# ORACLE

## Short Cut keys for ORACLE 8

SQL*Plus assigns the traditional functions to most of the command keys. However, the following command keys have special functions in SQL*Plus for Windows NT and Windows 95

| Key | Function |
|---|---|
| HOME | Top of screen buffer |
| END | Bottom of screen buffer |
| CTRL+HOME | Right side of screen buffer |
| CTRL+END | Left side of screen buffer |
| PAGE UP | Previous screen page |
| PAGE DOWN | Next screen page |
| CTRL+PAGE UP | Show page on left of current screen page |
| CTRL+PAGE DOWN | Show page on right of current screen page |
| F3 | Find |
| ALT+F3 | Find next |
| CTRL+C | Copy text |
| CTRL+V | Paste text |
| SHIFT+DEL | Clear the screen buffer |

Rathod Software

## ORACLE HAS FOLLOWING DATA TYPES

| Data type | Memory Storing Capacity | Example | Usage |
|---|---|---|---|
| Number | | Num1 Number(38,38) | To store Numeric Value |
| Date | 07 bytes | J_date Date | To store date Value in format =(dd-mon-yy) |
| Char | 2000 bytes | Name Char(30) | To store character value (character value array) |
| Varchar2 | 4000 bytes | Name Varchar2(25) | To store Alphanumeric data |
| Long | 2 GB | Remark Long | To store Alphanumeric data |
| Raw | 2000 bytes | | To store Binary data (non character details) |
| Long Raw | 2 GB | | To store Binary data |
| BLOB | 4 GB | B_LOB BLOB | To store Binary data which has 4 GB length |
| CLOB | 4 GB | C_LOB CLOB | To store Character data which has 4 GB length |
| BFILE | Depends on OS | B_FILE BFILE | To store data out of database. It's length is depend on os |

Rathod Software

## ORACLE OBJECTS

### * TABLE

Table is a used as a basic storage unit.
We can use CREATE COMMAND to create table.

Table is a collection of rows and columns, each columns has it's different values so it will create one record.

Record is a collection of interrelated data or collection of data it will create record set.

## SQL COMMANDS

It has three types describe as follows.

### 1). DDL (DATA DEFINITION LANGUAGE)

| | |
|---|---|
| **1). CREATE** | To create table or objects |
| **2). ALTER** | To alter existing database |
| **3). DROP** | To drop existing objects |
| **4). TRUNCATE** | To remove whole data at a time |

### 2). DML (DATA MANIPULATION LANGUAGE)

| | |
|---|---|
| **1). INSERT** | To insert data in table |
| **2). UPDATE** | To update existing data in table |
| **3). SELECT** | To view database |
| **4). DELETE** | To delete particular records in database |

### 3). TCL (TRANSACTION CONTROL LANGUAGE)

| | |
|---|---|
| **1). COMMIT** | To save buffer data to storage device |
| **2). ROLLBACK** | To undo saves |
| **3). SAVEPOINT** | To keep break in save action |

### 4). DCL (DECISION CONTROL LANGUAGE)

| | |
|---|---|
| **1). GRANT** | To provide rights for user on database |
| **2). REVOKE** | To revoke user rights |

Rahod Software

## 1). CREATE (DDL)

The CREATE TABLE command defines each column of the table uniquely. Each Column has a minimum three attributes, a name, data type and size.

**SYNTAX:**

```
CREATE TABLE <TABLE NAME>
(<COLUMNNAME1> <DATA TYPE> (<size>),
 <COLUMNNAME2> <DATA TYPE> (<size>));
```

**EXAMPLE:**

```
CREATE TABLE LOGIN1
(USERNAME VARCHAR2 (25),
PASSWORD VARCHAR2 (25));
```

## 2). INSERT (DML)

The INSERT command is used to insert data into database.

**SYNTAX:**

```
INSERT INTO<TABLE NAME>
VALUES(VALUE1,VALUE2,VALUE3,…);
```

**EXAMPLE:**

```
INSERT INTO LOGIN
VALUES('ADMIN','ADMIN');

INSERT INTO LOGIN1
VALUES ('&USERNAME','&PASSWORD');
```

## 3). UPDATE (DML)

The UPDATE command is used to change any details in database.

**SYNTAX:**

```
UPDATE <TABLE NAME>
SET<COLUMNNAME>=<VALUE>
WHERE <COLUMNNAME>=<VALUE>;
```

**EXAMPLE:**

```
UPDATE LOGIN
SET USERNAME='ADMIN';


UPDATE LOGIN
SET USERNAME='VIJAY'
WHERE USERNAME='ADMIN';
```

Python Software

## 4). DELETE (DML)

The DELETE command is used to delete any record or group of records.

### SYNTAX:

```
DELETE <TABLE NAME>
[WHERE<COLUMNNAME>=<VALUE>];
```

### EXAMPLE:

```
DELETE LOGIN;

DELETE LOGIN
WHERE USERNAME='ADMIN';
```

## 5). SELECT (DML)

The SELECT command is used to display records form database.

### SYNTAX:

```
SELECT<COLUMN LIST>
FROM<TABLE NAME>
[WHERE<CONDITION>]
[GROUP BY<COLUMN>]
[HAVING<CONDITION>]
[ORDER BY<COLUMN>][DESC];
```

### EXAMPLE:

```
SELECT * FROM LOGIN;

SELECT EMPNO,ENAME
FROM EMP;

SELECT * FROM EMP
WHERE SAL>1500;

select sum(sal) from emp where deptno=30 group by deptno;

select deptno,sum(sal) from emp where deptno=30 group by deptno;

select deptno,sum(sal) from emp  group by deptno;

SELECT SUM(SAL) FROM EMP
GROUP BY DEPTNO;


SELECT * FROM EMP
ORDER BY ENAME;
```

Rahod Software

```
SELECT * FROM EMP
ORDER BY ENAME DESC;
```

### 6). ALTER (DML)

The ALTER command is used to modify data base structure with it's options like ADD, MODIFY.

**SYNTAX:**

**1). ALTER WITH ADD**

```
ALTER TABLE <TABLE NAME>
ADD <COLUMN NAME> (SIZE);
```

**2). ALTER WITH MODIFY**

```
ALTER TABLE <TABLE NAME>
MODIFY <COLUMN NAME> (SIZE);
```

**NOTE :** IF WE WANT TO MODIFY ANY FIELDS IN DATABASE WE MUST SET IT TO NULL.

**EXAMPLE:**

```
ALTER TABLE <TABLE NAME>
ADD ACC NUMBER(3);

UPDATE LOGIN
SET ACC=NULL;

ALTER TABLE LOGIN
MODIFY ACC VARCHAR2(3);
```

### 7). RENAME (DML)

The RENAME command is used to modify TABLE NAME.

**SYNTAX:**

```
RENAME <TABLE NAME>
TO <NEW TABLE NAME >;
```

**EXAMPLE:**

```
RENAME LOGIN
TO LOGIN1;
```

### 8). COMMIT (TCL)

The COMMIT command is used to store data permanently in database.

**SYNTAX:**

Rathod Software

```
COMMIT;
```

**EXAMPLE:**

```
COMMIT;
```

### 10). SAVEPOINT (TCL)

The SAVEPOINT command is used to create restore point.

**SYNTAX:**

```
SAVEPOINT <SAVEPOINT NAME>;
```

**EXAMPLE:**

```
SAVEPOINT S1;

DELETE EMP WHERE SAL>=5000;

SAVEPOINT SAVE2;

UPDATE EMP
SET COMM=1500
WHERE SAL<=1200;
```

### 11). ROLLBACK (TCL)

The ROLLBACK command is used to restore data.

**SYNTAX:**

```
ROLLBACK TO <SAVEPOINT NAME>;
```

**EXAMPLE:**

```
ROLLBACK TO S1;
```

Rathod Software

## PRIMARY KEY

We can have only one primary key and a primary key column does not contain Null values.

## Constraint

This constraint declares a column as the primary key of the table. This constraint is similar to unique constraint except that only one column (or one group of columns) can be applied in this constraint.

**EXAMPLE:**

```
CREATE TABLE TEST
(RNO NUMBER(3) PRIMARY KEY,
NAME VARCHAR2(20));
```

## COMPOSITE KEY

THIS KEY IS USED TO CHECK TWO OR MORE FIELD AT A TIME.

**EXAMPLE:**

```
CREATE TABLE TEST1
(RNO NUMBER(3),
NAME VARCHAR2(20), PRIMARY KEY (RNO, NAME));
```

## FOREIGN KEY

A foreign key is a combination of columns with values based on the primary key values from another table.

A foreign key constraint, also known as a **referential integrity constraint**, specifies that the value of foreign key correspond to actual value of the primary key in the other table.

