(3));

Type created.

SQL> create type second as table of first;

Type created.

SQL> create table mark2 (rollno number(5), var1 second)
nested table var1 store as frook1;

Table created.

SQL> desc mark2;
Name Null? Type

ROLLNO NUMBER(5)

VAR1 SECOND

INSERTING RECORDS INTO NESTED TABLES

Example

```
SQL> insert into mark2 values(504,second(first('babu',68)));

1 row created.

SQL> select * from mark2;

   ROLLNO
----------
VAR1(NAME, ENG)
------------------------------------------------------------
     504
SECOND(FIRST('babu', 68))

     505
SECOND(FIRST('ddl', 76))
```

USES OF THE FUNCTION

Example

```
SQL> insert into the (select var1 from mark2 where rollno = 504)
  2  values (first('frook',90));

1 row created.

SQL> update the (select var1 from mark2 where rollno = 504) set eng = 100 where
name = 'babu';

1 row updated.
```

OBJECT TABLES

Example

```
SQL> create type first as object (eng number(3),
    tam number(3));

Type created.

SQL> create table second of first;
```

Table created.

SQL> insert into second values(first(76,89));

1 row created.

SQL> select * from second;

```
     ENG       TAM
---------- ----------
      76        89
      72        65
```

SQL> select eng from second;

```
     ENG
----------
      76
      72
```

REF OPERATOR

Description

It allows referencing of existing row objects. Each of the row objects has an OID value assigned to it. The OID value can be seen below

Example

SQL> select ref(b) from second b;

```
REF(B)
--------------------------------------------------------------------------------
0000280209193F8CC5E244445899490D1FF4800A32D052F9467C8646E0AFC17F7
F294F3CED0040F0
960000

0000280209C5292359106F482BAF0F920BC818EBC7D052F9467C8646E0AFC17F7
F294F3CED0040F0
960001
```

DEREF OPERATOR

Description

It takes a reference value and return the value of the row objects. The deref operator takes as its argument the OID generated for a references.

Example

```
SQL> create table mark1 (ss number(3), var1 ref first);

Table created.

SQL> desc mark1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SS                                                 NUMBER(3)
 VAR1                                               REF OF FIRST

SQL> insert into mark1 select 99,ref(a) from second a where eng = 76;

1 row created.

SQL> select * from mark1;

        SS
----------
VAR1
--------------------------------------------------------------------------------
        99
0000220208193F8CC5E244445899490D1FF4800A32D052F9467C8646E0AFC17F7
F294F3CED

SQL> select deref(a.var1) from mark1 a;

DEREF(A.VAR1)(ENG, TAM)
----------------------------------------
FIRST(76, 89)
```

VALUE OPERATOR

Example

```
SQL> select value(a) from second a;

VALUE(A)(ENG, TAM)
-----------------------------------
FIRST(76, 89)
FIRST(72, 65)
```

INVALID REFERENCES

Description

The object to which a reference points can be deleted.

Example

```
SQL> select deref(a.var1) from mark1 a;

DEREF(A.VAR1)(ENG, TAM)
---------------------------------------
FIRST(76, 89)
FIRST(70, 41)

SQL> delete from second where eng = 70;

1 row deleted.

SQL> select deref(a.var1) from mark1 a;

DEREF(A.VAR1)(ENG, TAM)
---------------------------------------
FIRST(76, 89)
```

OBJECT VIEWS WITH REFS

Example

```
SQL> create table first(rollno number(3) primary key, eng number(3));

Table created.

SQL> create table second(rollno number(3),tam number(3),foreign
     key(rollno) references  first(rollno));

Table created.
```

```
SQL> select * from first;

   ROLLNO       ENG
---------- ----------
      100        50
      101        60

SQL> select * from second;

   ROLLNO       TAM
---------- ----------
      100        65
      101        63
```

GENERATING OIDS

Example

```
SQL> create or replace type first_ty as object (rollno
    number(3),eng number(3));

Type created.

SQL> create view first_vi of first_ty with OBJECT OID(rollno)
    as select * from first;

View created.

SQL> select MAKE_REF (first_vi,100) from first;

MAKE_REF(FIRST_VI,100)
--------------------------------------------------------------------------------
00004A038A004686D0CBD1BC0B41BF8C9FEAC2364FF3F20000014260100010001000
100290000000000
090603002A00078401FE0000000A02C2020000000000000000000000000000000000
000000000

00004A038A004686D0CBD1BC0B41BF8C9FEAC2364FF3F20000014260100010001000
100290000000000
090603002A00078401FE0000000A02C2020000000000000000000000000000000000
000000000
```

GENERATION OF REFERENCES

Example

```
SQL>create view second_vi as select MAKE_REF (first_vi,rollno) var1,
   tam from second;

View created.
```

QUERYING OBJECT VIEWS

Example

```
SQL> select deref(a.var1) from second_vi a;

DEREF(A.VAR1)(ROLLNO, ENG)
------------------------------------------
FIRST_TY(100, 50)
FIRST_TY(101, 60)
```

TIP FOR ORACLE & DEVELOPER

1. set autocommit on/off-on means commits automatically after execute.
2. set head on/off-suppresses the column headings in a query result.
3. To display property sheet of 2 items at the same time
   (i)Select the first object,properties are displayed in the Property Palette. click the Freeze/Un freeze button in property menu.
   (ii)Shift+double-click the second object.If the second window is on top of the first.
5. select * from all_views; - to view all views.

COLUMN HEADINGS

Description

Column allows us to change the heading and format of any column in a select statement.

a<value> ----alphabets(a20 means 20 spaces allocated)

99999 -----numeric(numbers are formatted using '$','9')

Syntax

Column <column name> HEADING <column heading> FORMAT <format model>

Example

```
SQL> select * from first;

   ROLLNO       ENG NAM
---------- ---------- ----------
      100       150 babu
      101        60 ddl

SQL> column nam heading 'student name' format a20;
SQL> column eng heading 'english mark' format 9,99;
SQL> select * from first;

   ROLLNO   english mark         student name
----------       ------------     --------------------
      100       1,50   babu
      101         60    ddl
```

TRUNCATED

Description

To remove the text values based on format model.

Example

```
SQL> column nam heading 'student_names ' format a5 truncated;
SQL> select * from first;

   ROLLNO   english mark         stude
----------       ------------     -----
      100       1,50   babu
      101         60    ddl
```

CLEAR COLUMN HEADING

Syntax

1. Clear column {clear all column headings}
2. column <column name> clear {clear particular column headings}

Example

```
SQL> column eng clear;
SQL> select * from first;

   ROLLNO       ENG stude
---------- ---------- -----
      100        150 babu
      101         60 ddl

SQL> clear column
columns cleared
SQL> select * from first;

   ROLLNO       ENG NAM
---------- ---------- ----------
      100        150 babu
      101         60 ddl
```

ALIASES

Description

Aliases can be specified for the computed columns and the heading and formatting features for the aliases.

Example

```
SQL> select * from first;

   ROLLNO       ENG NAM
---------- ---------- ----------
      100        150 babu
      101         60 ddl

SQL> column english format $99,999.00;
```

SQL> select eng*.20 as english from first;

```
   ENGLISH
-----------
    $30.00
    $12.00
```

BREAK

Syntax

Break on <column name>

Example1

SQL> select * from bab;

```
NAME                ENG       TAM       MAT       ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100       90          12-MAR-98
ddl                 67        45        67          12-MAR-99
sandhya             45        67        89          12-MAY-99
mani                67        89        56          12-AUG-96
kavi                56        78        90          12-MAR-98
ram                 45        56        89          10-JAN-98
```

6 rows selected.

SQL> break on eng;

SQL> select * from bab order by eng;

```
NAME                ENG       TAM       MAT     O DATE
--------------- ---------- ---------- ---------- ---------
sandhya             45        67        89        12-MAY-99
ram                 56        89                  10-JAN-98
kavi                56        78        90        12-MAR-98
ddl                 67        45        67        12-MAR-99
mani                89        56                  12- AUG-96
babu                200       100       90        12-MAR-98
```

6 rows selected.

Example2

SQL> break on eng skip 1

SQL> select eng,tam,mat from bab;

ENG	TAM	MAT
200	100	90
67	45	67
45	67	89
67	89	56
56	78	90
45	56	89

6 rows selected.

CLEAR BREAK

Example

SQL> clear break;

breaks cleared

COMPUTE

Syntax

Break on <column name>;

compute <function> of <column name> on <break columnname>;

Example

SQL> select * from bab;

NAME	ENG	TAM	MAT	ODATE
babu	200	100	90	12-MAR-98
ddl	67	45	67	12-MAR-99
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96
kavi	56	78	90	12-MAR-98

```
SQL> break on name;
SQL> compute sum of eng on name;
SQL> select * from bab;

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
*************** ----------
sum                 200
ddl                 67        45        67 12-MAR-99
*************** ----------
sum                 67
sandhya              45        67        89 12-MAY-99
*************** ----------
sum                 45
mani                 67        89        56 12-AUG-96
*************** ----------
sum                 67
kavi                56        78        90 12-MAR-98
*************** ----------
sum                 56
```

TO CALCULATE GRANT TOTAL

Syntax

On Report

Example

```
SQL> break on report;
SQL> compute sum of eng on report;
SQL> select * from bab;

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
ddl                 67        45        67 12-MAR-99
sandhya              45        67        89 12-MAY-99
mani                 67        89        56 12-AUG-96
kavi                 56        78        90 12-MAR-98
            ----------
sum                 435
```

TO CLEAR BREAK AND COMPUTE

Example

Clear break;

Clear compute;

TO DISPLAY SETTINGS

Syntax

Show all (or) show <option>

Example

show pagesize;

show newpage;

show linesize;

SETTING WITH SET COMMAND

Example

Set pause on/off;

set pagesize 20;

set linesize 100;

set sqlprompt 'babu>';

set sqlterminator '=';

LIST

Description

To view latest command that we type in sql prompt;

Example

SQL>select name,eng from bab;

NAME ENG

--- ---

babu 200

ddl 67

sandhya 45

mani 67

kavi 56

SQL>list;

1* select name,eng from bab

CHANGE MISPELLED WORDS WITHOUT USING EDITOR

Syntax

Change/old expression/new expression;

Example1

```
SQL>select enb from first;

ERROR at line 1:
ORA-00904: invalid column name

SQL>change/enb/eng;
  1* select eng from first
```

Example2

```
SQL>select * from first;

   ROLLNO       ENG NAM
---------- ---------- ----------
      100        50 babu
      101        60 ddl

SQL>input where nam = 'babu';
SQL>list;
  1  select * from first
  2* where nam = 'babu'
SQL>select * from first
  2  where nam = 'babu';

   ROLLNO       ENG NAM
---------- ---------- ----------
       100          50 babu
       101
```

COMMENT LINE
Description
–- {rem or remark Single line comment}
/* */ {Multiple line comment}

rem can work outside the program.
/* */ can work inside the program when declaring in pl/sql.

