Reference Manual

for

Database Systems Lab

School of Computing Science and Engineering

VIT®
UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)

Vellore-632014,Tamil Nadu

Faculty Associated :

Prof. M.Venkatesan,	A.P(SG)
Prof. Hari Seetha,	A.P(SG)
Prof. K.Govinda,	A.P(SG)
Prof. E.Anupriya,	A.P(SG)
Prof. L.Ramanathan,	A.P
Prof. M.Anbarasi,	A.P
Prof. J.N.Swathi,	A.P
Prof. A.Geetha Mary,	A.P
Prof. K.Sharmila Bhanu,	A.P
Prof. S.Arun Kumar,	A.P
Prof. G.Guru Raghav,	A.P

CONTENTS

SQL STATEMENTS

SQL statements are classified as follows:

Data Retrieval Statement:

SELECT is the data extracting statement which retrieves the data from the database.

Data Manipulation Language (DML):

This language constitutes the statements that are used to manipulate with the data. It has three commands, which are INSERT, UPDATE and DELETE.

Data Definition Language (DDL):

This is the language used to define the structure of the tables. It sets up, changes, and removes data structures from the tables. It uses 5 commands, which are CREATE, ALTER, DROP, RENAME and TRUNCATE.

Data Transaction Language (DTL):

This is the language used to do undo and redo the transaction performed in the database. The commands are Commit, Rollback, and Save Point

Data Control Language:

This language is used to sanction the rights to the users to use the other user's database objects. The commands are Grant and Revoke

Consider the following schema based on which the example queries are discussed in this manual.

BASE SCHEMA

EMPLOYEE

```
  Name                         Type
--------------------------   ----------------------
EMPLOYEE_ID                  NUMBER(3)
FIRST_NAME                   VARCHAR2(10)
LAST_NAME                    VARCHAR2(10)
MGR                          NUMBER(4)
HIRE_DATE                    DATE
JOB_ID                       VARCHAR2(10)
SALARY                       NUMBER(10)
COMMISION                    NUMBER(8)
DEPTNO                       NUMBER(2)
```

DEPARTMENT

```
Name                    Type
---------------       -----------------
DEPTNO                 NUMBER(2)
DNAME                 VARCHAR2(14)
LOC                    VARCHAR2(13)
```

BONUS

```
Name                      Type
----------------       -------------------
ENAME                  VARCHAR2(10)
JOB                    VARCHAR2(9)
SAL                     NUMBER(10,2)
COMM                    NUMBER(10)
```

JOBGRADE

```
Name                      Type
------------------      ---------------------
JOB_ID                VARCHAR2(10)
GRADE                  NUMBER
LOSAL                  NUMBER
HISAL                  NUMBER
```

DATA TYPES IN ORACLE:

Data Type	**Description**
VARCHAR2(size)	Variable-length character data
CHAR(size)	Fixed-length character data
NUMBER(p,s)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data up to 2 gigabytes
CLOB	Character data up to 4 gigabytes
RAW and LONG RAW	Raw binary data
BLOB	Binary data up to 4 gigabytes
BFILE	Binary data stored in an external file; up to 4 gigabytes

ROWID	A 64 base number system representing the unique address of a row in its table

ORACLE 9I TABLE STRUCTURES

- Table can be created at any time
- No need to specify the size of table, the size is ultimately defined by the amount of space allocated to the database as a whole.
- Tables can have up to 1000 columns

NAMING RULES

Table names and Column names

- Must begin with a letter
- Must be 1-30 characters long
- Must contain only A-Z,a-z,0-9,_,$,#
- Must not duplicate the name of another object owned by the same user
- Must not be a reserved word

<u>Data Definition Language (DDL)</u>

The following are the DDL Commands:

1. Create 2. Alter 3. Drop 4. Truncate 5. Rename

1. a. Creating a table

Syntax:

```
Create table <Table Name>
( <Field1> <Data Type> <(width) <constraints> ,
  <Field2> <Data Type> <(width)> <constraints>,
..................................);
```

Example:

```
SQL> create table employee
```

```
( employee_id number(3),
first_name varchar2(10),
last_name varchar2(10),
mgr number(4),
hire_date date,
job_id varchar2(10),
salary number(10),
commision number(8),
deptno number(2));
```

Output:

```
Table created.
```

Example:

```
SQL> create table department
(deptno number(2),
 dname varchar(14),
  loc varchar(13));
```

Output:

```
Table created.
```

<u>Note:</u>

Other tables can be created in the similar way.

b. To view the Structure of the table, desc command is used

```
SQL> desc employee;

 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
EMPLOYEE_ID                                         NUMBER(3)
```

```
FIRST_NAME                    VARCHAR2(10)
LAST_NAME                     VARCHAR2(10)
MGR                           NUMBER(4)
HIRE_DATE                     DATE
JOB_ID                        VARCHAR2(10)
SALARY                        NUMBER(10)
COMMISION                     NUMBER(8)
DEPTNO                        NUMBER(2)
```

2. Alter Table Statement:

Alter command is used to perform the following action on the table:

a. Adding column in the existing table
b. Increasing and decreasing the column size and changing data types
c. Dropping column
d. Renaming the column
e. Adding and dropping constraints to the table(discussed in constraints topics)
f. Enabling & disabling constraints in the table(discussed in constraints topics)

a. To Add a column to the table (structure)

Add option is used to add a new column

Syntax:

Alter Table <Table-Name> Add <Field Name> <Type> (width);

Example:

```
SQL> alter table employee add address varchar2 (20);
```

Output:

```
Table altered.
```

b. To Modify a field of the table

- Increase the width or precision of numeric column
- Increase the width of numeric or character columns
- Decrease the width of the column only if the column contains only null values or if the table has no rows
- Change the data type only if the column contains null values

Syntax:

Alter Table <tablename> MODIFY (<column name > < newdatatype>);

Example:

SQL> alter table employee modify address varchar2 (10);

Output:

Table altered.

c. To Drop a field of the table

Drop option is used to delete a column or remove a constraint

Syntax:

Alter Table <tablename> DROP COLUMN < column name>;

Example:

SQL> alter table employee drop column address;

Output:

Table altered.

d.To rename a column

Syntax:

ALTER TABLE <tablename> RENAME COLUMN <oldcolumnname> TO

<newcolumn name>

Example:

SQL> alter table employee rename column mgr to manager;

Output:

Table altered.

- **To Drop a table - Deletes a Table along with all contents**

Syntax:

Drop Table <Table-Name>;

Example:

Drop Table Student_table;

Output:

Table Dropped

- **To Truncate a table - Deletes all rows from a table ,retaining its structure**

Syntax: Truncate Table <tablename>

Example:

```
SQL> truncate table employee;
```

Output:

```
Table truncated.
```

g. To rename a table- Renames a table with new name

Syntax:

Rename <oldtablename> To <newtablename>

Example:

```
SQL> rename employee to emp;
```

Output:

```
Table renamed
```

<u>Data manipulation Language (DML)</u>

The following are the DML Commands: **1. Insert 2. Delete 3. Update 4. Select**

Insert command is used to load data into the table.

a. Inserting values from user

Syntax:

Insert into <tablename> values (val1,val2 …);

Example:

```
SQL> insert into department values(10,'accounts','chennai');
```

Output:

```
1 row created.
```

b. Inserting values for the specific columns in the table

Syntax:

Insert Into <Table-Name> (Fieldname1, Fieldname2, Fieldname3,..) Values (value1, value2, value3,..);

Example:

```
SQL> insert into department (deptno,dname)values(20,'finance');
```

Output:

1 row created.

c. ***Inserting interactively(Inserting ,ultiple rows by using single insert command)***

Syntax:

Insert Into <tablename> Values(&<column name1> , &<column name2> …);

Example:

```
SQL> insert into employee values(&empid,'&fn','&ln',&mgr,'&hdate','&job',&sal,
   &comm,&dept);
Enter value for empid: 111
Enter value for fn: Smith
Enter value for ln: Ford
Enter value for mgr: 222
Enter value for hdate: 21-jul-2010
Enter value for job: J1
Enter value for sal: 30000
Enter value for comm: 0.1
Enter value for dept: 10
old   2: &comm,&dept)
new   2: 0.1,10)
```

Output:

1 row created.

Note: Column names of character and date type should be included with in single quotation.

- **Inserting null values**

Syntax:

Insert Into <tablename> Values (val1,’ ‘,’ ‘,val4);

Example:

```
insert into department  values( ‘101’,’’,chennai);
```

Output:

1 row created.

2. To Delete rows from a table

Syntax:

Delete from <table name> [where <condition>];

Example:

a) to delete all rows:

SQL> delete from department;

Output:

89 rows deleted.

b) conditional deletion:

SQL> delete from department where loc='chennai';

Output:

1 row deleted.

3. Modifying (Updating) Records:

a. Updating single column

Syntax:

UPDATE <table name> Set <Field Name> = <Value> Where <Condition>;

Example:

SQL> update department set loc='Hyderabad' where deptno=20;

Output:

1 row updated.

Note: Without where clause all the rows will get updated.

b. Updating multiple column [while updating more than one column, the column must be separated by comma operator]

Example: SQL> update department set loc='Hyderabad', dname= 'cse' where deptno=20;

Output:

1 row updated.