**EXAMPLE:**

```
CREATE TABLE EMP11
(EMPNO NUMBER(5) PRIMARY KEY,
ENAME VARCHAR2(25) NOT NULL,
JOB VARCHAR2(30) NOT NULL,
PHONE_NO VARCHAR2(15) NOT NULL,
SAL NUMBER(8,2),
GENDER CHAR CONSTRAINT CHK1 CHECK(SEX IN('M','F')),
DEPTNO NUMBER(2) NOT NULL,
CONSTRAINT CHK2 CHECK(DEPTNO IN(10, 20, 30, 40)),
CONSTRAINT CHK3 CHECK (SAL>0));

INSERT INTO EMP11
VALUES(&EMPNO,'&ENAME','&JOB','&PHONE_NO',&SAL,'&SEX',&DEPTNO)
/
```

# Example:

Siddhod Software

```
create table master_item1 (ino number(5), iname varchar2(50) primary
key(ino,iname));
ALTER TABLE master_item1 ADD CONSTRAINT table_pk PRIMARY KEY (ino, iname);
select * from master_item1;
describe master_item1;
insert into master_item1 values(1,'ajay');
```

## REFERENTIAL INTEGRITY CONSTRAINT

- It must have two tables for create referential integrity constraint.
- It must have one master table and one slave / child table.

**Example:**

```
CREATE TABLE dept1
(deptno number(2) primary key,
dname varchar2(25) not null,
loc varchar2(30) not null);

CREATE TABLE emp1
(eno number(3) primary key,
ename varchar2(25) not null,
add1 varchar2(30) not null,
city varchar2(12) not null,
job varchar2(25) not null,
hiredate date,
sal number(15, 2),
comm number(2),
deptno number(2),
constraint fk foreign key(deptno)
references dept1(deptno) on delete cascade);


INSERT INTO DEPT1
VALUES (&DEPTNO,'&DNAME','&LOC')
/

INSERT INTO emp1
VALUES (&eno, '&ename', '&add', '&city', '&job', '&hiredate', &sal, &comm, &deptno)
/
```

Patnod Software

## OPERATORS

### 1) ARITHMATIC

+
-
*
/
** Exponentiation * DUE
() Enclosed Operation * DUE

**example of exponentiation:**

**select 5*10**3 from dual;**

### 2) RELATIONAL

>
>=
<
<=
=
!=
BETWEEN

Example:

```
select * from emp1
where sal BETWEEN 1500 and 18000
```

LIKE

EXAMPLE:

```
select * from emp
where ename like 'C%'

select * from emp
where ename like'%LL%'
/
```

IS

EXAMPLE:

```
SELECT * FROM EMP
WHERE COMM IS NULL;
```

IN

EXAMPLE:

```
SELECT * FROM EMP
WHERE JOB IN('SALESMAN','CLERK')
```

### 3) LOGICAL

AND

Rathod Software

OR
NOT
```
select * from emp
where not sal<=3000
/
```

Rathod Software

# FUNCTIONS
## A). CHARACTER FUNCTIONS

**1). UPPER(STRING)**

```
SELECT UPPER(ENAME) FROM EMP;
```

**2). LOWER(STRING)**

```
SELECT LOWER(ENAME) FROM EMP;
```

**3). INITCAP(STRING)**

This function is used to convert string in proper case.

```
SELECT INITCAP('I LOVE INDIA') FROM DUAL;
```

**4). ASCII(CHARACTER)**

```
SELECT ASCII('A') FROM DUAL;
```

**5). CHR(NUMBER)**

```
SELECT CHR(97) FROM DUAL;
```

**6). LENGTH(STRING)**

```
SELECT LENGTH('I LOVE INDIA') FROM DUAL
/
SELECT LENGTH(ENAME),ENAME FROM EMP
/
```

**7). LTRIM(STRING)**

```
SELECT LTRIM('     I LOVE INDIA') FROM DUAL
/
```

**8). RTRIM(STRING)**

```
SELECT RTRIM('I LOVE INDIA       ') FROM DUAL
/
```

**9). SUBSTR(STRING,STARTINGPOSITION,NOCHARACTER)**

```
SELECT SUBSTR('I LOVE INDIA ',8,5) FROM DUAL
/
```

**10). INSTR(STRING,SEARCHING CHARACTER,STARTINGPOSITION,NOOFOCCURENCE)**

IT WILL RETURN POSITION OF LAST OCCURRENCE.

```
SELECT INSTR('I LOVE INDIA ','I') FROM DUAL;
SELECT INSTR('I LOVE INDIA ','I',3) FROM DUAL;
SELECT INSTR('I LOVE INDIA ','I',1,3) FROM DUAL;
```

IN THIRD EXAMPLE WE MUST SPACIFY STARTING POSITION. IF WE DO NOT DEFINE STARTING POSITION SO THAT IT WILL NOT DISPLAY ANY OUTPUT ON SCREEN.

Pathod Software

**11). REPLACE(STRING,FINDING TEXT,REPLACING TEXT)**

```
SELECT REPLACE('WIFE','W','KN') FROM DUAL;
```

**12). TRANSLATE(STRING,FIND TEXT,REPLACING TEXT)**

IT WILL REPLACE CHARACTER INSTADE OF ANOTHER CHARACTER.

```
SELECT TRANSLATE('NIL','LIN','END') FROM DUAL;
SELECT TRANSLATE('PAT OLP','PA ','BD') FROM DUAL;
SELECT TRANSLATE('THIS IS FOR TESTING','IS  ','ARE  ') FROM DUAL;
```

NOTE: ABOVE FUNCTION WILL REPLACE DATA CHARACTER BY CHARACTER….

**13). LPAD(STRING,NOOFCHARACTER,LENGTH,CHARACTERTOPAD)**

```
SELECT LPAD('INDIA',8,'*') FROM DUAL;
```

**14). RPAD(STRING,NOOFCHARACTER,LENGTH,CHARACTERTOPAD)**

```
SELECT RPAD('INDIA',8,'*') FROM DUAL;
```

**15). SOUNDX(STRING)**

```
SELECT SOUNDX(ENAME) FROM EMP;
```

## B). NUMBERIC FUNCTIONS

**1). ROUND(NUMBER,ROUNDING PLACE)**

```
SELECT ROUND(459.3167) FROM DUAL
/

SELECT ROUND(459.3167,3) FROM DUAL
/
```

**2). MOD(NO. TO DIVIDE, NO. TO DIVIDE WITH)**

```
SELECT MOD(4,2) FROM DUAL;
```

**3). SQRT(NUMBER)**

```
SELECT SQRT(25) FROM DUAL;
```

**4). ABS(NUMBER)**

```
SELECT ABS(-25) FROM DUAL;
```

**5). POWER(NUMBER,PAISED NUMBER)**

```
SELECT POWER(2,2) FROM DUAL;
```

**6). TRUNC(DECIMAL VALUE)**

```
SELECT TRUNC(22.55) FROM DUAL;
```

**7). CEIL(NUMBER)**

```
SELECT CEIL(22.10) FROM DUAL;
```

Rahul Software

**8). FLOOR(NUMBER)**

```
SELECT FLOOR(22.90) FROM DUAL;
```

**9). EXP(NUMBER)**

```
SELECT EXP(1) FROM DUAL;
```

**10). LN(NUMBER)**

```
SELECT LN(3) FROM DUAL;
```

## C). DATE FUNCTIONS

**1). SYSDATE()**

```
SELECT SYSDATE FROM DUAL;
```

**2). ADD_MONTHS(DATE,NO)**

```
SELECT ADD_MONTHS(SYSDATE,2) FROM DUAL;
```

**3). LAST_DAY(DATE)**

```
SELECT LAST_DAY(SYSDATE) FROM DUAL;
```

**4). MONTHS_BETWEEN(DATE,DATE2)**

```
SELECT MONTHS_BETWEEN(ADD_MONTHS(SYSDATE,2),SYSDATE) FROM DUAL;
SELECT MONTHS_BETWEEN('25-JUN-12','25-FEB-12') FROM DUAL;
```

**5). NEXT_DAY(DATE,DAY)**

```
SELECT NEXT_DAY(SYSDATE,'MON') FROM DUAL
/
```

**6). GREATEST(DATE,DATE2,DATE3,….)**

```
SELECT GREATEST(SYSDATE,SYSDATE-45,SYSDATE+25) FROM DUAL;
```

**7). LEAST(DATE,DATE2,DATE3,….)**

```
SELECT LEAST(SYSDATE,SYSDATE-45,SYSDATE+25) FROM DUAL;
```

Ratnod Software

## D). CONVERSION FUNCTIONS

**1). TO_CHAR(DATE,FORMAT)**

```
SELECT TO_CHAR(SYSDATE,'DD-MM-YYYY') FROM DUAL;
SELECT TO_CHAR(SYSDATE,'DD-MONTH-YY') FROM DUAL;
```

**2). TO_DATE(STRING,FORMAT)**

```
SELECT TO_CHAR(TO_DATE(101,'J'),'JSP') FROM DUAL
/

SELECT TO_CHAR(TO_DATE('10-01-08','MM-DD-YYYY')) FROM DUAL
/
```