SPOOL
Description
To store and print the query results. The extension is .lst.
Syntax
spool on;
spool <filename>.<extension>;
type sql query;
spoll off;
Example
SQL>spool on;
SQL>spool babu.lst;
SQL>select * from bab where eng = 65;

no rows selected

SQL>select * from bab where eng = 67;

NAME ENG TAM MAT ODATE
--- --- --- --- ---
ddl 67 45 67 12-MAR-99
mani 67 89 56 12-AUG-96

SQL>spool off;

Using file menu to open the file babu.lst the following result will display
SQL>
1 SQL>select * from bab where eng = 65;
2 no rows selected
3 SQL>select * from bab where eng = 67;
4 NAME ENG TAM MAT ODATE
5 --- --- --- --- ---

```
 6  ddl                67        45        67 12-MAR-99
 7  mani                 67        89        56 12-AUG-96
 8* SQL>spool off;
```

PSEUDO COLUMNS

Description

Rowid displays the identification of each and every.

Rownum displays the rownumbers.

Example

```
SQL>select * from first;

   ROLLNO       ENG NAM
---------- ---------- ----------
      100         50 babu
      101         60 ddl

SQL>select rowid from first;

ROWID
------------------
AAAH6YAABAAAPCWAAA
AAAH6YAABAAAPCWAAB

SQL>select rownum from first;

   ROWNUM
----------
         1
         2
```

QUERY TO DISPLAY DUPLICATE ROWS

Example

```
SQL> select * from bab;

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
```

babu 200 100 90 12-MAR-98
ddl 67 45 67 12-MAR-99
sandhya 45 67 89 12-MAY-99
mani 67 89 56 12-AUG-96
kavi 56 78 90 12-MAR-98

SQL> select * from bab x where rowid not in(select min(rowid) from bab where eng = x.eng);

NAME	ENG	TAM	MAT	ODATE
mani	67	89	56	12-AUG-96

QUERY TO DISPLAY ALTERNATE ROWS

Example

SQL> select name,eng from bab where(eng,rownum) in(select eng,mod(rownum,4) from bab);

NAME	ENG
kavi	56
ddl	67

QUERY TO DISPLAY OTHER ALTERNATE ROWS

Example

SQL> select * from bab where rowid not in(select rowid from bab where (eng,rownum) in (select eng,mod(rownum,4) from bab));

NAME	ENG	TAM	MAT	ODATE
babu	200	100	90	12-MAR-98
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96

QUERY TO DELETE ALTERNATE ROWS

Example

SQL> delete from bab where (eng,rownum) in(select eng,mod(rownum,4) from bab);

2 rows deleted.

SQL> select * from bab;

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
sandhya              45        67        89 12-MAY-99
mani                 67        89        56 12-AUG-96
```

QUERY TO PRINT SOME TEXT

Example

SQL> select name,eng,decode(mod(rownum,4),0,'********') print from bab;

```
NAME                ENG PRINT
--------------- ---------- --------
babu                200
ddl                 67
sandhya              45
mani                 67 ********
kavi                56
```

QUERY TO GET COLUMN WITHOUT SPECIFYING THE COLUMN NAME

Example

SQL> select * from babu;

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
ddl                 67        45         67 12-MAR-99
sandhya              45        67        89 12-MAY-99
mani                 67        89        56 12-AUG-96
kavi                56        78        90 12-MAR-98
```

SQL> select &a,&b,&z from bab where eng = 67;
Enter value for a: tam
Enter value for b: name
Enter value for z: odate
old   1: select &a,&b,&z from bab where eng = 67
new   1: select tam,name,odate from bab where eng = 67

```
       TAM NAME          ODATE
---------- -------------- ---------
        45 ddl           12-MAR-99
        89 mani          12-AUG-96
```

DELETE DUP ROWS BUT LEAVING ONE ROW UNDELETED

Example1

SQL> delete from bab where eng = 67 and rowid not in(select min(rowid) from bab where eng = 67);

1 row deleted.

SQL> select * from bab;

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                 200       100        90 12-MAR-98
ddl                   67        45        67 12-MAR-99
sandhya               45        67        89 12-MAY-99
kavi                  56        78        90 12-MAR-98
```

Example2

SQL> delete from bab x where rowid not in(select min(rowid) from bab where mat = x.mat);

1 row deleted.

SQL> select * from bab;

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                 200       100        90 12-MAR-98
ddl                   67        45        67 12-MAR-99
sandhya               45        67        89 12-MAY-99
mani                  67        89        56 12-AUG-96
```

DISPLAY ALL COLUMNS AND ROWID WITHOUT SPECIFYING THE COLUMN NAME

Example1

SQL> select rowid,bab.* from bab;

```
ROWID           NAME              ENG       TAM       MAT ODATE
------------------ -------------- ---------- ---------- ---------- ---------
AAAH4AAABAAAPCZAAA babu               200       100        90 12-MAR-98
AAAH4AAABAAAPCZAAB ddl                67        45        67 12-MAR-99
AAAH4AAABAAAPCZAAC sandhya            45        67        89 12-MAY-99
AAAH4AAABAAAPCZAAD mani               67        89        56 12-AUG-96
AAAH4AAABAAAPCZAAE kavi               56        78        90 12-MAR-98
```

Example2

```
SQL> select rowid,&n from bab;
Enter value for n: mat
old   1: select rowid,&n from bab
new   1: select rowid,mat from bab

ROWID                 MAT
------------------ ----------
AAAH4AAABAAAPCZAAA        90
AAAH4AAABAAAPCZAAB        67
AAAH4AAABAAAPCZAAC        89
AAAH4AAABAAAPCZAAD        56
AAAH4AAABAAAPCZAAE        90
```

DISPLAY MAX,MIN AND OTHER VALUES

Example1

```
SQL> select name,mat,'maximum' from bab where mat=(select max(mat) from
bab)
 union select name,mat,'   ' from bab where mat!=(select max(mat) from bab);

NAME              MAT 'MAXIMU
--------------- ---------- -------
babu               90 maximum
ddl                67
kavi               90 maximum
mani               56
sandhya            89
```

Example2

```
SQL> select name,mat,'maximum' from bab where mat = (select max(mat) from
bab)
union select name,mat,'   ' from bab where mat != (select max(mat) from bab)
```

```
and mat != (select min(mat) from bab) union select name,eng,'minimum'
from   bab where mat = (select min(mat) from bab);

NAME                 MAT 'MAXIMU
---------------- ---------- -------
babu                  90 maximum
ddl                  67
kavi                  90 maximum
mani                   67 minimum
sandhya                89
```

DISPLAY COLUMN AND COLUMN+1 VALUES

Example1

```
SQL> select eng from bab union select eng+1 from bab;

       ENG
----------
        45
        46
        56
        57
        67
        68
       200
       201

8 rows selected.
```

Example2

```
SQL> select eng from bab union select eng+1 from bab minus(select eng
from bab);

       ENG
----------
        46
        57
        68
       201
```

DISPLAY ROWS BETWEEN N^{TH} TO M^{TH} ROW

Example1
SQL> select * from bab;

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                 200       100        90 12-MAR-98
ddl                  67        45        67 12-MAR-99
sandhya               45        67        89 12-MAY-99
mani                  67        89        56 12-AUG-96
kavi                  56        78        90 12-MAR-98
```

SQL>select * from bab where rownum < &a minus select * from bab where rownum < &b;
Enter value for a: 4
Enter value for b: 2

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
ddl                  67        45        67 12-MAR-99
sandhya               45        67        89 12-MAY-99
```

Example2
SQL> select * from bab where rownum < &&a minus select * from bab where rownum < &&a-1;
Enter value for a: 4

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
sandhya               45        67        89 12-MAY-99
```

DISPLAY LINE NUMBER WITH A GAP OF 2(ANY NUMBER)
Example
SQL> select name,eng,decode(mod(rownum,2),0,rownum,null) line from bab;

```
NAME                ENG      LINE
--------------- ---------- ----------
babu                 200
ddl                  67         2
```

```
sandhya              45
mani                67        4
kavi                56
```

DISPLAY SECOND MAX VALUE OF A GIVEN COLUMN

Example

```
SQL> select * from bab;

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
ddl                67        45        67 12-MAR-99
sandhya              45        67        89 12-MAY-99
mani                67        89        56 12-AUG-96
kavi                56        78        90 12-MAR-98
mmm                 300       400        76 12-MAY-98

6 rows selected.

SQL> select max(mat) from bab where mat not in (select max(mat) from bab);

  MAX(MAT)
----------
    89
```

DISPLAY THIRD MAX VALUE OF A GIVEN COLUMN

Example

```
SQL> select max(mat) from bab where mat not in (select max(mat) from bab
union
  2  select max(mat) from bab where mat not in (select max(mat) from bab));

  MAX(MAT)
----------
      76
```

DISPLAY Nth MAX VALUE OF A GIVEN COLUMN

Example

```
SQL> select mat from bab a where &enter_no=(select count(mat) from
bab b where a.mat <= b.mat);
```

Enter value for enter_no: 4

```
       MAT
----------
        76
```

DISPLAY A PARTICULAR ROW

Example

SQL> select * from bab a where &n = (select count(rowid) from bab b where a.rowid >= b.rowid);

Enter value for n: 4

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
mani                 67        89        56 12-AUG-96
```

DISPLAY ALL DUPLICATE ROWS IN A TABLE

Example

SQL> select * from bab;

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
ddl                 67        45         67 12-MAR-99
sandhya              45        67         89 12-MAY-99
mani                 67        89         56 12-AUG-96
kavi                 56        78         90 12-MAR-98
mmm                 300       400         76 12-MAY-98
mmm                  23        45        100 12-JAN-99
```

7 rows selected.

SQL> select * from bab a where rowid not in (select max(rowid) from bab b where a.mat = b.mat or a.tam = b.tam or a.eng = b.eng or a.name = b.name or a.odate = b.odate);

```
NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
```

babu 200 100 90 12-MAR-98
ddl 67 45 67 12-MAR-99
mmm 300 400 76 12-MAY-98

NATURAL JOIN & CROSS JOIN

Example

SQL> select * from first;

```
   ROLLNO       ENG NAM
---------- ---------- ----------
      100         50 babu
      101         60 ddl
```

SQL> select * from second;

```
   ROLLNO       TAM
---------- ----------
      100         65
      101         63
```

SQL> select * from first natural join second;

```
   ROLLNO       ENG NAM             TAM
---------- ---------- ---------- ----------
      100         50 babu             65
      101         60 ddl              63
```

SQL> select * from first cross join second;

```
   ROLLNO       ENG NAM          ROLLNO       TAM
---------- ---------- ---------- ---------- ----------
      100         50 babu            100         65
      101         60 ddl             100         65
      100         50 babu            101         63
      101         60 ddl             101         63
```

MERGE COMMAND

Description

The merge command to perform inserts and updates into a single table in a single command.

Example1

```
SQL> select * from sandhya;

       ENG        TAM
---------- ----------
        45         67
         2         45
        78         90

SQL> select * from anusha;

       ENG        TAM
---------- ----------
         2         67
        78         87
        42         90
        80        100

SQL> edit merge

SQL>
  1  merge into sandhya s
  2  using (select eng,tam from anusha) a
  3  on (s.tam=a.tam)
  4  when matched then
  5  update set s.eng = a.eng
  6  when not matched then
  7* insert (s.eng,s.tam) values (a.eng,a.tam)
SQL> /

4 rows merged.