4. Selection of Records [Retrieving (Displaying) Data:]

Syntax:

Select <field1, field2 …fieldn> from <table name> where <condition>;

Example:

a) SQL> select * from department;

Output:

```
   DEPTNO     DNAME          LOC
---------- -------------- -------------
      10        accounts        chennai
      20        finance         Hyderabad
      30        IT              Bangalore
      40        marketing       chennai
```

Example:

b) SQL> select dname, loc from department;

Output:

```
DNAME          LOC
-------------- -------------
accounts        chennai
finance         Hyderabad
IT              Bangalore
marketing       Chennai
```

- **Using Alias name for a field**

Syntax:

Select <col1> <alias name 1> , <col2> < alias name 2> from < tab1>;

Example:

SQL> select dname, loc as location from department;

Output:

```
DNAME          LOCATION
-------------- -------------
accounts        chennai
finance         Hyderabad
IT              Bangalore
marketing       Chennai
```

- **With distinct clause [Used to retrieve unique value from the column]**

Syntax:

Select distinct <col2> from < tab1>;

Example:

SQL> select distinct loc from department;

Output:

```
LOC
-------------
chennai
Bangalore
Hyderabad
```

- **Creating Table using subquery**

Syntax:

Create table <new _table_name> as Select <column names> from <old_table_name>;

Example:

```
 SQL> create table copyOfEmp as select * from employee;
```

Output:

```
Table created.
```

- **To view the contents of new Table**

```
SQL> select * from copyofemp;
```

Output:

```
EMPLOYEE_ID   FIRST_NAME   LAST_NAME          MANAGER   HIRE_DATE   JOB_ID

SALARY  COMMISION     DEPTNO

 111                Smith              Ford                222             21-JUL-10     J1            30000

0.1                 1 0
```

- **To create a table with same structure as an existing table**

Syntax:

Create table <new _table_name> as Select <column names> from<old_table_name> where 1=2;

Example:

```
create table copyOfEmp2 as select * from employee where 1=2;
```

Output: Table created.

```
SQL> select * from copyofemp2;
```

Output:

```
no rows selected
```

```
SQL> desc copyofemp2;
```

Output:

```
Name                                      Null?    Type
----------------------------------------- -------- -----------------
EMPLOYEE_ID                                        NUMBER(3)
FIRST_NAME                                         VARCHAR2(10)
LAST_NAME                                          VARCHAR2(10)
MANAGER                                            NUMBER(4)
HIRE_DATE                                          DATE
JOB_ID                                             VARCHAR2(10)
SALARY                                             NUMBER(10)
COMMISION                                          NUMBER(8)
DEPTNO                                             NUMBER(2)
```

Note: Only structure of table alone is copied and not the contents.

- **Inserting into table using a subquery**

Syntax :

Insert into <new_table_name> (Select <columnnames> from <old_table_name>);

Example:

SQL> insert into copyofemp2 (select * from employee where employee_id > 100);

Output:

50 rows created.

Constraints

- Constraints enforce rules on the table whenever rows are inserted, updated and deleted from the table.
- Prevents the deletion of a table if there are dependencies from other tables.
- Name a constraints or the oracle server generate name by using SYS_cn format.
- Define the constraints at column or table level. constraints can be applied while creation of table or after the table creation by using alter command.

- View the created constraints from User_Constraints data dictionary.

Constraints Types

CONSTRAINT	**DESCRIPTION**
NOT NULL	Specifies that a column must have some value.
UNIQUE	Specifies that columns must have unique values.
PRIMARY KEY	Specifies a column or a set of columns that uniquely identifies as row. It does not allow null values.
FOREIGN KEY	Foreign key is a column(s) that references a column(s) of a table.
CHECK	Specifies a condition that must be satisfied by all the rows in a table.

1. Creating Constraints without constraint name

Syntax:

```
CREATE TABLE  < tablename>   (
<column name 1>  < datatype>,
<column name 2>  < datatype> UNIQUE ,
<column name 3>  < datatype> ,
 PRIMARY KEY ( <column name2>)
);
```

Example:

```
    CREATE TABLE emp_demo2
     ( employee_id    NUMBER(6) PRIMARY KEY,
      first_name     VARCHAR2(20) NOT NULL,
      last_name      VARCHAR2(25) NOT NULL,
      email          VARCHAR2(25) UNIQUE,
      phone_number   VARCHAR2(20) UNIQUE,
      job_id         VARCHAR2(10),
      salary         NUMBER(8,2) CHECK(SALARY>0),
      deptid  NUMBER(4)
     ) ;
```

2. Creating constraints with constraint name

Syntax:

```
     CREATE TABLE  < tablename1>   (
     <column name 1> < datatype> CONSTRAINT <constraint name1> UNIQUE,
     <column name 2> < datatype> CONSTRAINT <constraint name2> NOT NULL,
     constraint  < constraint name3 >  PRIMARY KEY ( <column name1>),
     constraint  <constraint name4>  FOREIGN KEY (<column name2>)
     REFERENCES  <tablename2>  (<column name1>)
     );
```

Example:

```
    CREATE TABLE emp_demo3
     ( employee_id    NUMBER(6) CONSTRAINT emp_eid PRIMARY KEY,
      first_name     VARCHAR2(20),
      last_name      VARCHAR2(25) CONSTRAINT emp_last_name_nn  NOT NULL,
      email          VARCHAR2(25) CONSTRAINT emp_email_nn  NOT NULL,
      phone_number   VARCHAR2(20),
      job_id         VARCHAR2(10) CONSTRAINT emp_job_nn  NOT NULL,
      salary        NUMBER(8,2) CONSTRAINT  emp_salary_nn  NOT NULL,
      deptid  NUMBER(4), CONSTRAINT emp_dept FOREIGN KEY(deptid)
       REFERENCES department(deptid)  ,
      CONSTRAINT     emp_salary_min CHECK (salary > 0) ,
      CONSTRAINT     emp_email_uk   UNIQUE (email)
        ) ;
```

3. With check constraint

Syntax:

```
   CREATE TABLE  < tablename>   (
   <column name1 >          < datatype> ,
    <column name 2>         < datatype>,
     CHECK  ( < column name 1  > in  ( values) )
   CHECK  ( < column name 2  >  between <val1> and  <val2> ) );
```

Example:

```
     CREATE TABLE emp_demo4
      ( emp_id     NUMBER(6),
       emp_name   VARCHAR2(15),
       salary     NUMBER(10) CHECK (salary between 1000 and 10000)
       );
```

Adding Constriants

Constraints can be added after the table creation by using alter command

Syntax: Add constraints

```
ALTER TABLE <tablename> ADD CONSTRAINT <constraint_name>  constriant_type
(<column name>);
```

Examples:

```
ALTER TABLE emp_demo4 ADD CONSTRAINT con_pk1  PRIMARY KEY(emp_id);
```

ALTER TABLE emp_demo4 **ADD CONSTRAINT** con_emp_uk **UNIQUE(**phoneno);

ALTER TABLE emp_demo4 **ADD CONSTRAINT** con_empfk **FOREIGN KEY(DNO) REFERENCES** department(dno);

ALTER TABLE emp_demo4 **ADD CONSTRAINT** con_emp_ck **CHECK** (salary >0);

ALTER TABLE emp_demo4 **MODIFY (<**Column name> <datatype> **CONSTRAINT** constraint_name **NOT NULL);**

Drop Constraints

Syntax

ALTER TABLE <tablename> DROP CONSTRAINT < constraint name >;

Drop the unique key on the email column of the employees table:

e.g **ALTER TABLE** employees **DROP UNIQUE** (email);

CASCADE Constraints

The CASCADE Constraints clause is used along with the Drop Column Clause.

- A foreign key with a cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted. This is called a cascade delete.
- A foreign key with a cascade delete can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Syntax:

CREATE TABLE table_name

(column1 datatype null/not null,

column2 datatype null/not null,

...

CONSTRAINT fk_column

FOREIGN KEY (column1, column2, ... column_n)

REFERENCES parent_table (column1, column2, ... column_n)

ON DELETE CASCADE

);

Example:

CREATE TABLE supplier

(supplier_id number**(**10**)not null,**

supplier_namevarchar2(50)**not null,**

contact_namevarchar2(50),

CONSTRAINT supplier_pk **PRIMARY KEY (**supplier_id**));**

CREATE TABLE products

(product_id number(10)**not null,**

suppl_id number(10) **not null,**

CONSTRAINT fk_supplier **FOREIGN KEY** (suppl_id) **REFERENCES**

supplier(supplier_id) **ON DELETE CASCADE);**

Because of the cascade delete, when a record with a particular supplier_ id is deleted from supplier table , then all the records of the same supplier_id will be deleted from products table also.