**3). TO_NUMBER(STRING,FORMAT)**

```
SELECT TO_NUMBER('100.00','9G999D99')FROM DUAL
/
```

## E). MISCELINEOUS FUNCTIONS

**1). USER**

```
SELECT USER FROM DUAL;
```

**2). UID**

```
SELECT UID FROM DUAL;
```

**3). VSIZE**

```
SELECT VSIZE(ENAME) FROM EMP;
```

**4). NVL(COLUMN NAME,VALUE IF NULL)**

This function replace the specified value when a NULL value is encountered. Using the SQL

```
UPDATE EMP SET salary=NVL(COMM,10);
```

**5). DECODE(COLUMN NAME,IF,THEN)**

```
SELECT DECODE (UPPER(JOB),'MANAGER','FIRST','CLERK','SECOND','OTHERS')
FROM EMP;
```

**(IT'S WORK LIKE IF…ELSE IF STRUCTURE.)**

**6). DESTINCT(COLUMN)**

```
SELECT DISTINCT(DEPTNO) FROM EMP;
```

Rathod Software

## F).GROUP FUNCTIONS

### 1). SUM(COLUMN NAME)

This function is used to get sum of whole column.

```
SELECT SUM(SAL) FROM EMP;

SELECT SUM(SAL),DEPTNO
FROM EMP
GROUP BY DEPTNO
/
```

### 2). AVG(COLUMN NAME)

This function is used to get average of whole column.

```
SELECT avg(SAL) FROM EMP;

SELECT AVG(SAL),DEPTNO
FROM EMP
GROUP BY DEPTNO
/
```

### 3). MIN(COLUMN NAME)

This function is used to get MINIMUM VALUE FROM column.

```
SELECT MIN(SAL) FROM EMP;
```

### 4). MAX (COLUMN NAME)

This function is used to get MAXIMUM VALUE FROM column.

```
SELECT MAX(SAL) FROM EMP;
```

### 5). COUNT (COLUMN NAME)

This function is used to get MAXIMUM VALUE FROM column.

```
SELECT COUNT(DEPTNO) FROM EMP GROUP BY DEPTNO;
```

Rathod Software

# SUB QUERY

```
SELECT * FROM EMP
WHERE SAL<(SELECT AVG(SAL) FROM EMP)
/

SELECT * FROM EMP
WHERE DNO IN(SELECT DNO FROM DEPT WHERE LOC='jamnagar' OR LOC='Rajkot')
/

SELECT * FROM EMP
WHERE DNO IN(SELECT DNO FROM DEPT WHERE LOC='Jamnagar' OR LOC='Rajkot') ORDER BY
DNO
/

SELECT * FROM EMP
WHERE DNO IN(SELECT DNO FROM DEPT WHERE LOC='Jamnagar' OR SALARY<(SELECT
AVG(SALARY) FROM EMP)) ORDER BY DNO;



SELECT MAX(HIREDATE) FROM EMP
WHERE HIREDATE IN(SELECT HIREDATE FROM EMP WHERE JOB='MANAGER' OR
DEPTNO IN(SELECT DEPTNO FROM DEPT WHERE LOC='NEW YORK' OR LOC='CHICAGO'))
/

SELECT MAX(HIREDATE) FROM EMP
WHERE HIREDATE IN(SELECT HIREDATE FROM EMP WHERE JOB='MANAGER' OR DEPTNO
IN(10,30))
/

SELECT * FROM EMP
WHERE DEPTNO=(SELECT MAX(DEPTNO) FROM DEPT WHERE DEPTNO IN (10,20))
/
```

# JOINS

- **SELF JOIN**
- **SIMPLE JOIN**
- **OUTER JOIN & INNER JOIN**

- **SELF JOIN**

```
SELECT X.EMPNO,Y.EMPNO FROM EMP X,EMP Y
/
```

Patnod Software

select x.eno,x.ename, x.salary, y.dno, y.dname from emp x, dept y where x.dno=y.dno;

SELECT X.ENAME "EMPLOYEE",Y.ENAME "MANAGER"
FROM EMP X, EMP Y
WHERE Y.EMPNO=X.EMPNO
/

**SELECT X.eno "EMPLOYEE NO",Y.ENAME "EMPLOYEE NAME",Y.DNO "DEPARTMENT NO" FROM EMP X, EMP Y**
**WHERE Y.ENO=X.ENO AND X.DNO=20; /**

- **SIMPLE JOIN /EQUI JOIN**

SELECT A.*,B.DNAME,D.LOC FROM EMP A, DEPT B,DEPT D
WHERE A.DNO=B.DNO ORDER BY ENAME
/

SELECT A.EMPNO,B.DNAME,D.LOC,D.DEPTNO FROM EMP A, DEPT B,DEPT D
WHERE A.DEPTNO=B.DEPTNO AND B.DEPTNO=D.DEPTNO ORDER BY ENAME
/

SELECT A.*,B.DNAME,b.LOC FROM EMP A, DEPT B
WHERE A.DEPTNO=B.DEPTNO
ORDER BY ENAME
/

SELECT A.ENAME,B.DNAME,D.LOC FROM EMP A, DEPT B,DEPT D
WHERE A.DEPTNO=B.DEPTNO AND A.DEPTNO=D.DEPTNO ORDER BY ENAME
/

- **OUTER JOIN & INNER JOIN**

SELECT EMPNO,ENAME,JOB,DEPT.DEPTNO,DEPT.DNAME FROM EMP,DEPT
WHERE DEPT.DEPTNO=EMP.DEPTNO(+)
/

# view

create table master_customer (cno number(5) primary key, cname varchar2(25) not null);

create table master_item
(ino number(5) primary key,
iname varchar2(25)not null);

Rahul Software

```
create table sales
(sno number(5) primary key,
sdate date not null,
cno number(5) references master_customer(cno) ,
ino number(5) references master_item(ino),
qty number(5) not null,
rate number(8,2) not null,
amount number(10,2),
ta number(11,2),
da number(11,2),
ga number(11,2),
remarks varchar2(200));



create view sales_view as
select
sales.sno,
sales.sdate,
sales.cno,
master_customer.cname,
sales.ino,
master_item.iname,
sales.qty,
sales.rate,
sales.amount,
sales.ta,
sales.da,
sales.ga,
sales.remarks
from sales,
master_customer,
master_item
where
sales.cno=master_customer.cno and
sales.ino=master_item.ino
/

CREATE OR REPLACE VIEW EMP_TMP AS SELECT
EMP.ENO,
EMP.ENAME,
EMP.SALARY,
EMP.DNO,
DEPT.DNAME,
DEPT.LOC
FROM EMP,DEPT
```

Rathod Software

WHERE EMP.DNO=DEPT.DNO;

# SEQUENCE-

Sequence is used to create auto number for tables.

**Syntax:**

Create sequence <sequence name>
Start with <value> increment by <value>
Maxvalue <value> minvalue <value>
Cycle /nocycle cache <value>;

**Example:**

```
create sequence s1 start with 1 increment by 1 maxvalue 10 minvalue 1
cycle cache 2
/

CREATE TABLE TMP1
 (TNO NUMBER(3),
 TNAME VARCHAR2(25))
/

INSERT INTO TMP1
VALUES(S1.NEXTVAL,'&TNAME');
```

Create sequence s2 Start with 10 increment by 1 Maxvalue 50 minvalue 10 Cycle;

```
INSERT INTO TMP1
VALUES(S2.NEXTVAL,'&TNAME');
```

Create sequence s5 start with 10 increment by 10 Maxvalue 50 minvalue 10 nocycle cache 11
/

```
INSERT INTO TMP1
```

Rathod Software

VALUES(S5.NEXTVAL,'&TNAME');

# SYNONYMS:

Synonyms is used to set pet name (“alias”) for any object, table or for synonyms.

**Syntax:**

Create synonym <synonym name>
From <object name>;

**Example:**

Create synonym s1 for emp1;

select * from s1
/

Rathod Software

# INDEX:

INDEX is used to access data very fast from database.