SQL> select * from sandhya;
```

```
       ENG        TAM
---------- ----------
         2         67
         2         45
        42         90
        80        100
        78         87
```

Example2

```
SQL> edit merge;

SQL>
  1  merge into sandhya s
  2  using (select eng,tam from anusha) a
  3  on (s.tam=a.tam and s.tam=500)
  4  when matched then
  5  update set s.eng = a.eng
  6  when not matched then
  7* insert (s.eng,s.tam) values (a.eng,a.tam)
SQL> /

4 rows merged.

SQL> select * from sandhya;

       ENG        TAM
---------- ----------
         2         67
         2         45
        42         90
        80        100
        78         87
        80        100
        78         87
        42         90
         2         67

9 rows selected.
```

REFERENCE BOOK

Example

```
SQL> select * from bab where name like '%a%a%';

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
sandhya              45        67        89 12-MAY-99
```

Example

```
SQL> select * from bab where name = (select name from bab where eng = 67);
                     *
ERROR at line 1:
ORA-01427: single-row subquery returns more than one row
```

Example

```
SQL> select name from bab order by length(name);

NAME
---------------
ddl
babu
mani
kavi
jjjjjj
sandhya

6 rows selected.

SQL> select floor(-55.78) from dual;

FLOOR(-55.78)
-------------
        -56

SQL> select systimestamp from dual;

SYSTIMESTAMP
---------------------------------------------------------
08-APR-06 03.41.28.000000 PM +05:30
```

```
SQL> select to_date('02/22/06','mm/dd/yy') as now from dual;

NOW
---------
22-FEB-06

SQL> select to_char(odate,'ddth-month-yyyy') from bab;

TO_CHAR(ODATE,'DDTH
------------------
12th-march    -1998
12th-march    -1999
12th-may      -1999
12th-august   -1996
12th-march    -1998

SQL> select substr(12345678,1,5)||'-'||substr(12345678,8) from dual;

SUBSTR(
-------
12345-8

SQL> select translate(76342,1234567,'asdfghj') from dual;

TRANS
-----
jhdfs

SQL> select name from bab where name < 'g%';

NAME
---------------
babu
ddl

SQL> select name from bab where name in (select name from stud) union
  2  (select name from student);

NAME
---------------
```

aaaa
babu
bbbbb
ddl
dhana
sandhya

6 rows selected.
Example
SQL> insert all
  2   when eng > 200 then
  3   into bab(eng,tam) values (100,100)
  4   when tam > 75 then
  5 into bab(eng,tam) values (200,200)
  6 select name,eng,tam from bab where eng > 10;

3 rows created.

SQL> select * from bab;

NAME   ENG   TAM   MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu   200   100   90 12-MAR-98
ddl   67   45   67 12-MAR-99
sandhya   45   67   89 12-MAY-99
mani   67   89   56 12-AUG-96
kavi   56   78   90 12-MAR-98
                   200   200
                   200   200
                   200   200

8 rows selected.

Example
SQL>
  1 CREATE OR REPLACE TRIGGER T3 BEFORE INSERT OR UPDATE ON bab
  2 FOR EACH ROW
  3 BEGIN
  4 IF :NEW.eng>100 AND :NEW.name<>'dhana' THEN
  5 RAISE_APPLICATION_ERROR(-20004,'NOT POSSIBLE');

```
  6  END IF;
  7* END;
SQL> /

Trigger created.
SQL> update bab set eng = 102 where name = 'ddl';
update bab set eng = 102 where name = 'ddl'
       *
ERROR at line 1:
ORA-20004: NOT POSSIBLE
ORA-06512: at "SCOTT.T3", line 3
ORA-04088: error during execution of trigger 'SCOTT.T3'

SQL> insert into bab(name,eng) values('dhana',122);

1 row created.

SQL> insert into bab(name,eng) values('ggg',122);
insert into bab(name,eng) values('ggg',122)
            *
ERROR at line 1:
ORA-20004: NOT POSSIBLE
ORA-06512: at "SCOTT.T3", line 3
ORA-04088: error during execution of trigger 'SCOTT.T3'
```

Example

```
SQL>
  1  CREATE OR REPLACE TRIGGER T6 BEFORE INSERT OR UPDATE ON bab
  2  FOR EACH ROW
  3  DECLARE
  4  A NUMBER;
  5  BEGIN
  6  SELECT eng INTO A FROM stud WHERE eng=:NEW.eng;
  7  EXCEPTION
  8  WHEN NO_DATA_FOUND THEN
  9  RAISE_APPLICATION_ERROR(-20004,'PARENT KEY NOT FOUND');
 10* END;
SQL> /

Trigger created.
```

```
SQL> update bab set eng = 67 where name = 'ddl';
update bab set eng = 67 where name = 'ddl'
       *
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at "SCOTT.T6", line 4
ORA-04088: error during execution of trigger 'SCOTT.T6'


SQL> update bab set eng = 250 where name = 'ddl';

1 row updated.

SQL> update bab set eng = 150 where name = 'ddl';
update bab set eng = 150 where name = 'ddl'
       *
ERROR at line 1:
ORA-20004: PARENT KEY NOT FOUND
ORA-06512: at "SCOTT.T6", line 7
ORA-04088: error during execution of trigger 'SCOTT.T6'
```

Example

```
SQL>
  1  CREATE OR REPLACE TRIGGER T9 BEFORE DELETE ON bab FOR EACH ROW
  2  DECLARE CURSOR C1 IS SELECT eng FROM stud WHERE
  3  eng=:OLD.eng;
  4  A NUMBER;
  5  BEGIN
  6  OPEN C1;
  7  FETCH C1 INTO A;
  8  IF C1%FOUND THEN
  9  RAISE_APPLICATION_ERROR(-20111,'CHILD RECORDS FOUND');
 10  END IF;
 11  CLOSE C1;
 12  EXCEPTION WHEN NO_DATA_FOUND THEN
 13  DBMS_OUTPUT.PUT_LINE('DELETING RECORDS...');
 14* END;
SQL> /
```

Trigger created.

SQL> select * from bab;

NAME	ENG	TAM	MAT	ODATE
babu	200	100	90	12-MAR-98
ddl	250	45	67	12-MAR-99
sandhya	45	67	89	12-MAY-99
mani	67	89	56	12-AUG-96
kavi	56	78	90	12-MAR-98
	90	90		

6 rows selected.

SQL> select * from stud;

ROLLNO	NAME	ENG	TAM	MAT
500	babu	250	278	300
501	dhana	200	89	90
503	ddl	200	89	90
504	sandhya	67	89	45
569	aaaa	67	34	100
100	bbbbb	89	45	65

6 rows selected.

SQL> delete from bab where eng = 67;
delete from bab where eng = 67
*
ERROR at line 1:
ORA-20111: CHILD RECORDS FOUND
ORA-06512: at "SCOTT.T9", line 8
ORA-04088: error during execution of trigger 'SCOTT.T9'

SQL> delete from bab where eng = 90;

1 row deleted.

Example1

```
SQL> select to_char(odate,'day') day from bab;

DAY
---------
thursday
friday
wednesday
monday
thursday


6 rows selected.

SQL> select to_char(odate,'d') da from bab order by da;

D
-
2
4
5
5
6


6 rows selected.

SQL> select to_char(odate,'day') day from bab order by to_char(odate,'d');

DAY
---------
monday
wednesday
thursday
thursday
friday


6 rows selected.
```

PART – II

PL/SQL PROGRAMMING

powered by C.Sivanantham

C.Sivanantham

BLOCK DIAGRAM OF PL/SQL

Syntax
Declare
<declaration part>
begin
<executable part>
.....
.....
exception
<exeception part>
end;

VARIABLE
Syntax
<Variable> <data type> <number>;
:= ➔ assigning a value to variable.

SIMPLE PROGRAMS
Example1

```
SQL>
  1  declare
  2  a varchar2(30);
  3  begin
  4  a:='welcome to pl_sql programming';
  5  dbms_output.put_line(a);
  6* end;
  7  /
welcome to pl_sql programming

PL/SQL procedure successfully completed.
```

Example2

```
SQL>
  1  declare
  2  a varchar2(30):='&enter_the_name';
  3  begin
  4  dbms_output.put_line('The given name is '||a);
  5* end;
SQL>/
Enter value for enter_the_name: babu
old   2: a varchar2(30):='&enter_the_name';
new   2: a varchar2(30):='babu';
The given name is babu

PL/SQL procedure successfully completed.
```

Example3

```
SQL>
  1  declare
  2  a number:=&enter_first_value;
  3  b number:=&enter_second_value;
  4  c number;
  5  begin
  6  c:=a+b;
  7  dbms_output.put_line('====================');
  8  dbms_output.put_line('The result of two values is '||c);
  9  dbms_output.put_line('====================');
 10* end;
SQL>/
Enter value for enter_first_value: 10
old   2: a number:=&enter_first_value;
new   2: a number:=10;
Enter value for enter_second_value: 12
old   3: b number:=&enter_second_value;
new   3: b number:=12;
====================================
The result of two values is 22
====================================

PL/SQL procedure successfully completed.
```

CONDITIONAL CONTROL

SQL> select * from student;

ROLLNO	NAME	ENG	TAM	MAT
500	babu	56	78	90
501	dhana	57	89	90
503	ddl	67	89	90
504	sandhya	67	89	45
569	aaaa	67	34	100
100	bbbbb	89	45	65

Syntax

1. if condition then
   sequence of statement
   end if;

2. if condition then
   sequence of statement
   else
   sequence of statement
   end if;

3. if condition then
   sequence of statement
   elsif condition then
   sequence of statement
   else
   sequence of statement
   end if;

Example

SQL>
```
 1  declare
 2  stuname student.name%type;
```

```
 3  begin
 4  select name into stuname from student where rollno = 500;
 5  if stuname = 'babu' then
 6  update student set eng = 100 where rollno = 500;
 7  else
 8  update student set eng = 200 where rollno = 500;
 9  end if;
10* end;
11  /
```

PL/SQL procedure successfully completed.

SQL> select * from student;

ROLLNO	NAME	ENG	TAM	MAT
500	babu	100	78	90
501	dhana	57	89	90
503	ddl	67	89	90
504	sandhya	67	89	45
569	aaaa	67	34	100
100	bbbbb	89	45	65

6 rows selected.

ITERATIVE CONTROL
1.SIMPLE LOOP
2.FOR LOOP
3.WHILE LOOP

1.SIMPLE LOOP
Syntax

```
loop
        sequence of statements;
end loop;
```

Example

```
SQL>
  1  declare
  2  a number := 100;
  3  begin
  4  loop
  5  a := a + 25;
  6  exit when a = 250;
  7  end loop;
  8  dbms_output.put_line('The result of a is ' || to_char(a));
  9* end;
SQL> /
The result of a is 250

PL/SQL procedure successfully completed.
```

2.WHILE LOOP

Syntax

```
        while condition
loop
        sequence of statements;
end loop;
```

Example

```
SQL>
  1  declare
  2  i number := 0;
  3  j number := 0;
  4  begin
  5  while i <= 100 loop
  6  j := j + 1;
  7  i := i + 1;
  8  end loop;
  9  dbms_output.put_line('The result of j is '||j);
 10* end;
SQL> /

PL/SQL procedure successfully completed
```

```
SQL> /
The result of j is 101
```

3.FOR LOOP

Syntax

```
        for counter in lowerbound .. upperbound
loop
        sequence of statements;
end loop;
```

Example1

```
SQL>
  1  begin
  2  for i in 1..2
  3  loop
  4  update student set eng = 200 where mat = 90;
  5  end loop;
  6* end;
SQL> /

PL/SQL procedure successfully completed.