Operators in SQL*PLUS

Type	**Symbol / Keyword**	**Where to use**
Arithmetic	+ , - , * , /	To manipulate numerical column values, WHERE clause
Comparison	=, !=, <, <=, >, >=, between, not between, in, not in, like, not like	WHERE clause
Logical	and, or, not	WHERE clause, Combining two queries

- **Between..And..**

Example:

SQL> select first_name, deptno from employee where salary between 20000 and 35000;

Output:

```
FIRST_NAME    DEPTNO
---------- ----------
Smith                10
```

- **IN**

Example:

SQL> select first_name, deptno from employee where job_id in ('J1','J2');

Output:

```
FIRST_NAME    DEPTNO
---------- ----------
Smith               10
Arun                30
Nithya              10
```

- **NOT IN**

Example:

SQL> select dname,loc from department where loc not in ('chennai','Bangalore');

Output:

```
DNAME        LOC
-------------- -------------
finance      Hyderabad
```

- **Like**

Use the LIKE condition to perform wild card searches of valid search string values.

Search conditions can contain either characters or numbers

% - denotes zero or many characters.

_ - denotes one character.

Example:

```
SQL> select dname,loc from department where loc like 'c%';
```

Output:

```
DNAME          LOC
-------------- -------------
accounts       chennai
marketing      Chennai
```

Example:

```
SQL> select dname,loc from department where loc like 'chen_ _ _';
```

Output:

```
DNAME          LOC
-------------- -------------
accounts       chennai
marketing      Chennai
```

Example:

```
SQL> select dname,loc from department where loc not like 'c%';
```

Output:

```
DNAME          LOC
-------------- -------------
finance        Hyderabad
IT             Bangalore
```

- **Between..and..**

Example:

```
SQL> select first_name, deptno, salary from employee where salary not between 20000 and
35000;
```

Output:

```
FIRST_NAME    DEPTNO    SALARY
---------- ---------- ----------
Arun                30     40000
Nithya              10     45000
```

Note: Inserting null value into location column of department table

Example:

```
SQL> insert into department(deptno,dname) values(40,'Sales');
```

Output:

```
1 row created.
```

- **is Null**

Example:

```
SQL> select * from department where loc is null;
```

Output:

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        40 Sales
```

Example:

```
SQL> select * from department where loc is not null;
```

Output:

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 accounts       chennai
        20 finance        Hyderabad
        30 IT             Bangalore
        40 marketing      chennai
```

LOGICAL OPERATORS: Used to combine the results of two or more conditions to produce a single result. The logical operators are: OR, AND, NOT.

Operator Precedence

- Arithmetic operators-Highest precedence
- Comparison operators
- NOT operator
- AND operator
- OR operator----Lowest precedence

The order of precedence can be altered using parenthesis.

Example:

```
SQL> select first_name, deptno, salary from employee where salary > 20000 ;
```

Output:

```
FIRST_NAME     DEPTNO     SALARY
---------- ---------- ----------
Smith              10      30000
Arun               30      40000
Nithya             10      45000
```

Example:

```
SQL> select first_name, deptno, salary from employee
where salary > 20000 and salary < 35000;
```

Output:

```
FIRST_NAME     DEPTNO     SALARY
---------- ---------- ----------
Smith              10      30000
```

Example:

```
SQL> select first_name, deptno, salary+100 from employee where salary > 35000;
```

Output:

```
FIRST_NAME     DEPTNO SALARY+100
---------- ---------- ----------
Arun               30      40100
```

Example:

```
SQL> update employee set salary = salary+salary*0.1 where employee_id = 111;
```

Output:

```
1 row updated.
```

Example:

```
SQL> select * from department where loc = 'chennai' or dname='IT';
```

Output:

```
   DEPTNO    DNAME         LOC
----------         -------------- -------------
       10          accounts        chennai
       30           IT                  Bangalore
       40          marketing       chennai
```

FUNCTIONS

- Single Row Functions
- Group functions

Single Row Functions

Returns only one value for every row can be used in SELECT command and included in WHERE clause

Types

- Character functions
- Numeric functions
- Date functions

CHARACTER FUNCTIONS:

Character functions accept a character input and return either character or number values. Some of them supported by Oracle are listed below

Syntax	**Description**
initcap (char)	Changes first letter to capital
lower (char)	Changes to lower case
upper (char)	Changes to upper case
ltrim (char, set)	Removes the set from left of char
rtrim (char, set)	Removes the set from right of char
translate(char, from, to)	Translate ‘from’ anywhere in char to ‘to’

replace(char, search string, replace string)	Replaces the search string to new
substr(char, m , n)	Returns chars from m to n length
lpad(char, length, special char)	Pads special char to left of char to Max of length
rpad(char, length, special char)	Pads special char to right of char to Max of length
chr(number)	Returns char equivalent
length(char)	Length of string

Examples:

Function	**Input**	**Output**
Initcap(char)	SQL>select initcap(‘hello’) from dual;	Hello
Lower(char)	SQL>select lower(‘FUN’) from dual;	fun
Upper(char)	SQL>select upper(‘sun’) from dual;	SUN
Ltrim(char, set)	SQL>select ltrim(‘xyzhello’,’xyz’) from dual;	hello
Rtrim(char, set)	SQL>select rtrim(‘xyzhello’,’llo’) from dual;	xyzhe
translate(char,from,to)	SQL>select translate(‘jack’,’j’,’b’) from dual;	back
Replace(char,from,to)	SQL>select replace(‘jack and jue’,’ j’, ’bl’) from dual;	black and blue

Example:

SQL> select initcap(dname) from department;

Output:

```
INITCAP(DNAME)
--------------
Accounts
Finance
It
Marketing
Sales
```

Lpad is a function that takes three arguments. The first argument is the character string which has to be displayed with the left padding. The second is the number which indicates the total length of the return value, the third is the string with which the left padding has to be done when required.

Example:

```
SQL> select lpad(dname,15,'*') lpd from department;
```

Output:

```
LPD
---------------
*******accounts
********finance
*************IT
******marketing
**********Sales
```

Example:

```
SQL> select rpad(dname,15,'*') rpd from department;
```

Output:

```
RPD
---------------
accounts*******
finance********
IT*************
marketing******
Sales**********
```

Length: returns the length of a string

Example:

```
 SQL> select dname, length(dname) from department;
```

Output:

```
DNAME          LENGTH(DNAME)
-------------- -------------
accounts                   8
finance                    7
IT                         2
marketing                  9
Sales                      5
```

Concatenation operator ||: is used to merge or more strings.

Example:

```
SQL> select dname || ' is located in ' || loc from department;
```

Output:

```
DNAME||'ISLOCATEDIN'||LOC
------------------------------------------
accounts is located in chennai
finance is located in Hyderabad
IT is located in Bangalore
marketing is located in chennai
Sales is located in
```

NUMERIC FUNCTIONS:

Numeric functions accept numeric input and returns numeric values as output.

Syntax	Description
abs ( )	Returns the absolute value
ceil ( )	Rounds the argument
cos ( )	Cosine value of argument
exp ( )	Exponent value
floor( )	Truncated value
power (m,n)	N raised to m
mod (m,n)	Remainder of m / n
round (m,n)	Rounds m’s decimal places to n
trunc (m,n)	Truncates m’s decimal places to n
sqrt (m)	Square root value

Examples:

Function	Input	Output
Abs(n)	SQL>select abs(-15) from dual	15

Ceil(n)	SQL>select ceil(48.778) from dual;	49
Cos(n)	SQL>select cos(180) from dual;	-0.59884601
Cosh(n):	SQL>select cosh(0) from dual;	1
Exp(n)	SQL>select exp(4) from dual;	54.59815
Floor(n)	SQL>select floor(4.678) from dual;	4
Power(m ,n)	SQL>select power(5,2) from dual;	25
Mod(m ,n)	SQL>select mod(11,2) from dual;	1
Round(m ,n)	SQL>select round(112.257,2) from dual;	112.26

Example:

SQL> select ln (2) from dual; (returns natural logarithm value of 2)

SQL>select sign (-35) from dual; (output is -1)

CONVERSION FUNCTIONS: Convert a value from one data type to another.

- **To_char ( )**

To_ char (d [,fmt]) where d is the date fmt is the format model which specifies the format of the date. This function converts date to a value of varchar2datatype in a form specified by date format fmt.if fmt is neglected then it converts date to varchar2 in the default date format.

Example:

SQL> select to_char (hire_date, 'ddth "of" fmmonth yyyy') from employee;

Output:

```
TO_CHAR(HIRE_DATE,'DDT
----------------------
21st of july 2010
05th of june 2008
12th of february 1999
```

- **To_ date ( )**

The format is to_date (char [, fmt]). This converts char or varchar data type to date data type. Format model, fmt specifies the form of character.

Example:

SQL>select to_date (‘December 18 2007’,’month-dd-yyyy’) from dual;

Output:

18-DEC-07 is the output.

Example:

SQL> select round(hire_date,'year') from employee;

Output:

```
ROUND(HIR
---------
01-JAN-11
01-JAN-08
01-JAN-99
```

- **To_ Number( )**

Allows the conversion of string containing numbers into the number data type on which arithmetic operations can be performed.