**Syntax:**
Create index <index name> on table name(column name);

**Example:**

```
Create index ind1 on emp (ename)
/
```

Rathod Software

# PL /SQL BLOCK

It has following structure.
**SYNTAX:**

BEGIN –optional, denotes beginning of block

DECLARE—optional, variable definitions

BEGIN—mandatory, denotes beginning of procedure section

EXCEPTION—optional, denotes beginning of exception section

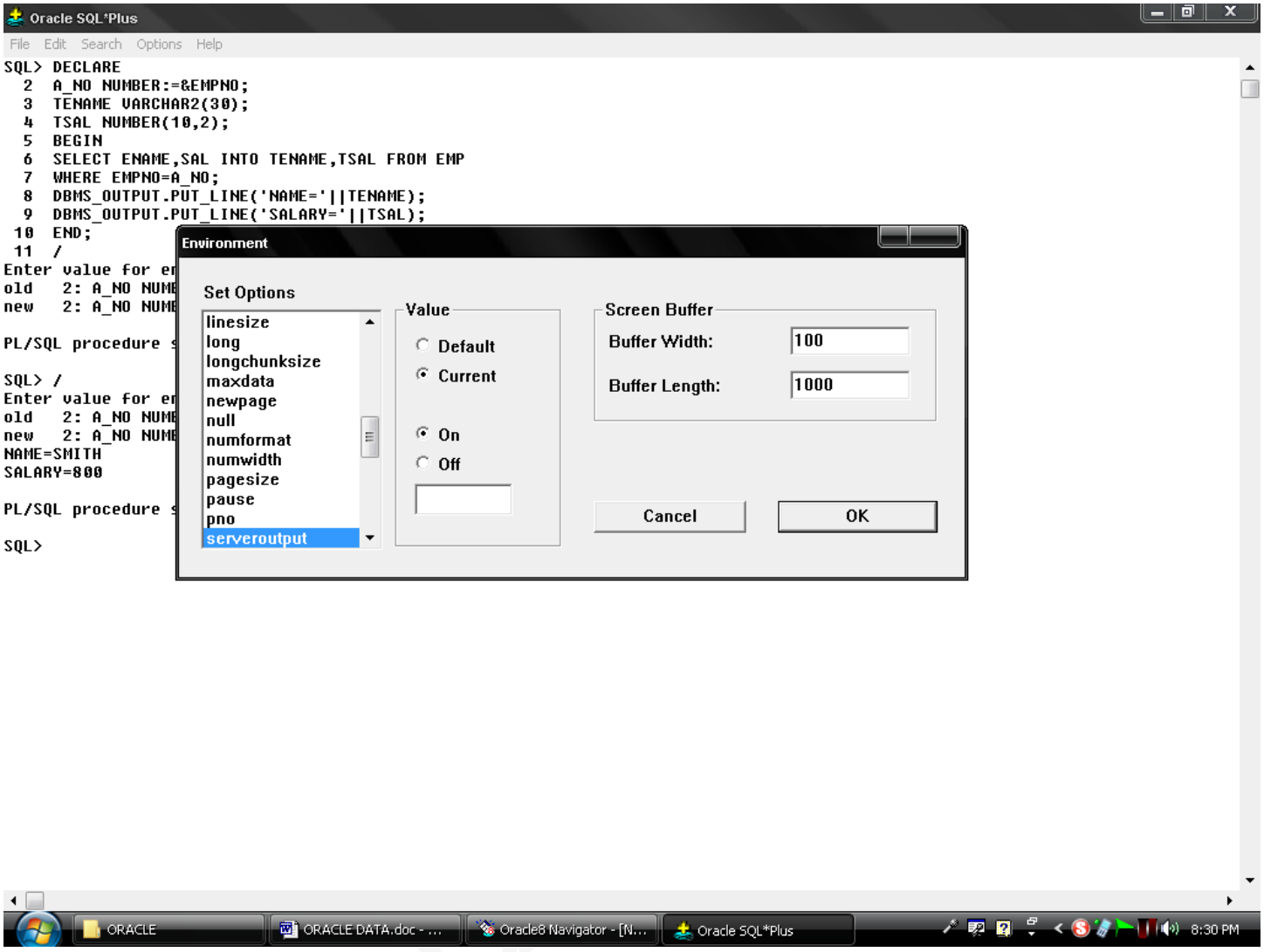
**EXAMPLE 1:**

```
DECLARE
A_NO NUMBER:=1;
TENAME VARCHAR2(30);
TSAL NUMBER(10,2);
BEGIN
SELECT ENAME,SALARY INTO TENAME,TSAL FROM EMP
WHERE ENO=A_NO;
DBMS_OUTPUT.PUT_LINE('NAME='||TENAME);
DBMS_OUTPUT.PUT_LINE('SALARY='||TSAL);
END;
```

Note: To see output we may use following steps

1. Go to options menu
2. select Environment option
3. it will display following screen

Rajnod Software




4. select serveroutput then select value=current then select on then click on ok

**EXAMPLE 2:**

```
DECLARE
      J NUMBER(5);
      T NUMBER(10);
BEGIN
      J:=10;
      SELECT AVG(SALARY) INTO T FROM EMP WHERE DNO=J GROUP BY
DNO;
      DBMS_OUTPUT.PUT_LINE('DEPARTMENT:'||J || '   SALARY:'||T);
END;
```

**NOTE: WE MAY GET THE ARGUMENT FROM THE USER BY USING :<VARIABLENAME> IN OLDER VERSION WE WERE USING &.**

**EXAMPLE 3:**

```
DECLARE
      J NUMBER(5);
      T NUMBER(10);
BEGIN
      J:=:J;
      SELECT AVG(SALARY) INTO T FROM EMP WHERE DNO=J GROUP BY
DNO;
      DBMS_OUTPUT.PUT_LINE('DEPARTMENT:'||J || '   SALARY:'||T);
END;
```

**WE CAN SAVE PL/SQL BLOCK BY USING**

**Step-1:** Click on Sql Scripts
**Step-2:** Then click on create button
**Step-3:** Then type the PL/SQL Block then set name for the same
**Step-4:** Then click on save button it will save your PL/SQL block
**Step-5:** Select specific Script from the list and click on run button
**Step-6:** It will ask you to execute selected script, then click on run button.
**Step-7:** It will display the result screen then click on view result option
**Step-8:** Then select detail option, then click on go button it will display the PL/SQL block and it's result on the screen.

**Note: we may save and execute scripts only in Internet Explorer other browsers are not supporting this functionality.**

Rathod Software

**EXAMPLE 3:**

```
DECLARE
   N NUMBER:=:ENO;
   I NUMBER(10,2):=:I;
BEGIN
   UPDATE EMP SET SAL=SAL+ I WHERE ENO=N;
END;
```

**Conditional statements in pl/sql block**

**EXAMPLE 4:**

```
declare
  billno varchar2(3);
  pname varchar2(25);
  iname varchar2(25);
  rate number;
  qty number;
  amt number;
  dis number;
  net number;
begin
  billno:=:billno;
  pname:=:pname;
  iname:=:iname;
  rate:=:rate;
  qty:=:qty;
  amt:=qty*rate;

  if amt>=5000 then
    dis:=amt*0.20;
  ELSif amt>=2000 then
    dis:=amt*0.15;
else
    dis:=amt*0.10;
End if;
net:=amt-dis;
dbms_output.put_line('Net amount is ='||net);
```

Method Software

```
 end;
/
```

**Example:**

```
declare
	a varchar2(50);
	 b varchar2(50);
	n number(5);
begin
	a:=:a;
	n:=length(a);

	loop
		exit when n<1;
		 b:=concat(b,substr(a,n,1));
		n:=n-1;

	end loop;
	if a<>b then
		dbms_output.put_line('false');
	end if;
end;
```

**Example:**

```
declare
	a number(5);
	n number(5);
	b number(5);
	c number(5);
	d number(5);
begin
	c:=8;
	n:=1;
	loop
		exit when n>5;
		b:=1;
		loop
			exit when b>c;
			dbms_output.put(' ');
			b:=b+1;
		end loop;
```

Ratnod Software

```
            c:=c-1;
            a:=1;
            loop
                  exit when a>n;
                  dbms_output.put(a);
                  a:=a+1;
            end loop;
            d:=a-2;
            loop
                  exit when d<1;
                  dbms_output.put(d);
                  d:=d-1;
            end loop;
                  dbms_output.put_line('');
      n:=n+1;
      end loop;
end;
```

**Example: GOTO**

```
DECLARE
      a number(2);
      cont varchar2(2);
      t number(8);
BEGIN
      t:=0;
      a:=0;
      <<loopstart>>
      a:=:a;
      t:= t + a;
      if a>=1 then
            cont:=:cont;
      end if;
      IF cont = 'y' THEN
            GOTO loopstart;
      END IF;
      dbms_output.put_line ('total is: ' || t);
END;
```

Method Software

**PL/SQL ATTRIBUTES:**

**1). %TYPE**

**2). %ROWTYPE**

**1). %TYPE**

**IT WILL STORE DATA TYPE WHICH IS USED IN DATABASE (TABLE) FOR PERTICULAR FIELD.**

**EXAMPLE:**

```
DECLARE
      TNAME EMP.ENAME%TYPE:=:TNAME;
      TSAL EMP.SALARY%TYPE;
BEGIN
      SELECT SALARY INTO TSAL FROM EMP WHERE
LOWER(ENAME)=LOWER(TNAME);
      DBMS_OUTPUT.PUT_LINE('SAL ='||TSAL);
END;
```

**EXAMPLE 2:**

```
DECLARE
 TDNO EMP.DNO%TYPE:=:DNO;
 TSAL EMP.SALARY%TYPE;
BEGIN
 SELECT SUM(SALARY) INTO TSAL FROM EMP WHERE DNO=TDNO GROUP
BY DNO;
 DBMS_OUTPUT.PUT_LINE('SAL IS='||TSAL);
END;
/
```