SQL> select * from student;

   ROLLNO NAME           ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
       500 babu          200        78        90
       501 dhana         200        89        90
       503 ddl           200        89        90
       504 sandhya        67        89        45
       569 aaaa           67        34       100
       100 bbbbb          89        45        65

6 rows selected.
```

Example2

```
SQL>
  1  declare
  2  a number(5);
  3  begin
  4  for a in reverse 1..10 loop
  5  dbms_output.put_line(to_char(a));
  6  end loop;
  7* end;
SQL> /
10
9
8
7
6
5
4
3
2
1

PL/SQL procedure successfully completed.
```

SEQUENTIAL CONTROL

Example

```
SQL>
  1  declare
  2  english student.eng%type;
  3  tamil student.tam%type;
  4  begin
  5  select eng,tam into english,tamil from student where name = 'babu';
  6  if english > tamil then
  7  goto updation;
  8  end if;
  9  <<updation>>
 10  update student set mat = 200 where name = 'babu';
 11* end;
SQL> /
```

PL/SQL procedure successfully completed.

SQL> select * from student;

ROLLNO	NAME	ENG	TAM	MAT
500	babu	200	78	200
501	dhana	200	89	90
503	ddl	200	89	90
504	sandhya	67	89	45
569	aaaa	67	34	100
100	bbbbb	89	45	65

6 rows selected.

PREDIFINED EXCEPTION
Syntax

```
Begin
        sequence of statements;
exception
        when exceptionname then
        sequence of statements;
when others then /* the last exception in
     the exception handler */
sequence of statements;
end;
```

Example

SQL>

```
 1  declare
 2  english student.eng%type;
 3  tamil student.tam%type;
 4  begin
 5  select eng,tam into english,tamil from student where name = 'babula';
 6  exception
 7  when no_data_found then
 8  dbms_output.put_line(' such an item value not available');
 9* end;
```

```
SQL> /
such an item value not available

PL/SQL procedure successfully completed.
```

USER DEFINED EXCEPTION
Syntax
<exceptionname>
exception;
raise <exceptionname>;

Example1

```
SQL>
  1  declare
  2  value exception;
  3  maths student.mat%type;
  4  begin
  5  select mat into maths from student where name = 'babu';
  6  if maths < 300 then
   7    raise value;
   8    else
   9  dbms_output.put_line(' The maths value is greater than 300 ');
   9    end if;
 10  exception
 11  when value then
 12  dbms_output.put_line(' The maths value is less than 300 ');
 13* end;
 14  /
The maths value is less than 300

PL/SQL procedure successfully completed.
```

Example2

```
SQL>
  1  declare
  2  a number(3);
  3  b number(3);
  4  c number(3);
  5  first exception;
  6  second exception;
```

```
 7  third exception;
 8  begin
 9  c := &enter_choice_1_2_3;
10  if c = 1 then
11  raise first;
12  elsif c = 2 then
13  raise second;
14  elsif c = 3 then
15  raise third;
16  end if;
17  exception
18  when first then
19  dbms_output.put_line('your selection is first class');
20  when second then
21  dbms_output.put_line('your selection is second class');
22  when third then
23  dbms_output.put_line('your selection is third class');
24* end;
SQL> /
Ent`r value for enter_choice_1_2_3: 2
old   9: c := &enter_choice_1_2_3;
new   9: c := 2;
your selection is second class

PL/SQL procedure successfully completed.
```

EXCEPTION_INIT PRAGMA

Description

Oracle error number is the desired code to be associated with the named exception.

Syntax

praga exception_init(exception name, oracle error number);

Example

```
SQL>
 1  declare
 2  error1 exception;
 3  pragma exception_init(error1,-1);
```

```
  4  begin
  5  insert into student values (500,'raja',34,45,67);
  6  exception
  7  when error1 then
  8  dbms_output.put_line('duplicate item number - primary key violation');
  9* end;
 10  /
duplicate item number - primary key violation

PL/SQL procedure successfully completed.
```

RAISE_APPLICATION_ERROR
Description
Error number is any parameter between –20,000 and –20,999.
The error message parameter must be fewer than 512.
Syntax
Raise_application_error (error number, error message);

Example

```
SQL>
  1  declare
  2  tamil student.tam%type;
  3  error1 exception;
  4  begin
  5  select tam into tamil from student where name = 'babu';
  6  if tamil < 100 then
  7  raise error1;
  8  else
  9  dbms_output.put_line('Tamil is greater than 10 ');
 10  end if;
 11  exception
 12  when error1 then
 13  raise_application_error(-20001,'The tamil mark is under 10');
 14* end;
SQL> /
declare
*
```

ERROR at line 1:
ORA-20001: The tamil mark is under 10
ORA-06512: at line 13

CURSORS

Description

Oracle provides 2 types of cursors

1. Explicit cursors
2. Implicit cursors

EXPLICIT CURSORS

Description

4 types of explicit cursors are

1. %Notfound
2. %Found
3. %Rowcount
4. %Isopen

Syntax
open <cursor_name>;
fetch <cursor_name> into <column_name>;
close <cursor_name>;

Example

SQL>

```
 1  declare
 2  num student.rollno%type;
 3  cursor a is select rollno from student where name = 'ddl';
 4  begin
 5  open a;
 6  loop
 7  fetch a into num;
 8  update student set mat = 500 where rollno = num;
 9  exit when a%notfound;
10  end loop;
11  dbms_output.put_line('Table is sussessfully updated ');
12  close a;
13* end;
```

```
SQL> /
Table is sussessfully updated

PL/SQL procedure successfully completed.
```

CURSOR USING %FOUND

Example

```
SQL>
  1  declare
  2  cursor c1 is select * from student;
  3  var1 student%rowtype;
  4  begin
  5  open c1;
  6  loop
  7  fetch c1 into var1;
  8  if c1%found then
  9  update student set eng = 500 where name = 'babu';
 10  else
 11  exit when c1%notfound;
 12  dbms_output.put_line('The record is not found');
 13  end if;
 14  end loop;
 15  close c1;
 16  dbms_output.put_line('==========================');
 17  dbms_output.put_line('Record is updated ');
 18  dbms_output.put_line('==========================');
 19* end;
SQL> /
==========================
Record is updated
==========================

PL/SQL procedure successfully completed.

SQL> select * from student;

   ROLLNO NAME            ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
      500 babu            500       278       300
      501 dhana           200        89        90
```

```
503 ddl          200    89    90
504 sandhya      67     89    45
569 aaaa         67     34    100
100 bbbbb        89     45    65
```

6 rows selected.

CURSOR USING %ROWCOUNT
Example

```
SQL>
 1  declare
 2  cursor a is select * from student where name = 'babu';
 3  myrecord student%rowtype;
 4  begin
 5  open a;
 6  loop
 7  fetch a into myrecord;
 8  exit when a%notfound;
 9 dbms_output.put_line(' fetched  ' || a%rowcount || 'row from table ');
10  end loop;
11* end;
SQL> /
fetched  1 row from  table

PL/SQL procedure successfully completed.
```

CURSOR %ISOPEN
Example

```
SQL>
 1  declare
 2  cursor a is select * from student;
 3  begin
 4  if not a%isopen then
 5  dbms_output.put_line(' The cursor is yet to be opened ');
 6  end if;
 7  open a;
 8  if a%isopen then
 9  dbms_output.put_line(' The cursor is now open');
10  end if;
11  close a;
```

```
 12* end;
 13  /
```
The cursor is yet to be opened
The cursor is now open

PL/SQL procedure successfully completed.

IMPLICIT CURSOR

Description

4 types of Implicit cursors are

1. Sql%notfound
2. sql%found
3. Sql%rowcount
4. Sql%isopen

CURSOR SQL%NOTFOUND

Example

SQL>

```
  1  begin
  2  delete from student where name = 'babula';
  3  if sql%notfound then
  4  dbms_output.put_line(' value not found');
  5  else
  6  dbms_output.put_line(' value found and deleted');
  7  end if;
  8* end;
SQL> /
```
value not found

PL/SQL procedure successfully completed.

CURSOR SQL%ROWCOUNT

Example

SQL>

```
  1  declare
  2  a number := &enter_roll_number;
  3  b student%rowtype;
  4  begin
  5  update student set eng = 600 where rollno = a;
```

```
  6  if sql%rowcount>1 then
  7  dbms_output.put_line(sql%rowcount || ' row updated');
  8  elsif sql%rowcount=1 then
  9  dbms_output.put_line(sql%rowcount || ' row updated');
 10  elsif sql%rowcount<1 then
 11  dbms_output.put_line(sql%rowcount || ' row updated');
 12  end if;
 13* end;
SQL> /
Enter value for enter_roll_number: 504
old   2: a number := &enter_roll_number;
new   2: a number := 504;
1 row updated

PL/SQL procedure successfully completed.

SQL> select * from student;

   ROLLNO NAME           ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
       500 babu           500        278        300
       501 dhana          200         89         90
       503 ddl            200         89         90
       504 sandhya        600         89         45
       569 aaaa            67         34        100
       100 bbbbb           89         45         65

6 rows selected.
```

CURSOR %FOR LOOP

Syntax

for <record_name> in <cursor_name> loop

sequence of statements;

end loop;

Example

TABLE

```
SQL> select * from student_detail;

NAME           ENG       MAT
```

---------- ---------- ----------
raja 45 67
babu 45 78
raju 78 23

SQL> select * from student;

ROLLNO NAME ENG TAM MAT
---------- ---------- ---------- ---------- ----------
500 babu 200 78 300
501 dhana 200 89 90
503 ddl 200 89 90
504 sandhya 67 89 45
569 aaaa 67 34 100
100 bbbbb 89 45 65

CODE

```
SQL>
 1  declare
 2  cursor a is select name from student where tam = 78;
 3  mas_rec student%rowtype;
 4  begin
 5  for mas_rec in a loop
 6  delete from student_detail where name = mas_rec.name;
 7  end loop;
 8  dbms_output.put_line('details  table record has been deleted');
 9* end;
SQL> /
details  table record has been deleted
```

OUTPUT

SQL> select * from student_detail;

NAME ENG MAT
---------- ---------- ----------
raja 45 67
raju 78 23

REF CURSORS
Description
When the Cursor variable has a return type is known as constrained or strong cursor.
When the Cursor variable has a no return type is known as unconstrained or weak cursor.
Syntax
type type_name is ref cursor return return_type;

Example1

```
SQL>
  1  declare
  2  type r1_cur is ref cursor;
  3  var1 r1_cur;
  4  num number(3);
  5  no number(3);
  6  begin
  7  no := &enter_the_no;
  8  if no = 10 then
  9  open var1 for
 10  select mat from student where name = 'babu';
 11  fetch var1 into num;
 12  dbms_output.put_line('The maths mark is ' || num);
 13  close var1;
 14  else
 15  open var1 for
 16  select eng from student where rollno = 501;
 17  loop
 18  fetch var1 into num;
 19  exit when var1%notfound;
 20  dbms_output.put_line('The english mark for student is   ' || num);
 21  end loop;
 22  close var1;
 23  end if;
 24* end;
 25  /
Enter value for enter_the_no: 10
old   7: no := &enter_the_no;
new   7: no := 10;
```

```
The maths mark is 300

PL/SQL procedure successfully completed.