Example: SQL> select to_number (‘100’) from dual;

DATE FUNCTIONS

Function Name	**Return Value**
ADD_MONTHS (date, n)	Returns a date value after adding *’n’*months to the date *’x’*.
MONTHS_BETWEEN (x1, x2)	Returns the number of months between dates x1 and x2.
ROUND (x, date_format)	Returns the date *‘x’*rounded off to the nearest century, year, month, date, hour, minute, or second as specified by the *‘date_format’*.
TRUNC (x, date_format)	Returns the date *‘x’* lesser than or equal to the nearest century, year, month, date, hour, minute, or second as specified by the ‘date_format’.
NEXT_DAY (x, week_day)	Returns the next date of the *‘week_day’*on or after the date *‘x’* occurs.
LAST_DAY (x)	It is used to determine the number of days remaining in a month from the date *'x'*specified.
SYSDATE	Returns the systems current date and time.

Example:

SQL> select sysdate from dual;

Output:

```
SYSDATE
---------
22-JUL-10
```

Example:

```
SQL> select hire_date from employee;
```

Output:HIRE_DATE

```
---------

21-JUL-10
05-JUN-08
12-FEB-99
```

Example:

```
SQL> select add_months(hire_date,3) from employee;
```

Output:

```
ADD_MONTH
---------
21-OCT-10
05-SEP-08
12-MAY-99
```

Example:

```
SQL> select months_between(sysdate,hire_date) from employee;
```

Output:

```
MONTHS_BETWEEN(SYSDATE,HIRE_DATE)
---------------------------------
                  .047992085
                  25.5641211
                  137.338315
```

Example:

```
SQL> select next_day(hire_date,'wednesday') from employee;
```

Output:

```
NEXT_DAY(
---------
```

```
28-JUL-10
11-JUN-08
17-FEB-99
```

Example:

```
SQL> select last_day(hire_date) from employee;
```

Output:

```
LAST_DAY(
---------
31-JUL-10
30-JUN-08
28-FEB-99
```

Group Functions: - Group functions are built-in SQL functions that operate on groups of rows and return one value for the entire group. These functions are: COUNT, MAX, MIN, AVG, SUM, DISTINCT

- Group functions operate on sets of rows to give one result per group of Employees

Dept_id	Salary
90	5000
90	10000
90	10000
60	5000
60	5000

→ The maximum salary in the employees table → Max (salary) 10000

Types of Group Functions

Syntax	**Description**
count (*), count (column name), count (distinct column name)	Returns number of rows

min (column name)	Min value in the column
max (column name)	Max value in the column
avg (column name)	Avg value in the column
sum (column name)	Sum of column values

Group Functions Syntax:

Select [column,] group_function(column),..

From   table

[where   condition]

[GROUP BY column];

Example:

Q.Display the average,highest, lowest and sum of salaries for all the sales representatives.

A. Select avg(salary), max(salary), min(salary), sum(salary) From employees where   job_id like ‘%rep%’;

Groups of Data : Divide rows in a table in to smaller groups by using the group by clause

Employee Table

Dept_id	Salary
10	4000
10	5000
10	6000
50	5000

→ The average salary in employees table for each department →

D_id	Avg(Salary)
10	5000
50	4000

50	3000

SET OPERATORS: UNION,UNION ALL,DIFFERENCE,MINUS

Example:

```
sql> select first_name from employees union select name from sample ;
```

Output:

```
FIRST_NAME
----------
DHANA
GUNA
JAI
JAISANKAR
KUMAR
RAJA
VENKAT
```

Example:

```
sql> select first_name from employees union all select name from sample ;
```

Output:

```
FIRST_NAME
----------
VENKAT
JAI
DHANA
GUNA
JAISANKAR
VENKAT
RAJA
KUMAR
```

Example:

```
sql>  select first_name from employees intersect select name from sample ;
```

Output:

```
FIRST_NAME
----------
```

```
VENKAT
```

Example:

sql> select first_name from employees minus select name from sample ;

Output:

```
FIRST_NAME
----------
DHANA
GUNA
JAI
```

JOINS :

A join is the SQL way of combining the data from many tables. It is performed by WHERE Clause which combines the specified rows of the tables.

Type	Sub type	Description
Simple join	Equi join (=) Non – equi join (<, <=, >, >=, !=, < >)	Joins rows using equal value of the column Joins rows using other relational operators(except =)
Self join	-- (any relational operators)	Joins rows of same table
Outer join	Left outer join ((+) appended to left operand in join condition) Right outer join ((+) appended to right operand in join condition)	Rows common in both tables and uncommon rows have null value in left column Vice versa

Simple Join:

a. EQUI JOIN OR INNER JOIN : A column (or multiple columns) in two or more tables match.

Syntax:

SELECT <column_name(s)>

FROM <table_name1>

INNER JOIN <table_name2>

ON <table_name1.column_name>=<table_name2.column_name>;

Example 1 :

```
SELECT employee.first_name, department.dname
FROM employee INNER JOIN department
ON employee.deptno = department.deptno;
```

Output:1

```
 DEPTNO FIRST_NAME
   ---------- ----------
      10       Smith
      30        Arun
      10        Nithya
```

Oracle automatically defaults the JOIN to INNER so that the INNER keyword is not required. They are the same query, though. It is preferred not to type the INNER keyword.

Example 2 using where Condition:

```
SELECT employee.ename, department.dname
FROM employee JOIN department
ON employee.deptno = department.deptno
WHERE department.dname = 'SALES';
```

Output 2:

```
 DEPTNO FIRST_NAME
   ---------- ----------
     10          Smith
     30          Arun
     10          Nithya
```

b. SELF JOIN: Is a join where a table is joined to itself.

Syntax:

```
SELECT <column_name(s)>
FROM <table_name1>
JOIN <table_name2>
ON <table_name1.column_name>=<table_name1.column_name>;
```

Example1:

```
SELECT e1.first_name, e2.first_name
FROM employee e1 join employee e2
on e1.mgr = e2.employee_id;
```

OR

```
SELECT e1.first_name, e2.first_name
FROM employee e1 join employee e2
where e1.mgr = e2.employee_id;
```

Output:

```
FIRST_NAME FIRST_NAME
----------------   --------------------
         john         john
```

An alias is just a way to refer to a column or table with a UNIQUE name. If we try to call both of the instances of the table EMP, Oracle wouldn't know which table instance we refer to. Using an alias clears this confusion

c. OUTER JOIN

An **outer join** tells Oracle to return the rows on the **left or right** (of the JOIN clause) even if there are no rows.

The **LEFT OUTER** keyword to the JOIN clause says, return the rows to the left (in this case DEPARTMENT) even if there are no rows on the right (in this case employee).

Syntax:

```
SELECT <column_name(s)>
```

FROM <table_name1>

LEFT OUTER JOIN <table_name2>

ON <table_name1.column_name>=<table_name2.column_name>;

Example:

```
SELECT department.dname, employee.first_name
FROM department LEFT OUTER JOIN employee
ON department.deptno = employee.deptno
WHERE department.dname = 'marketing’;
```

Output:

```
DNAME          FIRST_NAME
--------------      ----------
Marketing
```

The **RIGHT OUTER** keyword to the JOIN clause says ,return the rows to the right relation (in this case DEPARTMENT) even if there are no matching rows on the left relation (in this case employee).

Syntax:

SELECT <column_name(s)>

FROM <table_name1>

RIGHT OUTER JOIN <table_name2>

ON <table_name1.column_name>=<table_name2.column_name>;

Example:

```
SELECT employee.first_name, department.dname

FROM employee RIGHT OUTER JOIN department

ON employee.deptno = department.deptno

WHERE department.dname = 'marketing';
```

Output:

```
FIRST_NAME  DNAME
----------        --------------
                    marketing
```

d. FULL OUTER JOIN

Let's insert a new record into the employee table:

INSERT INTO EMPLOYEE (employee_id, first_name, last_name, mgr, hiredate, job-id,sal, comm, deptno) VALUES (9999, 'Joe ‘,’Blow', 7698, sysdate ,0008, 10500, 0, NULL);

<u>Note:</u>

We inserted an employee record that has no department. How can we get the records for all employees AND all departments? We would use **the FULL OUTER** join syntax:

Syntax:

SELECT <column_name(s)>

FROM <table_name1>

FULL OUTER JOIN <table_name2>

ON <table_name1.column_name>=<table_name2.column_name>;

Example:

SELECT employee.first_name, department.dname

FROM employee FULL OUTER JOIN department

ON employee.deptno = department.deptno;

Output:

```
FIRST_NAME DNAME
----------          --------------
Nithya           accounts
Smith            accounts
                 finance
john             IT
Arun             IT
                 marketing
john
```

e.Cross Join

Displays all the rows and all the colums of both the tables.

Synatx:

SELECT <column_name(s)> FROM <table_name1> CROSS JOIN<table_name2>;

Example:

select employee.deptno from employee cross join department;

Or
select employee.deptno from employee,department;

Output:

```
   DEPTNO
----------
        10
        10
        10
        10
        30
        30
        30
        30
        10
        10
        10
   DEPTNO
----------
        10
        30
        30
        30
        30
```

f. Natural Join

If two tables have same column name the values of that column will be displayed only once.