**2). %ROWTYPE**

**IT WILL ASSIGN ALL FIELDS IN ONE VAIABLE.**

Kathoju Software

**EXAMPLE:**

```
DECLARE
E EMP%ROWTYPE;
TENO EMP.ENO%TYPE:=:TENO;
BEGIN
SELECT * INTO E FROM EMP WHERE ENO=TENO;
DBMS_OUTPUT.PUT_LINE('EMP NAME='||E.ENAME);
DBMS_OUTPUT.PUT_LINE('JOB='||E.SALARY);
DBMS_OUTPUT.PUT_LINE('SAL='||E.DNO);
END;
```

**CONTROL STRUCTUERS IN PL/SQL BLOCK**

**1). SIMPLE IF**

SYNTAX:

```
IF <CONDITION> THEN
        STATEMENT BLOCK 1;
END IF;
```

**EXAMPLE:**

```
DECLARE
    A NUMBER;
BEGIN
     DBMS_OUTPUT.PUT_LINE('ENTER AGE=');
     A:=:A;
     IF (A>=18) THEN
          DBMS_OUTPUT.PUT_LINE('YOU CAN VOTE');
     END IF;
END;
```

**2). IF…ELSE**

SYNTAX:

```
IF <CONDITION> THEN
        STATEMENT BLOCK 1;
ELSE
        STATEMENT BLOCK 2;
END IF;
```

Rathod Software

**EXAMPLE:**

```
DECLARE
    A NUMBER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('ENTER AGE');
    A:=:A;
    IF (A>=18) THEN
        DBMS_OUTPUT.PUT_LINE('YOU CAN VOTE');
    ELSE
        DBMS_OUTPUT.PUT_LINE('YOU CAN NOT VOTE');
    END IF;
END;
```

3). IF…ELSIF

SYNTAX:

```
IF <CONDITION> THEN
        STATEMENT BLOCK 1;
ELSIF <CONDITION> THEN
        STATEMENT BLOCK 2;
ELSE
        STATEMENT BLOCK 3;
END IF;
```

**EXAMPLE:**

```
DECLARE
    A NUMBER;
BEGIN
    A:=:A;
    IF (A>=18) THEN
        DBMS_OUTPUT.PUT_LINE('YOU CAN VOTE');
    ELSIF(A<18) THEN
        DBMS_OUTPUT.PUT_LINE('YOU CAN NOT VOTE');
    ELSE
        DBMS_OUTPUT.PUT_LINE('INVALID INPUT');
    END IF;
END;
```

Rathod Software

**EXAMPLE:**

```
DECLARE
      A NUMBER;
BEGIN
      A:=:A;
      IF (A=1) THEN
            DBMS_OUTPUT.PUT_LINE('ONE');
      ELSIF(A=2) THEN
            DBMS_OUTPUT.PUT_LINE('TWO');
      ELSE
            DBMS_OUTPUT.PUT_LINE('INVALID INPUT');
      END IF;
END;
/
```

**LOOPS**

**1).SIMPLE LOOP**
**2).WHILE LOOP**
**3).FOR LOOP**

**1).SIMPLE LOOP**

SYNTAX:

```
      LOOP
            STATEMENT BLOCK;
      END LOOP;
```

**EXAMPLE 1:**

```
DECLARE
      N NUMBER:=1;
BEGIN
   LOOP
            DBMS_OUTPUT.PUT_LINE(N);
            N:=N+1;
            EXIT WHEN N>5;
   END LOOP;
```

Pathod Software

```
END;
/
```

**EXAMPLE 2:**

```
DECLARE
      N NUMBER:=1;
      s varchar2(30):=:s;
      m NUMBER:=1;
BEGIN
      M:=LENGTH(s);
      DBMS_OUTPUT.PUT_LINE(M);
            LOOP
                  DBMS_OUTPUT.PUT_LINE(substr(s,1,N));
                        N:=N+1;
                        EXIT WHEN N>M;
            END LOOP;
 END;
/
```

**2). WHILE LOOP**

SYNTAX:

```
WHILE<CONDITION> LOOP
      STATEMENT BLOCK;
END LOOP;
```

**EXAMPLE:**

```
DECLARE
      N NUMBER:=1;
BEGIN
   WHILE(N<=5) LOOP
         DBMS_OUTPUT.PUT_LINE(N);
         N:=N+1;
      END LOOP;
END;
/
```

**3). FOR LOOP**

SYNTAX:

```
FOR <VARIABLE> [IN REVERSE] <START>..<END>LOOP
      <STATEMENT>;
```

Rathod Software

```
END LOOP;
```

**EXAMPLE 1:**

```
DECLARE
N NUMBER:=1;
I NUMBER:=1;
BEGIN
FOR I IN N..5 LOOP
   DBMS_OUTPUT.PUT_LINE(N);
   N:=N+1;
END LOOP;
END;
/
```

**EXAMPLE 2:**

```
DECLARE
     N NUMBER:=5;
     B1 NUMBER;
BEGIN
     FOR B1 IN 1..N LOOP
          DBMS_OUTPUT.PUT_LINE(B1);
          N:=N+1;
     END LOOP;
END;
/
```

**EXAMPLE 3:**

```
     DECLARE
          N NUMBER;
          B1 NUMBER;
          S VARCHAR2(10):=:S;
     BEGIN
          N:=LENGTH(S);
          FOR B1 IN 1..N LOOP
               DBMS_OUTPUT.PUT_LINE(SUBSTR(S,1,B1));
          N:=N+1;
          END LOOP;
     END;
     /
```

## GOTO STATEMENT

Rathod Software

GOTO STATEMENT IS USED TO SKIP ANY STATEMENT BLOCK.

**SYNTAX:**

```
GOTO <LABEL NAME>;

<<LABEL NAME>>
```

**EXAMPLE:**

```
DECLARE
      N NUMBER;
BEGIN
      N:=:N;
      IF N<0 THEN
        GOTO L1;
      ELSE
        DBMS_OUTPUT.PUT_LINE('NOT NEGATIVE');
      END IF;
      <<L1>>
      DBMS_OUTPUT.PUT_LINE('NUMBER IS ENTERED'||N);
END;
/
```

Rathod Software

# ERROR HANDLING ON PL/SQL BLOCK

There are three methods to handle error in PL/SQL block.

1). Named exceptions (to handle named exceptions).
2). User named exception for I/O validation (to handle numbered exception).
3). User defined exceptions for business rule violation (to handle errors according to user's choice).

## EXAMPLE OF ORACLE'S NAMED EXCEPTION.

```
DECLARE
NO EMP.ENO %TYPE;
NAME EMP.ENAME %TYPE;
SALARY EMP.SALARY %TYPE;
BEGIN
 NO:=:NO;
SELECT ENAME,SALARY INTO NAME,SALARY
FROM EMP WHERE ENO=NO;

DBMS_OUTPUT.PUT_LINE('FOR EMPNO '||NO||' NAME AND SALARY IS
'||NAME||','||SALARY);
EXCEPTION

WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('RECORD WITH GIVEN NO IS NOT FOUND...');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('TOO MANY ROWS');
WHEN INVALID_CURSOR THEN
DBMS_OUTPUT.PUT_LINE('INVALID CURSOR');
WHEN ZERO_DIVIDE THEN
DBMS_OUTPUT.PUT_LINE('ZERO_DIVIDE');
WHEN DUP_VAL_ON_INDEX THEN
DBMS_OUTPUT.PUT_LINE('DUPLICATE VALUE ON INDEX...');
END;
```

Nepal Software

# EXAMPLE OF USER NAMED EXCEPTION

**NOTE:**

**IN FOLLOWING EXAMPLE WE WRITE PRAGMA EXCEPTION_INIT IT IS ONE TYPE OF PACKAGE SO THAT MESSAGE WILL CONVERT IN ERROR MESSAGE.**

```
DECLARE
 UNI_KEY_VIOLATED EXCEPTION;
 PRAGMA EXCEPTION_INIT(UNI_KEY_VIOLATED,-00001);
BEGIN
 INSERT INTO DEPT VALUES(&DEPTNO,'&DNAME','&LOC');
EXCEPTION
 WHEN UNI_KEY_VIOLATED THEN
 DBMS_OUTPUT.PUT_LINE('YOU HAVE ENTERED A VALUE IN UNIQUE OR
PRIMARY KEY FIELD');
END;
/
```

# EXAMPLE OF USER DEFINE EXCEPTION

```
DECLARE
MY_EXCEPTION EXCEPTION;
NO NUMBER(10);
SAL1 NUMBER(14,2);
INC NUMBER(5);
BEGIN
NO:=:NO;
INC:=:INC;
SELECT SALARY INTO SAL1 FROM EMP WHERE ENO=NO;
SAL1:=SAL1+INC;
IF SAL1<5000 THEN
UPDATE EMP SET SALARY=SAL1 WHERE ENO=NO;
DBMS_OUTPUT.PUT_LINE('RECORD UPDATED...');
ELSE
RAISE MY_EXCEPTION;
END IF;
COMMIT;
EXCEPTION
WHEN MY_EXCEPTION THEN
```

Method Software

```
DBMS_OUTPUT.PUT_LINE('SALARY CAN NOT BE UPDATED FOR THIS
EMPLOYEE');
END;
```

Rathod Software

# CURSOR

Cursor

Cursor is a resultant set of data (record set)

Type of cursor

**1).Implicit cursor (global cursor / in built cursor).**

In implicit cursor we must write SQL before its attribute.