SQL> /
Enter value for enter_the_no: 12
old   7: no := &enter_the_no;
new   7: no := 12;
The english mark for student is   200

PL/SQL procedure successfully completed.
```

Example2

```
SQL>
  1  declare
  2  type marks is record (
  3  rollno1 number(4),
  4  name1 varchar2(10),
  5  eng1 number(3),
  6  tam1 number(3),
  7  mat1 number(3));
  8  type stumark is ref cursor return student%rowtype;
  9  var1 stumark;
 10  var2 marks;
 11  begin
 12  open var1 for
 13  select * from student where name = 'babu';
 14  loop
 15  fetch var1 into var2;
 16  exit when var1%notfound;
 17  dbms_output.put_line('The values are  '||var2.rollno1||'   '
 18  ||var2.name1||'   '||var2.eng1||'    '||var2.tam1||'   '||var2.mat1);
 19  end loop;
 20  close var1;
 21* end;
 22  /
The values are  500   babu   200    78   300

PL/SQL procedure successfully completed.
```

SUBPROGRAMS

1.PROCEDURES(IN,OUT,INOUT)

2.FUNCTIONS

PROCEDURES

SQL> select * from student;

ROLLNO	NAME	ENG	TAM	MAT
500	babu	200	78	300
501	dhana	200	89	90
503	ddl	200	89	90
504	sandhya	67	89	45
569	aaaa	67	34	100
100	bbbbb	89	45	65

Syntax

1. Create or replace procedure <procedure_name> [parameter list] is
   <local declarations>;
   begin
   (executable statements)
   [exception]
   (exception handlers)
   end;
2. exec <procedure_name> (parameters);

Example

SQL>

```
 1  create or replace procedure marks (na varchar2) is
 2  eng1 number;
 3  tam1 number;
 4  begin
 5  select eng,tam into eng1,tam1 from student where name = na;
 6  if tam1 < eng1 then
 7  update student set tam = eng + tam where name = na;
 8  else
```

```
 9  dbms_output.put_line('The tamil mark is greater than english');
10  end if;
11  exception
12  when no_data_found then
13  dbms_output.put_line('no data returned');
14* end;
15  /
```

Procedure created.

```
SQL> exec marks('raw');
```

no data returned

PL/SQL procedure successfully completed.

```
SQL> exec marks('babu');
```

PL/SQL procedure successfully completed.

```
SQL> select * from student;
```

ROLLNO	NAME	ENG	TAM	MAT
500	babu	200	278	300
501	dhana	200	89	90
503	ddl	200	89	90
504	sandhya	67	89	45
569	aaaa	67	34	100
100	bbbbb	89	45	65

6 rows selected.

```
SQL> exec marks('babu');
```

The tamil mark is greater than english

PL/SQL procedure successfully completed.

PROCEDURE(IN PARAMETER)

Description

Used to pass values to the subprogram. Its acts like a constant and therefore it cannot be assigned a value.

Example

```
SQL>
  1  create or replace procedure marks_in(a in varchar2) is
  2  eng1 number(3);
  3  tam1 number(3);
  4  begin
  5  select eng,tam into eng1,tam1 from student where name = a;
  6  if tam1 > 100 then
  7  dbms_output.put_line('you are getting good marks mr. '||a);
  8  else
  9  dbms_output.put_line('yor are getting poor marks Mr.  '||a);
 10  end if;
 11* end;
 12  /

Procedure created.

SQL> exec marks_in('ddl');
yor are getting poor marks Mr.  ddl

PL/SQL procedure successfully completed.

SQL> exec marks_in('babu');
you are getting good marks mr. babu

PL/SQL procedure successfully completed.
```

PROCEDURE(OUT PARAMETER)

Description
Used to return values to the caller of a subprogram. Since the initial value for an out parameter is undefined. Its value be assigned to another variable.(executed output in another program).

Example

```
SQL>
  1  create or replace procedure marks_out(a in varchar2, b out number) is
  2  mat1 number;
  3  begin
  4  select mat into mat1 from student where name = a;
  5  if mat1 < 100 then
  6  b:=200;
  7  end if;
  8* end;
SQL> /

Procedure created.

SQL>
  1  declare
  2  a varchar2(5);
  3  b number;
  4  begin
  5  marks_out('ddl',b);
  6  dbms_output.put_line('the value of b is  '||to_char(b));
  7* end;
SQL> /
the value of b is  200
```

PL/SQL procedure successfully completed.

PROCEDURE(IN OUT PARAMETER)

Description

Used to pass initial values to the subprogram.It also returns updated values to the caller.(run in another program given below).

Example +6

```
SQL>
  1  create or replace procedure marks_inout (a in varchar2,b in out number) is
  2  eng1 number;
  3  tam1 number;
  4  mat1 number;
  5  begin
  6 select eng, tam,mat into eng1,tam1,mat1 from student where name = a;
  7  if eng1 < tam1 then
  8  b := mat1;
  9  end if;
 10* end;

SQL> /
Procedure created.
SQL>
  1 declare
  2 a varchar2(5);
  3 b varchar2(5);
  4 begin
  5 marks_inout('babu',b);
  6 dbms_output.put_line('The maths mark is '||b);
  7* end;
SQL> /
The maths mark is 300

PL/SQL procedure successfully completed.
```

FUNCTIONS
Description
Function is a subprogram that computes a value.
Syntax
create or replace function <function_name> [argument]
return datatype is
(local declaration)
begin
(executable statements)
[exception statements]
(exception handlers)
end;

Example

```
SQL>
  1  create or replace function marks_fun(a varchar2)
  2  return number is
  3  eng1 number;
  4  tam1 number;
  5  mat1 number;
  6  args number;
  7  begin
  8  select eng,tam,mat into eng1,tam1,mat1 from student where name = a;
  9  if (eng1+tam1) < mat1 then
 10  args := mat1;
 11  return args;
 12  else
 13  args := (eng1+tam1);
 14  return args;
 15  end if;
 16* end;
SQL> /

Function created.

SQL>
  1  declare
  2  aa varchar2(10);
  3  b number;
  4  begin
  5  aa := &enter_name;
```

```
  6  b := marks_fun(aa);
  7  dbms_output.put_line(' The value retured is  '||(b));
  8* end;
SQL> /
Enter value for enter_name: 'ddl'
old   5: aa := &enter_name;
new   5: aa := 'ddl';
The value retured is  289

PL/SQL procedure successfully completed.
```

PACKAGES

Description

A package is a database object, which is an encapsulation of related PL/SQL types, subprograms,cursors,exceptions,variables and constants.

It consists of two parts, a specification and a body. In specification we can declare types, variables, constants,exceptions,cursors and subprograms.

A package body implements cursors, subprograms defined in the package specification.

Syntax

```
1. create package <package_name> is <declarations>
   begin
   (executable statements)
    end   [package_name];

2. create package body <package_name> is
<declarations>
   begin
   (executable statements)
   end    [body_name];
```

Example (spcication & body program)

```
SQL> select * from student;
```

```
   ROLLNO NAME           ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
       500 babu            200       278       300
       501 dhana           200        89        90
       503 ddl             200        89        90
       504 sandhya          67        89        45
       569 aaaa             67        34       100
       100 bbbbb            89        45        65

SQL> select * from student_detail;

NAME           ENG       MAT
---------- ---------- ----------
raja            45        67
raju            78        23


SQL>
  1  create or replace package babula is
  2  procedure mark_pro (a varchar2);
  3  function mark_fun (b number) return number;
  4* end babula;
SQL> /

Package created.

SQL>
  1  create or replace package body babula as
  2  procedure mark_pro (a varchar2) is
  3  eng1 number(3);
  4  begin
  5  select eng into eng1 from student where name = a;
  6  if eng1 = 90 then
  7  dbms_output.put_line('english is equal to 90');
  8  else
  9  dbms_output.put_line('english is not equal to 90');
 10  end if;
 11  end mark_pro;
 12  function mark_fun (b number) return number is
```

```
 13  mat1 number(3);
 14  begin
 15  select mat into mat1 from student_detail where eng = b;
 16  if mat1 = 100 then
 17  return mat1;
 18  else
 19  return 900;
 20  end if;
 21  end mark_fun;
 22* end babula;
SQL> /

Package body created.

SQL> exec babula.mark_pro('babu');
english is not equal to 90

PL/SQL procedure successfully completed.

SQL>
  1  declare
  2  a number(3);
  3  b number(3);
  4  begin
  5  b := babula.mark_fun(45);
  6  dbms_output.put_line(' The value is  '||b);
  7* end;
SQL> /
The value is  900

PL/SQL procedure successfully completed.
```

OVERLOADING OF PROCEDURES

Syntax

```
Create or replace package degree_pack is
procedure bachelor_deg (name char, course char);
procedure bachelor_deg (name char, course number);
end degree_pack;
create or replace package body degree_pack is
procedure bachelor_deg (name char, course char) is
begin
-----
end [bachelor_deg];
procedure bachelor_deg (name char, course number) is
begin
------
end [bachelor_deg];
end [bachelor_pack];
```

example

```
create or replace package pa1 is
procedure pp (na varchar2);
procedure pp (na varchar2,e number);
end pa1;

create or replace package body pa1 as
procedure pp (na varchar2) is
eng1 number(3);
begin
select eng into eng1 from student where name = na;
dbms_output.put_line(eng1);

end pp;
procedure pp (na varchar2,e number) is
tam1 number(3);
begin
select tam into tam1 from student where name=na and eng=e;
dbms_output.put_line(tam1);

end pp;
end pa1;
```

example

```
create or replace package fu1 is
function ff(na varchar2) return number;
function ff(na varchar2,e number) return number;
end fu1;

create or replace package body fu1 as
function ff(na varchar2) return number is
eng1 number(3);
begin
select eng into eng1 from student where name = na;
return(eng1);
end ff;
function ff(na varchar2,e number) return number is
tam1 number(3);
begin
select tam into tam1 from student where name = na and eng = e;
return(tam1);
end ff;
end fu1;
```

output

```
SQL> select fu1.ff('babu') from dual;

FU1.FF('BABU')
--------------
          90

SQL> select fu1.ff('babu',90) from dual;

FU1.FF('BABU',90)
-----------------
           900

SQL> select fu1.ff(name) from student;

FU1.FF(NAME)
```

```
------------
        90
       200
       200
        67
        67
```

CURSORS IN PACKAGES

Example (spcication & body program)

```
SQL>
  1  create or replace package babu10 is
  2  cursor mark_cur return student%rowtype;
  3  procedure mark2 (a varchar2);
  4* end babu10;
SQL> /

Package created.