Syntax:

SELECT <column_name(s)> FROM <table_name1> Natural JOIN<table_name2>;

Example:

select deptno,first_name from employee natural join department;

Output:

```
DEPTNO FIRST_NAME

------          ----------
    10          Smith
    30          Arun
    10          Nithya
```

```
   30              john
```

SUB QUERIES

- Nesting of queries
- A query containing a query in itself   A
- Inner most sub query will be executed first
- The result of the main query depends on the values return by sub query
- Sub query should be enclosed in parenthesis

1. Sub query returning only one value

a. Relational operator before sub query.

Syntax:

SELECT <column_name(s)> FROM <table_name> WHERE < column name >

< relational op.> < sub query>;

Example:

```
SELECT employee_id ,first_name FROM employee
WHERE deptno =
(SELECT deptno FROM department
WHERE dname = ‘IT’)
```

Output:

```
EMPLOYEE_ID FIRST_NAME
-----------              ----------
      112                  Arun
      114                   john
```

2. Sub query returning more than one value

a. ANY

For the clause any, the condition evaluates to true if there exists at least one row selected by the sub query for which the comparison holds. If the sub query yields an empty result set, the condition is not satisfied.

Syntax:

```
SELECT <column_name(s)>
FROM <table_name>
 WHERE < column name >
 < relational op.> ANY (<sub query>);
```

Example:

```
SELECT employee_id ,first_name FROM employee
WHERE salary>= ANY
(SELECT salary FROM employee
WHERE deptno = 30)
AND deptno = 10;
```

Output:

```
EMPLOYEE_ID FIRST_NAME
-----------           ----------
      113                Nithya
      112                Arun
      111                Smith
      114                 john
      114                 john
```

b. ALL

For the clause all, in contrast, the condition evaluates to true if for all rows selected by the sub query the comparison holds. In this case the condition evaluates to true if the Sub query does not yield any row or value.

Syntax:

```
SELECT <column_name(s)>
FROM <table_name>
WHERE < column name > < relational op.> ALL (<sub query>);
```

Example:

```
SELECT employee_id ,first_name FROM employee
WHERE salary > ALL
```

```
(SELECT salary FROM employee
WHERE deptno = 30);
```

Output:

```
EMPLOYEE_ID FIRST_NAME
-----------            ----------
      113               Nithya
```

c. IN : Main query displays the values that match with any of the values returned by sub query.

Syntax:

SELECT <column_name(s)>

FROM <table_name>

WHERE < column name > IN (<sub query>);

Example:

```
SELECT employee_id ,first_name FROM employee
WHERE deptno IN
(SELECT deptno FROM department
WHERE loc = ’Bangalore’);
```

Output:

```
EMPLOYEE_ID FIRST_NAME
-----------              ----------
      114                john
      112                Arun
```

d. NOT IN

Main query displays the values that match with any of the values returned by sub query.

Syntax:

SELECT <column_name(s)>

FROM <table_name>

WHERE < column name > NOT IN (<sub query>);

Example:

```
SELECT employee_id ,first_name FROM employee
WHERE deptno NOT IN
(SELECT deptno FROM department
WHERE loc = ’Bangalore’);
```

Output:

```
EMPLOYEE_ID FIRST_NAME
-----------            ----------
      113              Nithya
      111              Smith
```

e. EXISTS

Main query displays the values that match with any of the values returned by sub query.

Syntax:

SELECT <column_name(s)>

FROM <table_name>

WHERE EXISTS (<sub query>);

Example:

```
SELECT *  FROM department
WHERE EXISTS
(SELECT  *  FROM employee
WHERE deptno = department.deptno);
```

Output:

```
   DEPTNO DNAME         LOC
    ---------- -------------- -------------
      10       accounts      chennai
      30        IT           Bangalore
```

f. NOT EXISTS

Main query displays the values that match with any of the values returned by sub query.

Syntax:

```
SELECT <column_name(s)>
FROM <table_name>
WHERE NOT EXISTS (<sub query>);
```

Example:

```
SELECT *  FROM department
WHERE NOT EXISTS
(SELECT  *  FROM employee
WHERE deptno = department.deptno);
```

Output:

```
   DEPTNO DNAME          LOC
   ---------- -------------- -------------
      20         finance       Hyderabad
      40       marketing      chennai
```

g. GROUP BY CLAUSE

Often applications require grouping rows that have certain properties and then applying an aggregate function on one column for each group separately. For this, SQL provides the clause group by <group column(s)>. This clause appears after the where clause and must refer to columns of tables listed in the from clause.

Rule:

Select attributes and group by clause attributes should be same.

Syntax:

```
        SELECT <column_name(s)>
        FROM <table_name>
        Where <conditions>
        GROUP BY <column2>, <column1>;
```

Example:

SELECT deptno, min(salary), max(salary)

FROM employee

GROUP BY deptno;

Output:

```
 DEPTNO MIN(SALARY) MAX(SALARY)
      ---------        -----------          -----------
       30             30000                  40000
                      30000                  30000
       10             33000                  45000
```

h. HAVING CLAUSE: used to apply a condition to group by clause

Syntax:

SELECT <column(s)>

FROM <table(s)>

WHERE <condition>

[GROUP BY <group column(s)>]

[HAVING <group condition(s)>];

Example:

SELECT deptno, min(salary), max(salary)

FROM employee

WHERE job_id = ’J2’

GROUP BY deptno

HAVING count(*) > 1;

Output:

```
    DEPTNO MIN(SALARY) MAX(SALARY)
----------      -----------                 -----------
        30          13000                    40000
```

A query containing a group by clause is processed in the following way:

1. Select all rows that satisfy the condition specified in the where clause.

2. From these rows form groups according to the group by clause.

3. Discard all groups that do not satisfy the condition in the having clause.

4. Apply aggregate functions to each group.

5. Retrieve values for the columns and aggregations listed in the select clause.

i. ORDER BY

Used along with where clause to display the specified column in ascending order or descending order . Default is ascending order

Syntax:

SELECT [distinct] <column(s)>

FROM <table>

[WHERE <condition>]

[ORDER BY <column(s) [asc|desc]>]

Example:

SELECT first_name, deptno, hire_date

FROM employee

ORDER BY deptno ASC, hire_date desc;

Output:

```
FIRST_NAME     DEPTNO  HIRE_DATE
  ----------          ----------       ---------
    Smith                10              21-JUL-10
    Nithya               10              12-FEB-99
     john                30              20-JAN-10
    Arun                 30              05-JUN-08
    john                                 20-JAN-10
```

VIEWS

<u>Definition</u>: A view is a named, derived, virtual table. A view takes the output of a query and treats it as a table; therefore a view can be thought of as a ‘stored query’ or a ‘virtual table’. We can use views in most places where tables can be used. To the user, accessing a view is like accessing a table. The RDBMS creates an illusion of a table, by assigning a name to the view and storing its definition in the database.

The tables upon which the views are based are called as ‘base tables’.

<u>CREATION OF A VIEW:</u>

The syntax for creating a view is given by:

create [or replace][[no][force]]view <view_name> [column alias name…]as <query>[with[check option]read only][constraint]];

Example:
SQL>create or replace view EMP_VIEW as select * from EMP;

This statement creates a view named EMP_VIEW .The data in this view comes from the base table EMP. Any changes made to the base table are instantly visible through the view EMP_VIEW.We can use select statement just like on a table.

SQL>select * from EMP_VIEW;

When create or replace is given, view is created if it is not available otherwise it is recreated.

<u>HOW DOES RDBMS HANDLE THE VIEWS</u>: When a reference is made by a user, the RDBMS finds the definition of the view stored in the database .It then translates the user’s request that referenced the view into an equivalent request against the source tables of the view. Thus RDBMS maintains the illusion of the view.

<u>TYPES OF VIEWS:</u> The different types of views are

- Column subset view
- Row subset view
- Row-Column subset view
- Grouped view
- Joined view

<u>COLUMN SUBSET VIEW</u>:

A column subset view is one where all the rows but only some of the columns of the base table form the view. The create view

Example:

SQL>create or replace view CSV as select empno, ename, sal from EMP;

This view includes only columns empno, ename, sal of EMP table. Since there is no where clause it includes all the rows.

ROW SUBSET VIEW:

A row subset view is one where all columns but some rows of the source table form the view. All the columns of the base table participate in the view but all rows do not.

Example:

SQL> create or replace view RSV as select * from EMP where deptno=10;

The where clause restricts the no. of rows to those of employees working in Department Number 10.

ROW-COLUMN SUBSET VIEW:

A row-column subset view is a view which includes only some rows and columns of the base table.

Example:

SQL>create or replace view RCS as select EMPNO, ENAME, SAL from EMP where deptno=10;

GROUPED VIEW:

The query specified in the view definition can include the GROUP BY clause. This type of view is called as Grouped View.

Example:

SQL>create or replace view GV (dno, avgsal) as select deptno, AVG (SAL) from emp group by deptno;

JOINED VIEWS:

A joined view is formed by specifying a two or more table query in the view definition. A joined view draws its data from two or more tables and presents the result as a single virtual table.

Example:

SQL>create or replace view JV(empno,ename,sal,dname,loc) as select empno,ename,sal,dname,loc from EMP,DEPT where EMP.deptno=DEPT.deptno;

<u>CREATING A READ ONLY VIEW</u>:

Use with read only clause to prevent the users from manipulating records via the view.