Cursor Attributes.

1) SQL%ISOPEN
2) SQL%FOUND
3) SQL%NOTFOUND
4) SQL%ROWCOUNT

1).SQL%ISOPEN

Checks cursor is opened or not.

2).SQL%FOUND

Checks if record set is found or not.

3).SQL%NOTFOUND

Checks if record set is not found.

4).SQL%ROWCOUNT

It will return number of records in data set as an integer number.

Rathod Software

**EXAMPLE:**

```
DECLARE
 C_SAL NUMBER(8);
 V_E NUMBER:=:V_E;
BEGIN
IF SQL%ISOPEN THEN
  DBMS_OUTPUT.PUT_LINE('IMPLICIT CURSOR IS OPEN');
ELSE
  DBMS_OUTPUT.PUT_LINE('IMPLICIT CURSOR IS not OPEN');
END IF;
 SELECT SALARY INTO C_SAL FROM EMP WHERE ENO=V_E;
 DBMS_OUTPUT.PUT_LINE('SALARY IS='||C_SAL);
IF NOT SQL%ISOPEN THEN
 DBMS_OUTPUT.PUT_LINE('IMPLICIT CURSOR IS CLOSED');
END IF;
END;
```

# IMPLICIT CURSOR WITH FOR LOOP

EXAMPLE:

```
DECLARE
 E EMP%ROWTYPE;
dno number(8):=:dno;
BEGIN
 FOR E IN (SELECT * FROM EMP WHERE DNO=dno) LOOP

   DBMS_OUTPUT.PUT_LINE('-                                    -');
    DBMS_OUTPUT.PUT_LINE('ENO='||E.ENO);
    DBMS_OUTPUT.PUT_LINE('ENAME='||E.ENAME);
    DBMS_OUTPUT.PUT_LINE('SAL='||E.SALARY);


 END LOOP;
END;
```

**2).Explicit cursor (created by user explicitly)**

Rathod Software

Cursor Attributes.

5) %ISOPEN
6) %FOUND
7) %NOTFOUND
8) %ROWCOUNT

1).%ISOPEN

Checks cursor is opened or not.

2).%FOUND

Checks if record set is found or not.

3).%NOTFOUND

Checks if record set is not found.

4).%ROWCOUNT

It will returns number of records in data set as an integer number.

* Explicit cursor

This cursor can be accessed by using specific name of the cursor created by the user.

Example:
Mycursor % found

Syntax:

```
DECLARE
    <declaration of other variables>
    Cursor <cursorname>
    <SQL statement>;
BEGIN
    OPEN <cursor name>;
    If <cursor name> %ISOPEN THEN
        LOOP
        FATCH<cursorname> INTO <LISTOF VARIABLE>;
        EXIT WHEN <CURSORNAME>%NOTFOUND;
        IF <cursorname> %FOUND THEN
```

Rathod Software

```
                <SQL statements>
            ELSE
                <USER MESSAGE>
            ENDIF;
        END LOOP;
        ELSE
            <USERMESSAGE ABOUT CLOSE CURSOR>
        END IF;
            CLOSE <CURSOR NAME>;
    END;
```

**EXAMPLE 1:**

```
DECLARE
N NUMBER;
CURSOR C IS SELECT * FROM EMP;
A C%ROWTYPE;
BEGIN
OPEN C;
LOOP
FETCH C INTO A;
EXIT WHEN C%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('-----------------------------------------------');
DBMS_OUTPUT.PUT_LINE('Emp NO IS='||A.ENO);
DBMS_OUTPUT.PUT_LINE('Emp NO IS='||A.ename);
DBMS_OUTPUT.PUT_LINE('Emp NO IS='||A.salary);
DBMS_OUTPUT.PUT_LINE('Emp NO IS='||A.dno);
DBMS_OUTPUT.PUT_LINE('-----------------------------------------------');
END LOOP;
CLOSE C;
END;/
```

**EXAMPLE 2:**

```
DECLARE
    CURSOR C IS SELECT * FROM EMP;
    N EMP%ROWTYPE;
BEGIN
    OPEN C;
        IF C%ISOPEN THEN
        LOOP
            FETCH C INTO N;
                EXIT WHEN C%NOTFOUND;
            IF C%FOUND THEN
                Dbms_output.put_line('no is ='||N.ENO);
                Dbms_output.put_line('name is ='||N.ENAME);
                Dbms_output.put_line('sal is ='||N.SALARY);
            END IF;
        END LOOP;
```

Rathoo Software

```
        END IF;
    CLOSE C;
END;
/
```

## EXAMPLE 3:

```
DECLARE
CURSOR C IS SELECT * FROM EMP WHERE SALARY>=17000;
N EMP%ROWTYPE;
T_DNAME VARCHAR2(50);
BEGIN
OPEN C;
IF C%ISOPEN THEN
LOOP
FETCH C INTO N;
EXIT WHEN C%NOTFOUND;
IF C%FOUND THEN
Dbms_output.put_line('-              -');
Dbms_output.put_line('no is ='||N.ENO);
Dbms_output.put_line('no is ='||N.ENAME);
Dbms_output.put_line('no is ='||N.SALARY);
Dbms_output.put_line('no is ='||N.DNO);
SELECT DNAME INTO T_DNAME  FROM DEPT WHERE DNO=N.DNO;
Dbms_output.put_line('no is ='||T_DNAME);
Dbms_output.put_line('------------------');
END IF;
END LOOP;
END IF;
CLOSE C;
END;/
```

## Example 4:

```
DECLARE
    N NUMBER;
    CURSOR C IS SELECT * FROM EMP;
    A C%ROWTYPE;
    s number;
    t_dname varchar(25);
BEGIN
    OPEN C;
    LOOP
    FETCH C INTO A;
    EXIT WHEN C%NOTFOUND;
        if a.salary<=17000 then
            s:=a.salary *10/100;
        elsif a.salary >=20000 and a.salary <=30000 then
            s:=a.salary *15/100;
        end if;
        DBMS_OUTPUT.PUT_LINE('-----------------------------------------------');
        DBMS_OUTPUT.PUT_LINE('NO IS='||A.ENO);
        DBMS_OUTPUT.PUT_LINE('Name IS='||A.ENAME);
```

Pathod Software

```
            DBMS_OUTPUT.PUT_LINE('Salary IS='||A.salary);
            DBMS_OUTPUT.PUT_LINE('Com. is='||s);
            Select dname into t_dname from dept where dno=a.DNO;
            DBMS_OUTPUT.PUT_LINE('Department IS='||t_dname);
            update emp set co=s where eno=a.eno;
            DBMS_OUTPUT.PUT_LINE('-----------------------------------------------');

    END LOOP;
    CLOSE C;
END;
```

**Example -5 :**

```
DECLARE
    CURSOR D IS SELECT * FROM DEPT;
    CURSOR C IS SELECT * FROM EMP;
    N EMP%ROWTYPE;
    M DEPT%ROWTYPE;
BEGIN
    OPEN C;
    OPEN D;
    IF C%ISOPEN THEN
            IF D%ISOPEN THEN
                    LOOP
                            FETCH C INTO N;
                            FETCH D INTO M;
                                    EXIT WHEN C%NOTFOUND;
                            IF D%FOUND THEN
                                    Dbms_output.put_line('no is ='||M.DNO);
                                    Dbms_output.put_line('name is ='||M.DNAME);
                                    Dbms_output.put_line('-                    -');
                                    Dbms_output.put_line('-                    -');
                            END IF;
                            IF C%FOUND THEN
                                    Dbms_output.put_line('no is ='||N.ENO);
                                    Dbms_output.put_line('name is ='||N.ENAME);
                                    Dbms_output.put_line('sal is ='||N.SALARY);
                                    Dbms_output.put_line('-                    -');
                            END IF;
                    END LOOP;
            END IF;
    END IF;
    CLOSE C;
    CLOSE D;
END;
/
```

**Example 6:**

```
DECLARE
CURSOR D IS SELECT * FROM DEPT;
CURSOR C IS SELECT * FROM EMP;
N EMP%ROWTYPE;
```

Python Software

```
M DEPT%ROWTYPE;
BEGIN
    OPEN C;
    IF C%ISOPEN THEN

                LOOP
                    FETCH C INTO N;
                    EXIT WHEN C%NOTFOUND;
                    IF C%FOUND THEN
                        Dbms_output.put_line('no is ='||N.ENO);
                        Dbms_output.put_line('name is ='||N.ENAME);
                        Dbms_output.put_line('sal is ='||N.SALARY);
                       OPEN D;
                        IF D%ISOPEN THEN
                        LOOP
                            FETCH D INTO M;
                            EXIT WHEN D%NOTFOUND;
                            IF N.DNO=M.DNO THEN
                                IF D%FOUND THEN
                                    Dbms_output.put_line('name is ='||M.DNO);
                                    Dbms_output.put_line('name is
='||M.DNAME);
                                END IF;
                            END IF;
                        END LOOP;
                        CLOSE D;
                        END IF;
                        Dbms_output.put_line('----------------------------------------
--------');
                    END IF;
                END LOOP;

    END IF;
    CLOSE C;

END;
```

## Cursor WITH for loops

Another technique commonly used to control the loop…end loop within a PL/SQL block is the **FOR** variable **IN** value construct.