SQL>
  1  create or replace package body babu10 as
  2  cursor mark_cur return student%rowtype is
  3  select * from student;
  4  procedure mark2(a varchar2) is
  5  var1 student%rowtype;
  6  begin
  7  open mark_cur;
  8  loop
  9  fetch mark_cur into var1;
 10  exit when mark_cur%notfound;
 11  dbms_output.put_line('The returned values are   ' ||var1.rollno);
 12  dbms_output.put_line('The returned values are   ' ||var1.eng);
 13  end loop;
 14  end mark2;
 15* end babu10;
SQL> /

Package body created.

SQL> exec babu10.mark2('babu');
```

```
The returned values are   500
The returned values are   200
The returned values are   501
The returned values are   200
The returned values are   503
The returned values are   200
The returned values are   504
The returned values are   67
The returned values are   569
The returned values are   67
The returned values are   100
The returned values are   89
```

PL/SQL procedure successfully completed.

DATABASE TRIGGERS

Syntax

Create or replace Trigger <Trigger_name>
[before/after] [insert/update/delete] on <table-name>
[for each statement/for each row] [when <condition>];

Example1

```
SQL> create table stud as select * from student;
Table created.
SQL> select * from stud;

   ROLLNO NAME           ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
       500 babu            200       278       300
       501 dhana           200        89        90
       503 ddl             200        89        90
       504 sandhya          67        89        45
       569 aaaa             67        34       100
       100 bbbbb            89        45        65
```

```
6 rows selected.

SQL>
  1  create or replace trigger mark12
  2  before delete on stud
  3  begin
  4  raise_application_error (-20001,'Not permisson to delete the record');
  5* end;
SQL> /

Trigger created.

SQL> delete from stud where mat=300;
delete from stud where mat=300
            *
ERROR at line 1:
ORA-20001: Not permisson to delete the record
ORA-06512: at "SCOTT.MARK12", line 2
ORA-04088: error during execution of trigger 'SCOTT.MARK12'
Example2
SQL>
  1  create or replace trigger mark13
  2  before update on stud for each row
  3  begin
  4  if :new.eng < :old.eng then
  5  raise_application_error(-20001,'The update value of english is less than old
value');
  6  end if;
  7* end;
SQL> /

Trigger created.

SQL> update stud set eng = 150 where name = 'babu';
update stud set eng = 150 where name = 'babu'
       *
ERROR at line 1:
ORA-20001: The update value of english is less than old value
ORA-06512: at "SCOTT.MARK13", line 3
ORA-04088: error during execution of trigger 'SCOTT.MARK13'
```

```
SQL> update stud set eng = 250 where name = 'babu';

1 row updated.

SQL> select * from stud;

   ROLLNO NAME            ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
       500 babu            250       278       300
       501 dhana           200        89        90
       503 ddl             200        89        90
       504 sandhya          67        89        45
       569 aaaa             67        34       100
       100 bbbbb            89        45        65

6 rows selected.

Example3
SQL>
  1  create or replace trigger mark13
  2  before update on stud for each row
  3  begin
  4  update stud set eng = 100 where name = 'babu';
  5* end;
SQL> /

Trigger created.

SQL> update stud set eng = 200 where tam = 45;
update stud set eng = 200 where tam = 45
       *
ERROR at line 1:
ORA-04091: table SCOTT.STUD is mutating, trigger/function may not see it
ORA-06512: at "SCOTT.MARK13", line 2
ORA-04088: error during execution of trigger 'SCOTT.MARK13'

SQL> update stud set eng = 100 where name = 'babu';
update stud set eng = 100 where name = 'babu'
```

```
      *
ERROR at line 1:
ORA-04091: table SCOTT.STUD is mutating, trigger/function may not see it
ORA-06512: at "SCOTT.MARK13", line 2
ORA-04088: error during execution of trigger 'SCOTT.MARK13'
```

Example

```
SQL>
  1  CREATE OR REPLACE TRIGGER T3 BEFORE INSERT OR UPDATE ON bab
  2  FOR EACH ROW
  3  BEGIN
  4  IF :NEW.eng>100 AND :NEW.name<>'dhana' THEN
  5  RAISE_APPLICATION_ERROR(-20004,'NOT POSSIBLE');
  6  END IF;
  7* END;
SQL> /

Trigger created.
SQL> update bab set eng = 102 where name = 'ddl';
update bab set eng = 102 where name = 'ddl'
       *
ERROR at line 1:
ORA-20004: NOT POSSIBLE
ORA-06512: at "SCOTT.T3", line 3
ORA-04088: error during execution of trigger 'SCOTT.T3'

SQL> insert into bab(name,eng) values('dhana',122);

1 row created.

SQL> insert into bab(name,eng) values('ggg',122);
insert into bab(name,eng) values('ggg',122)
            *
ERROR at line 1:
ORA-20004: NOT POSSIBLE
ORA-06512: at "SCOTT.T3", line 3
ORA-04088: error during execution of trigger 'SCOTT.T3'
```

Example

```
SQL>
```

```
  1  CREATE OR REPLACE TRIGGER T6 BEFORE INSERT OR UPDATE ON bab
  2  FOR EACH ROW
  3  DECLARE
  4  A NUMBER;
  5  BEGIN
  6  SELECT eng INTO A FROM stud WHERE eng=:NEW.eng;
  7  EXCEPTION
  8  WHEN NO_DATA_FOUND THEN
  9  RAISE_APPLICATION_ERROR(-20004,'PARENT KEY NOT FOUND');
 10* END;
SQL> /

Trigger created.

SQL> update bab set eng = 67 where name = 'ddl';
update bab set eng = 67 where name = 'ddl'
       *
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at "SCOTT.T6", line 4
ORA-04088: error during execution of trigger 'SCOTT.T6'


SQL> update bab set eng = 250 where name = 'ddl';

1 row updated.

SQL> update bab set eng = 150 where name = 'ddl';
update bab set eng = 150 where name = 'ddl'
       *
ERROR at line 1:
ORA-20004: PARENT KEY NOT FOUND
ORA-06512: at "SCOTT.T6", line 7
ORA-04088: error during execution of trigger 'SCOTT.T6'

Example
SQL>
  1  CREATE OR REPLACE TRIGGER T9 BEFORE DELETE ON bab FOR EACH ROW
  2  DECLARE CURSOR C1 IS SELECT eng FROM stud WHERE
  3  eng=:OLD.eng;
```

```
  4  A NUMBER;
  5  BEGIN
  6  OPEN C1;
  7  FETCH C1 INTO A;
  8  IF C1%FOUND THEN
  9  RAISE_APPLICATION_ERROR(-20111,'CHILD RECORDS FOUND');
 10  END IF;
 11  CLOSE C1;
 12  EXCEPTION WHEN NO_DATA_FOUND THEN
 13  DBMS_OUTPUT.PUT_LINE('DELETING RECORDS...');
 14* END;
SQL> /

Trigger created.

SQL> select * from bab;

NAME                 ENG       TAM       MAT ODATE
---------------- ---------- ---------- ---------- ---------
babu                 200       100        90 12-MAR-98
ddl                  250        45        67 12-MAR-99
sandhya               45        67        89 12-MAY-99
mani                  67        89        56 12-AUG-96
kavi                  56        78        90 12-MAR-98
                      90        90

6 rows selected.

SQL> select * from stud;

    ROLLNO NAME            ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
       500 babu            250       278       300
       501 dhana           200        89        90
       503 ddl             200        89        90
       504 sandhya          67        89        45
       569 aaaa             67        34       100
       100 bbbbb            89        45        65

6 rows selected.
```

C.Sivanantham

```
SQL> delete from bab where eng = 67;
delete from bab where eng = 67
            *
ERROR at line 1:
ORA-20111: CHILD RECORDS FOUND
ORA-06512: at "SCOTT.T9", line 8
ORA-04088: error during execution of trigger 'SCOTT.T9'


SQL> delete from bab where eng = 90;

1 row deleted.
```

ENABLING,DISABLING AND DROP TRIGGERS

Syntax

1. alter trigger <trigger name> disable;
2. alter table <table_name> disable <trigger name>;{apply only too many triggers on the table}.
3. alter table <table name> disable all triggers;
4. alter table <table name> enable <trigger name>;
5. alter table <table name> enable all triggers;
6. drop trigger <trigger name>;

Example

```
SQL> alter trigger mark12 disable;
Trigger altered.

SQL> alter trigger mark12 enable;
Trigger altered.

SQL> alter table stud disable all triggers;
Table altered.

SQL> alter table stud enable all triggers;
Table altered.

SQL> drop trigger mark12;
```

Trigger dropped.

USING BFILE
Description
Let us add another column to the table stud named photo, which is a bfile column.

Example
SQL>alter table stud add column photo bfile;
Description
Let us create an image file in paintbrush and name the file as “emp1.bmp”
This is assumed to reside in directory “C:”.Let us create an alias for this directoy as

Example
SQL> create directory bdir as ‘C:\’;
Directory created

SQL>insert into stud (rollno,photo) values(508,BFILENAME(‘BDIR’,’EMP1.BMP));
1 row created

```
SQL> declare
nfile bfile;
length interger;
begin
select photo into nfile from stud where rollno = 508;
length := dbms_lob.getlength(nfile);
if length is null then
dbms_output.put_line(‘LOB is null ‘);
else
dbms_output.put_line(‘The length is ‘|| length);
end if;
end;

SQL>/
The length is 308278
PL/SQL procedure successfully completed.
```

PL/SQL TABLES

Syntax
declare
type <name of type> is table of <col_def>
index by binary_integer;

<PL/SQL_Tablename> <typename>;

PL/SQL TABLES

Syntax
declare
type <name of type> is table of <col_def>
index by binary_integer;

<PL/SQL_Tablename> <typename>;

Example
SQL> select * from stud;

ROLLNO	NAME	ENG	TAM	MAT
500	babu	250	278	300
501	dhana	200	89	90
503	ddl	200	89	90
504	sandhya	67	89	45
569	aaaa	67	34	100
100	bbbbb	89	45	65

SQL>

```
 1  declare
 2  type stutype is table of stud.name%type
 3  index by binary_integer;
 4  list stutype;
 5  name1 varchar2(10);
 6  i binary_integer := 0;
 7  begin
 8  name1 := '&enter_the_name';
 9  list(i) := name1;
10  if list(i) = 'aaa' then
```

```
 11  update stud set name = 'asha' where eng = 89;
 12  dbms_output.put_line(' updation performed  ');
 13  end if;
 14* end;
SQL> /
Enter value for enter_the_name: aaa
old   8: name1 := '&enter_the_name';
new   8: name1 := 'aaa';
updation performed

PL/SQL procedure successfully completed.