Example:

SQL>create or replace view WRO as select * from EMP with read only;

Note: A view can be created without a base table using FORCE option of create view command.

Example:

SQL>create or replace force view FVIEW as select * from MYDEPT;

In this query MYDEPT table does not exist, so view is created with compilation errors. When MYDEPT table is created and this query is executed, the view is automatically recompiled and become valid.

<u>VIEW WITH CHECK OPTION</u>:

This option specifies that inserts and updates performed through the view must result in rows that the view query can select. The CHECK OPTION can be used to maintain integrity on a view.

Example:

SQL>insert into RSV (empno, ename, sal, deptno) values (1000,’dinesh’, 5500, 20);

Though the view is created for deptno 10, we are able to insert records for other department numbers .This can be restricted using WITH CHECK OPTION clause while creating a view.

Example:

SQL>create view DEPTNO10_VIEW as select * from EMP where deptno=10 WITH CHECK OPTION CONSTRAINT CHK_DNO10;

The above statement creates a view DEPTNO10 with a check constraint. This will enforce the view to be inserted or updated only for the department number 10. No other departments can be inserted or updated.

DROPPING A VIEW:

A view can be dropped by using DROP VIEW command.A view becomes invalid if its associated base table is dropped.

Example:

SQL>drop view DEPTNO10;

This will not affect the base table EMP.

ADVANTAGES OF VIEWS:

- Valid Information: Views let different users see a table from different perspectives. Only the part that is relevant to the users is visible to them.
- Restricted Access: Views restrict access to the table. Different users are allowed to see only certain rows or certain columns of a table.
- Simplified Access: Views simplify database access. For example a view that is a join of three tables where a user does not require all the data in all three tables.
- Data Integrity: Data Integrity can be maintained by having WITH CHECK OPTION while creating a view.

RESTRICTIONS ON VIEWS:

- A view’s query cannot select the CURRVAL or NEXTVAL pseudo columns.
- If a view’s query selects the ROWID, ROWNUM or LEVEL pseudo columns, they must have aliases in the view’s query.
- A view can’t be created with an ORDER BY clause.
- A view can’t be updated, deleted and inserted if it is a grouped view.
- A view created from multiple tables can’t be updatable.
- If a view is based on a single underlying table then you can insert, update or delete rows in this view. This will actually insert, update or delete rows in the underlying table. There are restrictions again on doing this:
- You cannot insert if the underlying table has a NOT NULL column that does not appear in the view.
- You cannot insert or update if any of the view’s columns referenced in insert or update consist of functions or calculations.

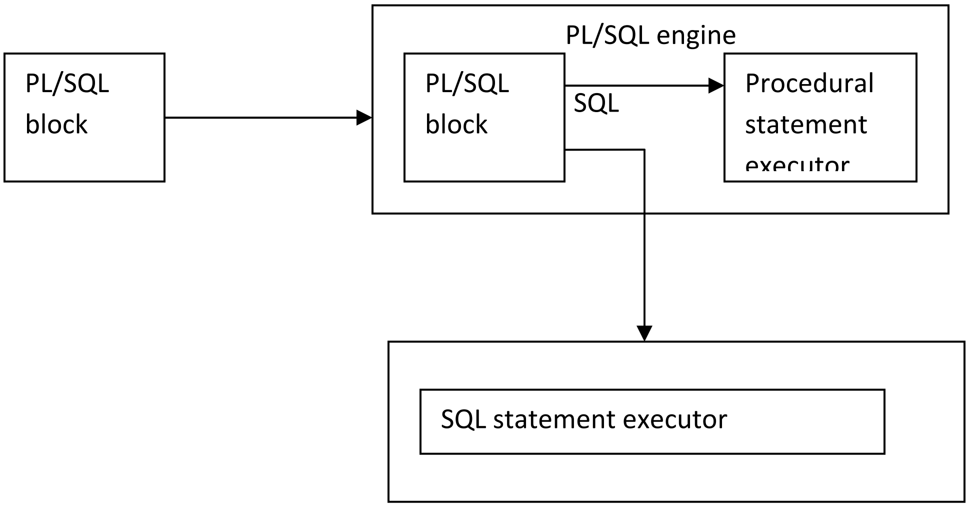
- You cannot insert, update or delete if the view contains GROUP BY, DISTINCT or a reference to a pseudo column ROWNUM.

PL/SQL

Overview of PL/SQL

PL/SQL is the procedural extension to SQL with design features of programming languages. Data manipulation and query statements of SQL are included within procedural units of code.

Pl/SQL Environment



The PL/SQL engine in the oracle server process the pl/sql block and it separates SQL staments and sends them individually to the SQL statements executor

Benefits of PL/SQL

- Integration
- Improved performance
- Modularized program development
- Portability
- Identifiers

PL/SQL Block structure

DECLARE (optional)

Variables, cursors, user-defined exceptions

BEGIN (Mandatory)

-SQL statements

-PL/SQL statements

EXCEPTION (optional)

Action to perform when error occur

END;

The PL/SQL Block consists of three sections:

DECLARATIVE

It contains all variables, constants, cursors and user defined exceptions that are referenced in the executable and declarative sections.

EXECUTABLE

It contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block.

EXCEPTION HANDLING

It specifies the actions to perform when errors and abnormal conditions arise in the executable section.

PL/SQL Block Types

A PL/SQL program comprises one or more blocks.

It is classified into two blocks

- **Anonymous Blocks**

It is unnamed blocks. It is declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at run time.

- **Subprograms**
  Subprograms are named PL/SQL blocks that can accept parameters and can be invoked. It can be declared either as procedures or as functions.

Sample PL/SQL programs

To write PL/SQL programs, create a script file and run the script file or use editor.

Steps to create script file

Step1:

```
SQL> edit z:\oracle\sql\var1.sql
```

Step2:

Type the program in notepad

Step 3:

Save the program

Step4:

Run the program

```
SQL> @z:\oracle\sql\var1.sql
```

```
SQL> set serveroutput on;
```

This command is used to display the statement executed by dbms_output.put_line package.

Program 1: Write a program to print a variable value.

```
SQL> declare

  2  a number:=3;

  3  begin
```

```
dbms_output.put_line(a);
end;
/
   3
```

Program 2: Write a program to print your name and regno.

```
declare
v_name varchar2(10);
v_regno number;
begin
v_name:='venkat';
v_regno:=39;
dbms_output.put_line( 'the name is' || v_name);
dbms_output.put_line('the no is' || v_regno);
end;
```

```
SQL> /
the name is venkat
          the no is 39

PL/SQL procedure successfully completed.
```

Program 3: Write a program to retrieve ssn number of employee whose name is x.

Assume the following table:

```
      SSN        NAME          ESSN    DEPTNO    SALARY
---------- ---------- ---------- ---------- ----------
      101        x              102       1
      102        y              103       2
      103        z              102       3
      104        p              102       4
      105        q
```

```
declare v_no number;
begin
select ssn into v_no from emp where name='x';
dbms_output.put_line(v_no);
end;
SQL>/
101
PL/SQL procedure successfully completed.
```

SCALAR VARIABLE

It holds a single value and has no internal components.

Examples : number, character, date, boolean

Example: using scalar variable

```
  1  declare
  2  v_name varchar2(10);
  3  V_count binary_integer:=10;
  4  V_totalsal number(9,2);
  5  v_orderdate date:=sysdate;
  6  c_tax constant number(3,2):=6.23;
  7  v_valid boolean not null:=true;
  8  v_regno number default 23;
  9  begin
 10  v_name:='venkat';
 11  v_totalsal:=10000.23;
 12  dbms_output.put_line(v_name);
 13  dbms_output.put_line(v_count);
 14  dbms_output.put_line(v_orderdate);
 15  dbms_output.put_line(c_tax);
 16  dbms_output.put_line(v_regno);
 17 end;
 18  /
venkat
10
19-AUG-05
6.23
23
```

DECLARING VARIABLE WITH THE %TYPE ATTRIBUTE

The % type attribute is used to declare a variable according to:

1. A database column definition
2. Another previously declared variable

Example: using % type attribute

```
  1  declare
  2  v_no emp.ssn%type;
  3  V_name varchar2(10):='venkat';
  4  name  v_name%type;
  5  begin
  6  v_no:=10;
  7  name:='ven';
  8  dbms_output.put_line(v_no);
  9  dbms_output.put_line(name);
 10*end;
 11  /
10
ven

PL/SQL procedure successfully completed.
```

BIND VARIABLES

A bind variable is a variable that is declared in a host environment. Bind variables can be used to pass run-time values, which can be either number or character, into or out of one or more PL/SQL programs.

Example:

```
SQL> variable a number;

SQL> ed

File:

  1  begin

  2  select ssn into:a from emp where name='x';

  3  dbms_output.put_line(:a);

  4 end;

SQL> /

101

PL/SQL procedure successfully completed.

SQL> print a;

         A

----------

       101


PL/SQL procedure successfully completed.
```

REFERENCING NON PL/SQL VARIABLES

To reference host variables, prefix the references with a colon (:) to distinguish them from declared PL/SQL variable.

```
SQL> variable gg number;
SQL> define aa=1000;

SQL> set verify off

SQL> declare

  2  v_sal number(9,2):=&aa;

  3  begin

  4  :gg:=v_sal/12;

  5  end;

  6  /

PL/SQL procedure successfully completed.

SQL> print gg;

        GG

----------

83.3333333
```

PL/SQL BLOCK SYNTAX AND GUIDELINES

A line of pl/sql text contains groups of characters known as lexical units.