This is an example of a machine defined loop exit i.e. when all the values in the **FOR** construct are exhausted looping stops.

**SYNTAX:**

**FOR memory variable IN cursorname LOOP**

**} STATEMENT BLOCK**

Rathod Software

**END LOOP;**

A cursor for loop automatically does the following:
* Implicitly declares its loop index as a %rowtype record.
* Opens a cursor
* Fetches a row from the cursor for each loop
* Closes the cursor when all rows have been processed

**EXAMPLE 1:**

```
DECLARE
   CURSOR EMP_CUR IS SELECT * FROM EMP WHERE SAL<=2000;
BEGIN
   FOR I IN EMP_CUR LOOP
        DBMS_OUTPUT.PUT_LINE('-                -');
        DBMS_OUTPUT.PUT_LINE('NO IS='||I.EMPNO);
        DBMS_OUTPUT.PUT_LINE('NO IS='||I.ENAME);
        DBMS_OUTPUT.PUT_LINE('NO IS='||I.SAL);
        DBMS_OUTPUT.PUT_LINE('NO IS='||I.JOB);
        DBMS_OUTPUT.PUT_LINE('-                -');
   END LOOP;
END;
/
```

**EXAMPLE 2:**

```
CREATE TABLE EMP_BACKUP1
(ENO NUMBER(10),
ENAME VARCHAR2(25),
SAL NUMBER(10,2))
/

   DECLARE
   CURSOR EMP_CUR IS SELECT * FROM EMP WHERE SAL<=2000;
   BEGIN
   FOR I IN EMP_CUR LOOP
    INSERT INTO EMP_BACKUP1
   VALUES(I.EMPNO,I.ENAME,I.SAL);
```

Rathod Software

```
 UPDATE EMP SET SAL=SAL+100 WHERE SAL=I.SAL;
END LOOP;
END;
/
```

Rathod Software

**Parameterized cursor.**

Till now, all the cursors that have been declared and used fetch a pre-determined set of records. Records, which satisfy conditions, set in the WHERE clause of the SELECT statement mapped to the cursor. In other words, the criterion on which the ACTIVE Data Set is determined is hard coded and never changes.

**Declaring A Parameterized cursor**

**Syntax:**

**Cursor cursorname (variablename datatype)**
**Is <select statement…>**

**Example:**

```
DECLARE
 CURSOR C1 IS SELECT * FROM EMP;
 CURSOR C2 (NAME VARCHAR2) IS SELECT NAME FROM EMP_BACKUP1 WHERE ENAME=NAME;
 N C1%ROWTYPE;
 NM C2%ROWTYPE;
 BEGIN
 OPEN C1;
   LOOP
   FETCH C1 INTO N;
        EXIT WHEN C1 %NOTFOUND;
        OPEN C2(N.ENAME);
             FETCH C2 INTO NM;
             IF C2 %FOUND THEN
                  DBMS_OUTPUT.PUT_LINE ('RECORD WITH '||N.EMPNO||'AND'||N.ENAME||'IS
EXISTED');
             ELSE
                  INSERT INTO EMP_BACKUP1 VALUES(N.EMPNO,N.ENAME,N.SAL);
                  DBMS_OUTPUT.PUT_LINE ('RECORD Added with name='||N.ENAME);
             END IF;
        CLOSE C2;
   END LOOP;
 CLOSE C1;
 COMMIT;
 end;
/
```

Rathod Software

**TRIGGER**

**NOTE:**

**WE MUST SET SERVEROUTPUT ON WHEN WE ARE EXECUTING TRIGGER.**

Trigger is used to declarative constraints can be used to constrain data input. However both have significant difference as mentioned below:

1). A declarative integrity constraint is a statement about Database that is always true. A constraint applies it existing data in the table and any statement that manipulates the table.

2). Triggers constrain what a transaction can do. A trigger does not apply to data loaded before the trigger was created, so it does not guarantee all data in table conforms to the rules established by an associated triggers.

3). A trigger enforces a transitional constraint, which Cannot be enforced by a declarative integrity Constraint.

Rathod Software

Trigger has basic three parts:
1). A triggering event or statement.
2). A trigger restriction
3). A trigger action

**Types of triggers:**
**1). Row trigger (restriction)**
**2). Statement trigger (restriction)**
**3). before trigger (events)**
**4). after trigger (events)**

**Syntax:**

```
CREATE OR REPLACE TRIGGER<TRIGGERNAME>
BEFORE/AFTER UPDATE OR DELETE OR INSERT
OF<COLUMNNAME>
ON <TABLENAME>
[FOR EACH ROW]
[REFERENCING OLD AS <OLD>
NEW AS <NEW>]
DECLARE
      <VARIABLE> DECLARATION;
BEGIN
      <ACTION STATRS>
EXCEPTION
      <EXCEPTION DETAILS>
END;
```

Rathod Software

**EXAMPLE 1:**

```
CREATE OR REPLACE TRIGGER TRIG1
BEFORE UPDATE OR DELETE OR INSERT
ON DEPT
BEGIN
 RAISE_APPLICATION_ERROR(-20000,'TRANSACTION CAN NOT POSSIBLE READ
ONLY');
END;
/
```

```
CREATE TABLE LOGIN1
(USERNAME VARCHAR(25),
PASSWORD VARCHAR(25));
```

**EXAMPLE 2:**

```
CREATE OR REPLACE TRIGGER TR2 AFTER UPDATE OR INSERT ON LOGIN1
BEGIN
DBMS_OUTPUT.PUT_LINE('RECORD UPDATED SUCCESSFULLY...');
END;
/
```

**EXAMPLE 3:**

**This example is used for row trigger**

```
CREATE OR REPLACE TRIGGER TR3 AFTER DELETE ON LOGIN1 FOR EACH ROW
 BEGIN
 DBMS_OUTPUT.PUT_LINE('RECORD DELETED SUCCESSFULLY...');
 END;
 /
```

Rathod Software

**NESTED TABLE**

WE CAN USE NESTED TABLE FOR PARENT AND CHILD RELATIONSHIP WE CAN SOLVE THIS PROBLEM BY USING IT.

**SYNTAX:**

```
CREATE TYPE <TYPE NAME> AS OBJECT
(<FIELD NAME> <FIELD DATA TYPE>(SIZE),
<FIELD NAME> <FIELD DATA TYPE>(SIZE)..);
```

**EXAMPLE:**

```
CREATE TYPE A AS OBJECT
(AREA VARCHAR2(25),
STRTNO VARCHAR2(3),
PINCODE VARCHAR2(10),
CONO NUMBER(15));

CREATE TABLE STUD_INFO1
(RNO NUMBER(3) PRIMARY KEY,
NAME VARCHAR2(25) NOT NULL,
ADDRESS A);

INSERT INTO STUD_INFO1 VALUES(1,'ABC',A('ASDF','1','361440'))
/
```

Rathod Software

**PROCEDURE**

PROCEDURE IS A PL/SQL BLOCK. WE CAN STORE PROCEDURE IN DATABASE STORAGE PROPERTY SO THAT WE CAN USE THIS PROCEDURE IN MULITIPLE APPLICATION.

**SYNTAX:**

```
CREATE OR REPLACE PROCEDURE <PROCEDUE NAME>
(PARAMETER [IN/OUT/IN OUT] DATA TYPE)
[IS/AS]
```

**EXAMPLE:**

```
CREATE OR REPLACE PROCEDURE PRINT(X VARCHAR2) AS
BEGIN
   DBMS_OUTPUT.PUT_LINE(X);
END;

BEGIN
   PRINT(SYSDATE);
END;
/

BEGIN
PRINT('THIS IS FOR TESTING');
END;
/
```

Rathod Software

**IT HAS THREE PARAMETER'S TYPE;**
**1). IN**
**2). OUT**
**3). IN OUT**

**1). IN**

WE CAN USE IN PARAMETER TYPE WHEN WE DON'T KNOW ABOUT DATA TYPE.