SQL> select * from stud;

   ROLLNO NAME            ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
       500 babu           250       278       300
       501 dhana          200        89        90
       503 ddl            200        89        90
       504 sandhya         67        89        45
       569 aaaa            67        34       100
       100 asha            89        45        65

6 rows selected.

MAINTAINING ROW COUNT IN PL/SQL TABLES

SQL>
  1  declare
  2  type stutype is table of number(4) index by
  3  binary_integer;
  4  count stutype;
  5  i binary_integer := 0;
  6  begin
  7  loop
  8  i := i + 1;
  9  count(i) := i;
 10  exit when i = 10;
 11  end loop;
 12  dbms_output.put_line(' Total number of rows are  '||to_char(i));
```

```
 13* end;
SQL> /
Total number of rows are  10

PL/SQL procedure successfully completed.
```

INSERTING AND FETCHING ROWS USING A PL/SQL TABLE

Example1

```
SQL>
  1  declare
  2  type stutype1 is table of stud.tam%type
  3  index by binary_integer;
  4  count1 stutype1;
  5  i binary_integer := 0;
  6  begin
  7  while (i < 1)
  8  loop
  9  count1(i) := 85;
 10  insert into stud(tam) values(count1(i));
 11  i := i +1;
 12  end loop;
 13  dbms_output.put_line('The given value is inserted');
 14* end;
SQL> /
The given value is inserted

PL/SQL procedure successfully completed.

SQL> select * from stud;

    ROLLNO NAME            ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
       500 babu            250        278        300
       501 dhana           200         89         90
       503 ddl             200         89         90
       504 sandhya          67         89         45
       569 aaaa             67         34        100
       100 bbbbb            89         45         65
                            85
```

7 rows selected.

Example2

```
SQL>
  1  declare
  2  stu_rec stud%rowtype;
  3  type stutype is table of varchar2(15)
  4  index by binary_integer;
  5  count1 stutype;
  6  i binary_integer := 0;
  7  begin
  8  for stu_rec in(select name from stud)
  9  loop
 10  i := i + 1;
 11  count1(i) := stu_rec.name;
 12  dbms_output.put_line(' The name is '||i||'.'||count1(i));
 13  end loop;
 14* end;
SQL> /
The name is 1.babu
The name is 2.dhana
The name is 3.ddl
The name is 4.sandhya
The name is 5.aaaa
The name is 6.asha

PL/SQL procedure successfully completed.
```

DELETING ROWS FROM A PL/SQL TABLE

Description

PL/SQL tables are not stored in the database hence the delete and drop commands cannot be executed.

Example

```
SQL>
  1  declare
  2  type stutype is table of number(5)
  3  index by binary_integer;
  4  mark stutype;
```

```
  5  begin
  6  for i in 1..10
  7  loop
  8  mark(i) := i;
  9  end loop;
 10  mark.delete;
 11  dbms_output.put_line(' Rows of the table are deleted ');
 12* end;
SQL> /
Rows of the table are deleted

PL/SQL procedure successfully completed.
```

RECORDS
Syntax
Type <Type name> is Record (column definitions…….);

<PL/SQL record type> <type name>;

Example1

Method 1

```
SQL>
  1  declare
  2  type stutype is record(eng1 number(3),tam1 number(3));
  3  rec2 stutype;
  4  rec3 stutype;
  5  begin
  6  rec2.eng1 := 100;
  7  rec2.tam1 := 200;
  8  rec3 := rec2;
  9  dbms_output.put_line('The english mark is '||rec3.eng1);
 10  dbms_output.put_line('The tamil mark is '||rec3.tam1);
 11* end;
SQL> /
The english mark is 100
The tamil mark is 200
```

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

Method 2

```
SQL>
 1  declare
 2  type stutype is record(eng1 number(3) := 30,tam1 stud.mat%type := 50);
 3  rec2 stutype;
 4  rec3 stutype;
 5  begin
 6  rec2.eng1 := 100;
 7  rec2.tam1 := 200;
 8  rec3 := rec2;
 9  dbms_output.put_line('The english mark is '||rec3.eng1);
10  dbms_output.put_line('The tamil mark is '||rec3.tam1);
11* end;
SQL> /
The english mark is 100
The tamil mark is 200

PL/SQL procedure successfully completed.
```

Method 3

```
SQL>
 1  declare
 2  type stutype is record(eng1 number(3) := 30,tam1 stud.mat%type := 50);
 3  rec2 stutype;
 4  rec3 stutype;
 5  begin
 6  rec2.tam1 := 200;
  7  rec3 := rec2;
  8  dbms_output.put_line('The english mark is '||rec3.eng1);
  9  dbms_output.put_line('The tamil mark is '||rec3.tam1);
 10* end;
 11  /
The english mark is 30
The tamil mark is 200

PL/SQL procedure successfully completed.
```

Example2

```
SQL> select * from stud;

   ROLLNO NAME           ENG       TAM       MAT
---------- ---------- ---------- ---------- ----------
       500 babu            250       278       300
       501 dhana           200        89        90
       503 ddl             200        89        90
       504 sandhya          67        89        45
       569 aaaa             67        34       100
       100 bbbbb            89        45        65

6 rows selected.

SQL>
  1  declare
  2  type stutype is record(tam1 number(3),eng1 number(3));
  3  rec1 stutype;
  4  begin
  5  select eng,tam into rec1 from stud where name = 'babu';
  6  dbms_output.put_line('The english mark is '||rec1.eng1);
  7  dbms_output.put_line('The tamil mark is '||rec1.tam1);
  8* end;
  9  /
The english mark is 278
The tamil mark is 250

PL/SQL procedure successfully completed.
Example3
SQL>
  1  declare
  2  type stutype is table of number(5)
  3  index by binary_integer;
  4  rec1 stutype;
  5  begin
  6  for i in 1..5
  7  loop
  8  rec1(i) := i;
  9  dbms_output.put_line('inserted values are  '||rec1(i));
 10  end loop;
 11  rec1.delete(3);
```

```
 12  dbms_output.put_line('Third record is deleted ');
 13* end;
SQL> /
inserted values are  1
inserted values are  2
inserted values are  3
inserted values are  4
inserted values are  5
Third record is deleted

PL/SQL procedure successfully completed.
```

NOTE:

The row number has to be passed as an argument to the delete opertor and only that particular row will be deleted.

NESTED TABLES AND VARYING ARRAYS

Syntax

Type table_name is table of table_type;

Example1

```
SQL>
  1  declare
  2  type stutype is table of number;
  3  i integer;
  4  mark stutype := stutype(10,20,30);
  5  begin
  6  mark(1) := 100;
  7  mark(2) := 200;
  8  for i in 1..mark.count loop
  9  dbms_output.put_line(' Value of ('||i||') element in mark is '||mark(i));
 10  end loop;
 11* end;
SQL> /
Value of (1) element in mark is 100
Value of (2) element in mark is 200
Value of (3) element in mark is 30

PL/SQL procedure successfully completed.
```

Example2

```
SQL>
  1  declare
  2  type stutype is table of number;
  3  mark stutype;
  4  mark1 stutype := stutype( );
  5  begin
  6  if mark is null then
  7  dbms_output.put_line(' mark is null ');
  8  else
  9  dbms_output.put_line(' mark is not null ');
 10  end if;
 11  if mark1 is null then
 12  dbms_output.put_line(' mark1 is null ');
 13  else
 14  dbms_output.put_line(' mark1 is not null ');
 15  end if;
 16* end;
SQL> /
mark is null
mark1 is not null

PL/SQL procedure successfully completed.
```

NESTED TABLES IN THE DATABASE

Description

Let the table contain a structure(stud1) as shown below:

column	Type
Rollno	number(4)
Name	varchar2(10)
Eng	number(3)
Tam	number(3)
Mat	number(3)

Let us we create a type that is based on eng, tam and mat.

Example1

```
SQL> create type mark as object (eng number(3),tam number(3),mat number(3));
SQL>/
Type created.

SQL> create type mark1 as table of mark;
SQL>/
Type created.

SQL> create table stud1 (rollno number(4), name varchar2(10), mk mark1)
nested table mk store as mark1_tab
SQL>/
Table created

SQL>
  1  declare
  2  var1 mark1 := mark1(mark(45,56,78));
  3  begin
  4  insert into stud1 values (505,'babu',mark1(mark(68,78,88),mark(45,55,65),
  5  mark(12,22,33)));
  6  insert into stud1 values (506,'ddl',var1);
  7* end;
SQL> /

PL/SQL procedure successfully completed.

SQL> select * from stud1;

    ROLLNO NAME
---------- ----------
MK(ENG, TAM, MAT)
------------------------------------------------------------
      505 babu
MARK1(MARK(68, 78, 88), MARK(45, 55, 65), MARK(12, 22, 33))

      506 ddl
MARK1(MARK(45, 56, 78))
```

Example2

```
SQL>
```

```
  1  declare
  2  var1 mark1 := mark1(mark(70,80,90),mark(50,60,70));
  3  begin
  4  update stud1 set mk = var1 where name = 'babu';
  5* end;
SQL> /

PL/SQL procedure successfully completed.

SQL> select * from stud1;

   ROLLNO NAME
---------- ----------
MK(ENG, TAM, MAT)
-----------------------------------------------------------
      505 babu
MARK1(MARK(70, 80, 90), MARK(50, 60, 70))

      506 ddl
MARK1(MARK(45, 56, 78))
```

Example3

```
SQL>
  1  begin
  2  delete from stud1 where name = 'ddl';
  3* end;
SQL> /

PL/SQL procedure successfully completed.