Lexicals are classified as follows:

- Delimiters
- Identifiers, which include reserved words
- Literals
- Character literals
- Numeric literals

COMMENTS:

-- single line commenting

/* beginning */ ending

PL/SQL HAS ITS OWN ERROR HANDLING:

- SQLCODE
- SQL ERRM

DATA TYPE CONVERSION

PL/SQL performs implicit conversions. For E.g. numeric to char.

The following program highlights conversion involving DATE.

```
SQL> ed

  1  declare

  2  vdate date;

  3  begin

  4  vdate:=to_date('aug 19 ,2005','mon dd,yyyy');

  5  dbms_output.put_line(vdate);

  6 end;

SQL> /

19-AUG-05

PL/SQL procedure successfully completed.
```

SQL FUNCTIONS IN PL/SQL:

All SQL functions are allowed except decode function and group functions.

NESTED BLOCKS AND VARIABLE SCOPE

```
SQL> declare
  2  v_a number:=3;
  3  begin
  4  declare
  5  v_b number:=4;
  6  begin
  7  dbms_output.put_line(v_b);
     dbms_output.put_line(v_a);
  8  end;
  9  dbms_output.put_line(v_a);
 10  end;
 11  /
4
3

PL/SQL procedure successfully completed.
```

QUALIFYING AN IDENTIFIER An identifier is qualified by using the block label prefix. In the example the outer block is labeled as outer. In the inner block the variable is reference by label, when variable names are same.

```
SQL> ed
```

```
 1  <<outer>>
 2  declare
 3  v_a number:=3;
 4  begin
 5  declare
 6  v_a number:=4;
 7  begin
 8  dbms_output.put_line(v_a);
 9  dbms_output.put_line(outer.v_a);
 10  end;
 11 end;
 12  /
4
3
PL/SQL procedure successfully completed.
```

PROGRAMMING GUIDELINES:

Category	Case Conversion	Examples
SQL statements	uppercase	SELECT,INSERT
PL/SQL statements	uppercase	DECLARE,BEGIN,IF

Data types	uppercase	VARCHAR2,BOOLEAN
Identifiers	lowercase	v_sal
Database tables and column	lowercase	emp, dept

INTERACTING WITH ORACLE SERVER:

- Extracts row of data from the database using **select**
- Effects changes in the database by using DML commands
- Controls a transaction with commit, rollback and save point

NOTES:

- An end in pl/sql block is not the end of transaction.
- A block can span multiple transactions, a transaction can span multiple blocks.
- DDL commands (create,alter,drop) and DCL commands(grant,revoke) are not directly supported.

```
SQL> select * from emp;

      SSN          NAME      ESSN     DEPTNO     SALARY

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

      101          x              102         1

      102          y              103         2

      103           z              102         3

      104          p              102         4

      105          q

SQL> ed

  1  declare

  2  v_ssn number;
  3  v_name varchar2(10);
  4  begin
  5  select ssn,name into v_ssn,v_name from emp where name='x';
  6  dbms_output.put_line(v_ssn);
  7  dbms_output.put_line(v_name);
  8 end;
SQL> /
101
x
PL/SQL procedure successfully completed.
```

RETRIEVING DATA IN PL/SQL:

```
SQL> select * from job_grade;
GRA    LOWEST_SAL         HIGHEST_SAL
---- ---------- ----------------------------------------
a          3000                    4000
b          5000                    6000
c          3000                    6000
d          4000                   10000
e          2000                   6000

SQL> ed

  1  declare
  2  v_lsal job_grade.lowest_sal%type;
  3  v_hsal job_grade.highest_sal%type;
  4  begin
  5  select sum(lowest_sal),sum(highest_sal) into v_lsal,v_hsal from job_grade;
  6  dbms_output.put_line(v_lsal);
  7  dbms_output.put_line(v_hsal);
  8 end;
  9  /
17000
32000
PL/SQL procedure successfully completed.
```

NAMING CONVENTIONS

A local variable in pl/sql name must not be equal to column names present in database .

```
declare
lastname varchar2(10);
begin
delete from emp where lastname-lastname;
```

The above code will delete all employees because of the naming convention problem.

MANIPULATING DATA USING PL/SQL

SUBSTIUTION VARIABLE:

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

```
  1  declare
  2  v_sal number;
  3  begin
  4  v_sal:=&v_sal;
  5  dbms_output.put_line(v_sal);
  6 end;
  7  /
Enter value for v_sal: 2000
2000
PL/SQL procedure successfully completed.
```

INSERTION

```
SQL> ed
Wrote file afiedt.buf

  1   begin
  2   insert into emp(ssn,name) values(123,'venkat');
  3   dbms_output.put_line('record inserted');
  4 end;
  5  /
record inserted
PL/SQL procedure successfully completed.
```

USAGE OF SUBSTITUTION VARIABLE:

```
SQL> ed
Wrote file afiedt.buf
  1   begin
  2   insert into emp(ssn,name) values(&ssn,'&name');
  3   dbms_output.put_line('record inserted');
  4  end;
SQL> /

Enter value for ssn: 124
Enter value for name: sampath
record inserted
PL/SQL procedure successfully completed.
```

UPDATE:

```
SQL> ed
Wrote file afiedt.buf
  1  declare
  2  v_sal number;
  3   begin
  4  v_sal:=&v_sal;
  5  update job_grade set lowest_sal=v_sal where gra='a';
  6   dbms_output.put_line('record updated');
  7  end;
SQL> /
Enter value for v_sal: 12000
record updated
PL/SQL procedure successfully completed.
SQL> select *from job_grade;

GRA  LOWEST_SAL          HIGHEST_SAL
---- ---------- ------------------------------------------
a         12000                    4000
b          5000                    6000
c          3000                    6000
d          4000                    10000
e          2000                    6000
```

DELETE:

```
SQL> ed
Wrote file afiedt.buf
  1    declare
  2    v_sal number;
  3     begin
  4    v_sal:=&v_sal;
  5    delete from job_grade where lowest_sal=v_sal;
  6   dbms_output.put_line('record deleted');
  7  end;
  8  /
Enter value for v_sal: 5000
record deleted


PL/SQL procedure successfully completed.
SQL> select * from job_grade;
GRA  LOWEST_SAL HIGHEST_SAL
---- ---------- -----------
a         12000        4000
c          3000        6000
d          4000       10000
e          2000        6000
```

CONTROL STRUCTURES

- **IF statements**
  - **If –then-end if**
  - **If-then-else-end if**
  - **If-then-elseif-end if**
- **Case expressions**
- **Loop statements**
  - **Basic loops**
  - **While loops**
  - **For loops**

Syntax of IF:

```
If condition then
   Statements;
Else if condition then
   Statements;
Else
   Statements;
End if;
```

Examples:

1) Find the greatest among two numbers

```
declare
a number;
b number;
begin
a:=&a;
b:=&b;
if a>b then
dbms_output.put_line('gratest number is'||a);
else
dbms_output.put_line('gratest number is'||b);
end if;
end;
```

```
SQL> /

Enter value for a: 12 old 5: a:=&a;
new 5: a:=12;
Enter value for b: 4 old 6: b:=&b;

new 6: b:=4;
greatest number is12

PL/SQL procedure successfully completed
```

2) **if else with database**

```
DEPT_ID    DEPT_NAME        MANAGER_ID       LOCATION_ID
---------  ----------       ----------       -----------
10         cse              200              1700
20         it               300              1800
30         mech             400              1500
40         ece              500              1600
```

```
1 declare
2 v_id departments.dept_id%type;
3 v_dname departments.dept_name%type;
4 begin
```

```
5 select dept_id,dept_name into v_id,V_dname from departments where manager_id=200;

6 if v_id=11 then

7 dbms_output.put_line(v_dname);

8 elsif v_dname='cse' then

9 dbms_output.put_line(v_id);

10 else

11dbms_output.put_line('recorde not match');

12 end if;

13 end;

SQL> / 10

PL/SQL procedure successfully completed.
```

3) **if/else if/ else**

```
1 declare

2 a number;

3 b number;

4 c number;

5 begin

6 a:=&a;

7 b:=&b;

8 c:=&c;

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

10 dbms_output.put_line('gratest number is'||a);
```

```
11 elsif(b>a) and (b>c) then
12 dbms_output.put_line('greatest number is'||b);
13 else
14 dbms_output.put_line('greatest number is'||c);
15 end if;
16 end;
SQL>/
Enter values for a,b and c: 6 4 12
C is greater :12.


PL/SQL procedure successfully completed.
```

4) Case Expressions

A case expression selects a result and returns it. To select the result, the case expression uses an expression whose value is used to select one of several alternatives.