**2). OUT**

WE CAN USE OUT PARAMETER TYPE WHEN WE WANT TO RETURN ANY DETAILS OR OUTPUT.

**3). IN OUT**

THIS PARAMETER IS WORK'S LIKE IN AND OUT BOTH.

Rathod Software

# EXAMPLE 1:

```
CREATE OR REPLACE PROCEDURE FIND_SAL
 (ENO IN NUMBER)
 IS
 VSAL NUMBER;
 BEGIN
 SELECT SAL INTO VSAL FROM EMP WHERE EMPNO=ENO;
 UPDATE EMP SET SAL=VSAL+100 WHERE EMPNO=ENO;
 PRINT('THE SAL IS='||VSAL);
 END;
 /


DECLARE
   N NUMBER:='&N';
BEGIN
   FIND_SAL(N);
END;
/
```

# EXAMPLE 2:

```
CREATE OR REPLACE PROCEDURE ADD_FUN1
(V1 IN NUMBER,V2 IN NUMBER,V3 IN NUMBER)
IS
ANS NUMBER;
BEGIN
ANS:=V1+V2+V3;
PRINT(ANS);
END;

DECLARE
   N NUMBER:='&N';
   N1 NUMBER:='&N1';
   N2 NUMBER:='&N2';
BEGIN
   ADD_FUN1(N,N1,N2);
END;
/
```

# EXAMPLE 3:

```
CREATE OR REPLACE PROCEDURE ADD_FUN11
(V1 IN NUMBER,V2 IN NUMBER,V3 IN NUMBER, ANS OUT NUMBER)IS
ANS1 NUMBER;
BEGIN
ANS1:=V1+V2+V3;
```

Rahod Software

```
ANS:=ANS1;
END;

DECLARE
ANS NUMBER;
BEGIN
ADD_FUN11(&V1,&V2,&V3,ANS);
PRINT('THE OUTPUT IS='||ANS);
END;
/
```

Rathod Software

# FUNCTON

FUNCTON IS A PL/SQL BLOCK. WE CAN STORE FUNCTON IN DATABASE STORAGE PROPERTY SO THAT WE CAN USE THIS FUNCTON IN MULITIPLE APPLICATION.

**SYNTAX:**

CREATE OR REPLACE FUNCTON <FUNCTION NAME>
(PARAMETER [IN/OUT/IN OUT] DATA TYPE)
RETURN <DATATYPE>[IS/AS]

**IT HAS THREE PARAMETER'S TYPE;**
**1). IN**
**2). OUT**
**3). IN OUT**

**1). IN**

WE CAN USE IN PARAMETER TYPE WHEN WE DON'T KNOW ABOUT DATA TYPE.

**2). OUT**

WE CAN USE OUT PARAMETER TYPE WHEN WE WANT TO RETURN ANY DETAILS OR OUTPUT.

**3). IN OUT**

THIS PARAMETER IS WORK'S LIKE IN AND OUT BOTH.

Rathod Software

# EXAMPLE 1:

```
CREATE OR REPLACE FUNCTION ADD_FUN(V1 IN NUMBER,V2 IN NUMBER,V3 IN
NUMBER)
RETURN NUMBER IS
ANS NUMBER;
BEGIN
ANS:=V1+V2+V3;
RETURN ANS;
END;
/


DECLARE
A NUMBER;
BEGIN
A:=ADD_FUN(&V1,&V2,&V3);
DBMS_OUTPUT.PUT_LINE('SUM IS='||A);
END;
/
```

# EXAMPLE 2:

```
CREATE OR REPLACE FUNCTION FIND(N IN NUMBER,NAME OUT VARCHAR2)
RETURN NUMBER IS
SAL1 NUMBER;
BEGIN
SELECT SAL,ENAME INTO SAL1,NAME FROM EMP
WHERE EMPNO=N;
RETURN SAL1;
END;
/

DECLARE
A NUMBER;
NAME VARCHAR2(30);
BEGIN
A:=FIND(&N,NAME);
DBMS_OUTPUT.PUT_LINE('NAME IS='||NAME);
DBMS_OUTPUT.PUT_LINE(‘SALIS='||A);
END;
/
```

Method Software

# CREATING NEW USERS.

**SYNTAX:**

CREATE USER <USER NAME>
IDENTIFIED BY <PASSWORD>;

EXAMPLE:

CREATE USER OPERATOR IDENTIFIED BY ASDF;

# CONNECTING AS USER.

**SYNTAX:**

CONNECT<USER NAME>

**EXAMPLE:**

CONNECT
->USERNAME :OPERATOR
->PASSWORD :ASDF

# GRANT COMMAND.

**SYNTAX:**

GRANT<PRIVILEGES> ON <TABLE NAME>
<USERNAME>[WITH GRANT OPTION];

**EXAMPLE:**

GRANT SELECT ON EMP TO VIJAY;
GRANT ALL ON EMP TO VIJAY;

Ratnod Software

# REVOKE COMMAND.

SYNTAX:

REVOKE <PRIVILEGES> ON <TABLE NAME>
FROM <USERNAME>;

EXAMPLE:

REVOKE SELECT ON EMP FROM VIJAY;

Rathod Software

# VARRAY

We can use varray when we have same data types in different fields in nested table.

**SYNTAX:**

```
CREATE TYPE <TYPENAME> AS
VARRAY (<ARRAYSIZE>) OF <DATATYPE> (SIZE);
```

**EXAMPLE:**

```
CREATE TYPE MARKS AS VARRAY(3) OF NUMBER(7);
```

**CREATING TABLE WITH VARRAY:**

```
CREATE TABLE STUD_MARKS1
(RNO NUMBER(3),
NAME VARCHAR2(25),
SCORE MARKS);
```

**INSERTING DATA INTO TABLE.**

```
INSERT INTO STUD_MARKS1
VALUES(1,'GOHIL',MARKS(45,55,65))
/

INSERT INTO STUD_MARKS1
VALUES(&RNO,'&NAME ',MARKS(&M1,&M2,65))
/
```

Rathod Software

## PACKAGES

Package is used to group any procedures or any function.

Example:

```
CREATE OR REPLACE PACKAGE PACK1 IS
PROCEDURE MSG(X VARCHAR2);
FUNCTION SPELL (N NUMBER) RETURN VARCHAR2;
END PACK1;
/

CREATE OR REPLACE PACKAGE BODY PACK1 IS
PROCEDURE MSG (X VARCHAR2) IS
BEGIN
      DBMS_OUTPUT.PUT_LINE(X);
END MSG;

FUNCTION SPELL (N NUMBER) RETURN VARCHAR2 IS
BEGIN
RETURN TO_CHAR(TO_DATE(N,'J'),'JSP');
END SPELL;
END PACK1;
```

**Execution of package.**

**1)**

```
DECLARE
    N NUMBER:=&N;
    N1 NUMBER:=&N1;
    BEGIN
    PACK1.MSG(N+N1);
    END;
   /
```

**2).**

```
SQL> var a varchar2(200)
SQL> exec:a:=pack1.spell(121)

PL/SQL procedure successfully completed.

SQL> print a
```

Pathod Software

# PARTITION

PARTITION IS USED TO DEVIDE TABLE.

**SYNTAX:**

```
CREATE TABLE <TABLE NAME> <DEFINATION>
PARTITION BY RANGE(COLUMN NAME)
(PARTITION <PARTITION NAME>
VALUE LESS THAN <VALUE>,
PARTITION <PARTITION NAME>
VALUE LESS THAN<VALUE>);
```

**EXAMPLE:**

```
CREATE TABLE ITEM
(ITEMNO NUMBER(6),
ITEM_NAME VARCHAR2(30),
ITEM_PRICE NUMBER(10,2))
PARTITION BY RANGE(ITEMNO)
(PARTITION P1 VALUES LESS THAN (100),
PARTITION P2 VALUES LESS THAN (200),
PARTITION P3 VALUES LESS THAN(MAXVALUE))
/
```

```
CREATE TABLE invoices
 (invoice_no    NUMBER NOT NULL,
  invoice_date  DATE   NOT NULL,
  comments      VARCHAR2(500))
 PARTITION BY RANGE (invoice_date)
 (PARTITION invoices_q1 VALUES LESS THAN (TO_DATE('01/04/2001', 'DD/MM/YYYY')),
  PARTITION invoices_q2 VALUES LESS THAN (TO_DATE('01/07/2001', 'DD/MM/YYYY')) ,
  PARTITION invoices_q3 VALUES LESS THAN (TO_DATE('01/09/2001', 'DD/MM/YYYY')) ,
  PARTITION invoices_q4 VALUES LESS THAN (TO_DATE('01/01/2002', 'DD/MM/YYYY')))
 /
```

**Describe command:**

```
describe emp;
```

Rahul Software