SQL> select * from stud1;

   ROLLNO NAME
---------- ----------
MK(ENG, TAM, MAT)
-----------------------------------------
      505 babu
MARK1(MARK(70, 80, 90), MARK(50, 60, 70))
```

Example4

```
SQL> select * from stud1;

   ROLLNO NAME
---------- ----------
MK(ENG, TAM, MAT)
--------------------------------------------------------
      505 babu
MARK1(MARK(68, 78, 88), MARK(45, 55, 65), MARK(12, 22, 33))

      506 ddl
MARK1(MARK(45, 56, 78))

SQL>
  1  declare
  2  var1 stud1.mk%type;
  3  var2 stud1.name%type;
  4  cursor m_cur is select name, mk from stud1 where name = 'babu';
  5  begin
  6  open m_cur;
  7  loop
  8  fetch m_cur into var2,var1;
  9  exit when m_cur%notfound;
 10  dbms_output.put_line('The name selected are '||var2);
 11  for i in 1..var1.count
 12  loop
 13  dbms_output.put_line('The nested table values are '||var1(i).eng ||
 14  '  '||var1(i).tam|| '   ' ||var1(i).mat);
 15  end loop;
 16  end loop;
 17* end;
SQL> /
The name selected are babu
The nested table values are 68  78   88
The nested table values are 45  55   65
The nested table values are 12  22   33

PL/SQL procedure successfully completed.
```

VARRAYS

Syntax

Type type_name is varray (maximum_size) of element_type (not null);

Example1

```
SQL>
  1  declare
  2  type eng1 is varray(5) of number(3);
  3  type tam1 is varray(5) of number(3);
  4  type mat1 is varray(5) of number(3);
  5  var1 eng1;
  6  var2 tam1 := tam1(10,20);
  7  begin
  8  if var1 is null then
  9  dbms_output.put_line('var1 contains null values');
 10  end if;
 11  if var2 is null then
 12  dbms_output.put_line('var2 contains null values');
 13  else
 14  dbms_output.put_line('var2 contains not null values');
 15  end if;
 16* end;
SQL> /
var1 contains null values
var2 contains not null values

PL/SQL procedure successfully completed.
```

Example2

```
SQL>
  1  declare
  2  cursor c1 is select * from stud1;
  3  var1 c1%rowtype;
  4  begin
  5  for var1 in c1 loop
  6  dbms_output.put_line (' The output values are  '||var1.rollno);
  7  end loop;
  8* end;
SQL> /
The output values are  505
The output values are  506
PL/SQL procedure successfully completed.
```

COLLECTIONS IN NESTED TABLES AND VARYING ARRAYS

Description

The collection methods are Exists(X), Count, Limit, First, Last,Prior(X),Next(X),Extend(x,y),Trim(x),Delete.

Syntax

Example1

```
SQL>
  1  declare
  2  type colors is table of varchar2(10);
  3  c colors := colors('violet','indigo','blue','green','yellow');
  4  i integer;
  5  begin
  6  i := c.count;
  7  dbms_output.put_line('No of currently available colors using COUNT is '||i);
  8  dbms_output.put_line('-----------------------------------------');
  9  i := c.first;
 10  dbms_output.put_line('value of element('||i||') using FIRST is '||c(i));
 11  dbms_output.put_line('-----------------------------------------');
 12  i := c.next(i);
 13  dbms_output.put_line('value of element('||i||') using NEXT is '||c(i));
 14  dbms_output.put_line('-----------------------------------------');
 15  i := c.last;
 16  dbms_output.put_line('value of element('||i||') using LAST is '||c(i));
 17  dbms_output.put_line('-----------------------------------------');
 18  i := c.prior(i);
 19  dbms_output.put_line('value of element('||i||') using PRIOR is '||c(i));
 20  dbms_output.put_line('-----------------------------------------');
 21  dbms_output.put_line('value of element 3 using index number is '||c(3));
 22  dbms_output.put_line('-----------------------------------------');
 23  dbms_output.put_line('prior element of element 2 using index number is
'||c(c.prior(2)));
 24* end;
 25  /
No of currently available colors using COUNT is 5
-----------------------------------------
value of element(1) using FIRST is violet
-----------------------------------------
```

```
value of element(2) using NEXT is indigo
-----------------------------------------
value of element(5) using LAST is yellow
-----------------------------------------
value of element(4) using PRIOR is green
-----------------------------------------
value of element 3 using index number is blue
-----------------------------------------
prior element of element 2 using index number is violet

PL/SQL procedure successfully completed.
```

Example2

```
SQL>
  1  declare
  2  type colors is table of varchar2(10);
  3  c colors := colors('violet','indigo','blue','green','yellow');
  4  i integer;
  5  begin
  6  i := c.count;
  7  dbms_output.put_line('No of currently available colors using COUNT before
TRIMMING IS '||i);
  8  dbms_output.put_line('-----------------------------------------');
  9  dbms_output.put_line('value of the elements before trimming ');
 10  for i in 1..c.count loop
 11  dbms_output.put_line('value of element ('||i||') in the collection variable c is
'||c(i));
 12  end loop;
 13  c.trim(2);
 14  dbms_output.put_line('-----------------------------------------');
 15  dbms_output.put_line('No of currently available colors using COUNT after
TRIMMING is '||c.count
 16  dbms_output.put_line('-----------------------------------------');
 17  dbms_output.put_line('value of the elements after trimming');
 18  for i in 1..c.count loop
 19  dbms_output.put_line('value of element ('||i||') in the collection variable c is
'||c(i));
 20  end loop;
 21* end;
SQL> /
```

```
No of currently available colors using COUNT before TRIMMING IS 5
----------------------------------------
value of the elements before trimming
value of element (1) in the collection variable c is violet
value of element (2) in the collection variable c is indigo
value of element (3) in the collection variable c is blue
value of element (4) in the collection variable c is green
value of element (5) in the collection variable c is yellow
----------------------------------------
No of currently available colors using COUNT after TRIMMING is 3
----------------------------------------
value of the elements after trimming
value of element (1) in the collection variable c is violet
value of element (2) in the collection variable c is indigo
value of element (3) in the collection variable c is blue

PL/SQL procedure successfully completed.
```

Example3

```
SQL>
  1  declare
  2  type colors is table of varchar2(10);
  3  c colors := colors('violet','indigo','blue');
  4  i integer;
  5  begin
  6  i := c.count;
  7  dbms_output.put_line('No of currently available colors using COUNT before
TRIMMING
  8  IS '||i);
  9  dbms_output.put_line('----------------------------------------');
 10  dbms_output.put_line('Maximum number of allowed elements in the
collection variable
 11  c is '||c.limit);
 12* end;
SQL> /
No of currently available colors using COUNT before TRIMMING
IS 3
----------------------------------------
Maximum number of allowed elements in the collection variable
c is
```

PL/SQL procedure successfully completed.

MEMBER FUNCIONS AND PROCEDURES

Syntax

```
Create type type_name [is as] object (
attribute_name data type, attribute_name data
type……,
member function specification,
member function specification, member procedure
specification);

Create type body type_name [is as]
begin
member function body;
member function body;
member function body;
end;
```

Example1

```
SQL>
  1  create or replace type m10 as object
  2  (
  3  e number(3),
  4  t number(3),
  5  member function geteng return number,
  6* member function gettam return number);
  7  /

Type created.

SQL>
  1  create or replace type body m10 as
  2  begin
  3  member function geteng return number is
  4  begin
  5  return e;
```

```
  6  end;
  7  member function gettam return number is
  8  begin
  9  return t;
 10  end;
 11* end;
 12  /

Type Body created.

SQL>
  1  declare
  2  mm m10;
  3  begin
  4  mm := m10(34,45);
  5  dbms_output.put_line(mm.geteng);
  6  dbms_output.put_line(mm.gettam);
  7* end;
SQL>/
        34
        45

Example2
SQL>
  1  create or replace type add_ty as object
  2  (h_no number(3),
  3  h_street varchar2(20),
  4  h_city varchar2(20),
  5  h_state varchar2(10),
  6  h_pin number(6),
  7  phone number(6),
  8  member procedure changeadd(no in number, st in varchar2, city in
varchar2, state in varchar2, pin in number),
 10  member function getcity return varchar2,
 11  member function getstate return varchar2,
 12  member function getpin return number,
 13* member function getphone return number);
SQL> /

Type created.
```

```
SQL>
  1  create or replace type body add_ty as
  2  member procedure changeadd(no in number,st in varchar2,city in
varchar2,state in varchar2,pin i
  3  begin
  4  if (no is null) or (st is null) or (city is null) or (state is null) or (pin is null) then
  5  raise_application_error(-20010,'null data');
  6  else
  7  h_no := no;
  8  h_street := st;
  9  h_city := city;
 10  h_state := state;
 11  h_pin := pin;
 12  end if;
 13  member function getcity return varchar2 is
 14  begin
 15  return h_city;
 16  end;
 17  member function getstate return varchar2 is
 18  begin
 19  return h_state;
 20  end;
 21  member function getpin return number is
 22  begin
 23  return h_pin;
 24  end;
 25  member function getphone return number is
 26  begin
 27  return phone;
 28  end;
 29* end;
 SQL> /

 Type body created.

SQL>
  1  declare
  2  add1 add_ty;
  3  add2 add_ty;
```

```
  4  begin
  5  add1 := add_ty(100,'first st','chennai','tn',600098,345445);
  6  add1.changeadd(10,'lock st','chennai','tn',600024);
  7  dbms_output.put_line(add1.getcity);
  8  dbms_output.put_line(add1.getstate);
  9  dbms_output.put_line(add1.getpin);
 10  dbms_output.put_line(add1.getphone);
 11* end;
SQL> /
        Chennai
        tn
        600098
        345445
```

CASE STATEMENT

Description

As of Oracle9i,We can use the CASE function in place of DECODE. The CASE function uses the keywords when,then,else,and end.

Example1

```
SQL> select * from bab;

NAME                 ENG       TAM       MAT ODATE
---------------- ---------- ---------- ---------- ---------
babu                 200       100        90 12-MAR-98
ddl                   67        45        67 12-MAR-99
sandhya               45        67        89 12-MAY-99
mani                  67        89        56 12-AUG-96
kavi                  56        78        90 12-MAR-98
mmm                  300       400        76 12-MAY-98
mmm                   23        45       100 12-JAN-99

7 rows selected.


SQL>
  1  select case  when eng>30 and eng<50  then 'GOOD'
  2               when eng>50 and eng<100 then 'EXCELLENT'
  3              else 'AVERAGE'
  4         end marks
  5*    from bab
```

```
SQL> /

MARKS
---------
AVERAGE
EXCELLENT
GOOD
EXCELLENT
EXCELLENT
AVERAGE
AVERAGE

7 rows selected.
```

Example2

```
SQL> select * from bab;

NAME                ENG       TAM       MAT ODATE
--------------- ---------- ---------- ---------- ---------
babu                200       100        90 12-MAR-98
ddl                 67        45         67 12-MAR-99
sandhya             45        67         89 12-MAY-99
mani                67        89         56 12-AUG-96
kavi                56        78         90 12-MAR-98
mmm                 300       400        76 12-MAY-98
mmm                 23        45        100 12-JAN-99

SQL> edit case
SQL>
  1  select distinct
  2     CASE name
  3     when 'babu' then 'Mr.k.babu'
  4     when 'ddl'  then 'Mrs.dhanalakshmi'
  5     when 'sandhya' then 'Miss.sandhya babu'
  6     else 'aaaa'
  7  end Staff_Names
  8* from bab
  9  /

STAFF_NAMES
-----------------
```

Miss.sandhya babu
Mr.k.babu
Mrs.dhanalakshmi
aaaa

Example3

```
SQL>
  1  select
  2     CASE name
  3     when 'babu' then 'Mr.k.babu'
  4     when 'ddl'  then 'Mrs.dhanalakshmi'
  5     when 'sandhya' then 'Miss.sandhya babu'
  6     else 'dummy'
  7  end Staff_Names
  8* from bab
  9  /

STAFF_NAMES
----------------
Mr.k.babu
Mrs.dhanalakshmi
Miss.sandhya babu
dummy
dummy
dummy
dummy

7 rows selected.
```