Syntax :

```
CASE selector

    WHEN
        expression1
    THEN result1 WHEN
        expression2
     THEN result2

    ---------

    WHEN expression N THEN result
    N [ELSE resultN+1
    END;
```

Example:

```
1 declare
2 va varchar2(10);
3 v_result varchar2(10);
4 begin
5 va:=&va;
6 v_result:=
7 CASE va
8 WHEN 'a' THEN 'excellent'
9 WHEN 'b' THEN 'very good'
10 WHEN 'c' THEN 'good'
11 ELSE 'poor'
12 end;
13 dbms_output.put_line('grade is'||v_result);
14 end;
15 /
```

```
SQL> Enter value for va: 'a' old 5: va:=&va;

new 5: va:='a';

grade is excellent

PL/SQL procedure successfully completed.


SQL> /


Enter value for va: 'b' old 5: va:=&va;

new 5: va:='b';

grade is very good

PL/SQL procedure successfully completed.

SQL> /


Enter value for va: 'c' old 5: va:=&va;
```

```
new 5: va:='c';
grade is good

PL/SQL procedure successfully completed.
```

5) General Loop

Syntax:

```
LOOP
...
IF ... THEN
...
EXIT; -- exit loop immediately
END IF;
END LOOP;
```

Example for General Loop:

```
Declare
Var number:=1
Begin
dbms_out.put_line(var);
Loop
Var:=var+1;
dbms_out.put_line(var);
If var=10 Then
exit;
End if;
End loop;
End;
```

Example for General Loop:

```
Declare
Var number:=1
Begin
dbms_out.put_line(var);
Loop
Var:=var+1;
dbms_out.put_line(var);
Exit when var=10;
End loop;
End;
```

6) While Loop

Syntax:

```
WHILE condition LOOP
sequence_of_statements;
...
END LOOP;
```

Example:

```
declare
var number:=1;
Begin
While var<10 Loop
dbms_output.put_line(‘var= ‘ ||var);
var:=var+1;
End Loop;
End;
```

7) For Loop

Syntax:

```
FOR counter IN [REVERSE] lower_bound..upper_bound LOOP
sequence_of_statements;
...
END LOOP;
```

Example:

```
Declare
Var number;
Begin
For var in 1..10
loop
Dbms_output.put_line(var);
End loop;
End;
```

8) Goto Syntax:

Syntax:

```
If <cond> Then
GOTO <lbl1>
End if;
```

CURSORS

The oracle server uses work areas, called private SQL areas, to execute SQL statement and to store processing information. This area is called cursor.

Cursor types:

- Implicit: queries returns only one row
- Explicit : queries returns more than one row

Explicit cursor

Active set: set of rows returned by multiple rows

Controlling explicit cursor

Open the cursor and execute the query associated with the cursor which identifies the result set.

Fetch

Retrieves the current row an advance the current row

Close the cursor

Syntax:

Cursor declaration

cursor cuname is select;

Open the cursor

open cursor name;

Close the cursor

close cursor name;

Fetch

fetch cname into variable or record

<u>Explicit Cursor Attributes</u>: To determine the status of the cursor, the cursor’s attributes are checked.Cursors have the following four attributes that can be used in a PL/SQL program.

%isopen -To check if the cursor is opened or not

%found-To check if a record is found and can be fetched from the cursor

%rowcount-To check for the number of rows fetched from the cursor

%notfound-To check if no more records can be fetched from the cursor

%isopen, %found,%notfound are boolean attributes which are set to either TRUE or FALSE.

A Simple Example:

```
declare
v_name wer1.name%type;
v_ssn wer1.ssn%type;
cursor emp_c is select * from wer1;
begin
open emp_c;
for i in 1..5 loop
fetch emp_c into v_name,v_ssn;
dbms_output.put_line(v_name);
end loop;
close emp_c;
end;
```

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

```
SQL> set serveroutput on;

SQL> / x

x
x
y
y
```

2) %row count

```
declare
v_name wer1.name%type;
v_ssn wer1.ssn%type;
cursor emp_c is select * from wer1;
begin
open emp_c;
for i in 1..5 loop
fetch emp_c into v_name,v_ssn;
exit when emp_c%rowcount>4;
dbms_output.put_line(v_name);
end loop;
close emp_c;
end;
SQL> / x
x
x
```

```
y

PL/SQL procedure successfully completed.
```

3) Cursor with record

It processes the rows of the active set by fetching values into a PL/SQL record.

```
1 declare
2 cursor emp_c is select * from wer1;
3 emp_record emp_c%rowtype;
4 begin
5 open emp_c;
6 for i in 1..5 loop
7 fetch emp_c into emp_record;
8 exit when emp_c%notfound;
9 insert into wer(name,ssn) values(emp_record.name,emp_re
10 end loop;
11 commit;
12 close emp_c;
13 end;
14 /
PL/SQL procedure successfully completed.
SQL> select * from wer;


 NAME        SSN
 ----------  ----------
 x           101
```

```
x           101
x           101
x           101
x           101
x           101
x           101
x           101
x           101
x           101
y           102
y           102
12 rows selected.
```

4) Cursor with parameters

It passes the parameter values to the cursor in a cursor FOR loop. This means that you can open and close an explicit cursor several times in a block, returning a different active set on each occasion.

Example:

```
declare
v_number number;
v_name varchar2(10);
cursor c1(eno number,ename varchar2) is
select ssn,name from emp where ssn=eno and name=ename;
begin
open c1(101,'x');
```

```
8 fetch c1 into v_number,v_name;
9 dbms_output.put_line(v_number);
10 close c1;
11 open c1(102,'y');
12 fetch c1 into v_number,v_name;
13 dbms_output.put_line(v_number);
14 close c1;
15 end;
16 /
101
102
PL/SQL procedure successfully completed.
```

5) Update

The update clause in the cursor query locks the affected rows when the cursor is opened.

Example:

```
declare

v_number number;
v_name varchar2(10);

cursor c1(eno number,ename varchar2) is select ssn,name from emp where ssn=eno and
name=ename for update of name nowait;
begin
```

```
open c1(101,'x');
fetch c1 into v_number,v_name;
dbms_output.put_line(v_number);
close c1;
open c1(102,'y');
fetch c1 into v_number,v_name;
dbms_output.put_line(v_number);
close c1;
end;
```

EXCEPTIONS

Syntax

```
When exception1
then Statement1
Statement2

……..

When exception2
 then Statement1
 Statement2

……..

When others
then Statement1
```

Statement2 ……..

Sample predefined exceptions:

NO_DATA_FOUND

TOO_MANY_ROWS

INVALID_CURSOR

ZERO_DIVIDE

DUP_VAL_ON_INDEX

Example:

1)

```
declare
a number;
b number;
c number;
begin
a:=5;
b:=0;
c:= a/b;
exception
when zero_divide then
dbms_output.put_line('zero divide error');
end;
```

```
SQL> /
zero divide error
PL/SQL procedure successfully completed.
```

2) Non-predefined error

Trapping a non-predefined exception

1. Declare the name for the exception within the declarative section
2. Associate the declared exception with the standard oracle server error number using the PRAGMA EXCEPTION_INIT statement

   Syntax : PRAGMA EXCEPTION_INIT(exception, error_number);

3. Reference the declared exception within the corresponding exception –handling routine.

Example:

```
declare
emp_remain exception;
pragma exception_init
(emp_remain,-2292);
begin
delete from emp where deptno=&deptno;
commit;
exception
when emp_remain then
dbms_output.put_line('cannot remove dept'|| 'employee exist');
end;
```

```
SQL> /

Enter value for deptno: 2

old 6: delete from emp where deptno=&deptno;

new 6: delete from emp where deptno=2;

cannot remove deptemployee exist

PL/SQL procedure successfully completed


SQL> /

Enter value for deptno: 8

old 6: delete from emp where deptno=&deptno;

new 6: delete from emp where deptno=8;

PL/SQL procedure successfully completed
```

3) Functions for trapping exceptions

When an exception occurs, you can identify the associated error code or error message by using two functions.

SQLCODE: It returns the numeric value for the error code

SQLERRM: It returns character data containing the message associated with the error number.

Syntax:

```
declare;

v_error_code number;

v_error_messgage varchar2(255);
```

```
begin
when others then rollback;
v_error_code:=sqlcode;
v_error_message:=sqlerrm;
dbms_output.put_line(v_error_code||v_error_message);
end;
```

User defined exception:

User defined PL/SQL exception must be

- Declared in the declare section of a PL/SQL block
- Raised explicitly with RAISE statements

Example:

```
declare
invalid_dept exception;
begin
delete from emp where deptno=&deptno;
if sql%notfound then
raise invalid_dept;
end if;
exception
when invalid_dept then
dbms_output.put_line('the deptnumber is not valid');
end;
```

```
Enter value for deptno: 10
old 4: delete from emp where deptno=&deptno;
 new 4: delete from emp where deptno=10;
the deptnumber is not valid
PL/SQL procedure successfully completed.
```

PL/SQL Block Types

A PL/SQL program comprises one or more blocks.

It is classified into two blocks

- **Anonymous Blocks**
  It is unnamed blocks. It is declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at run time.

- **Subprograms**
  Subprograms are named PL/SQL blocks that can accept parameters and can be invoked. It can be declared either as procedures or as functions.

Overview of subprograms

A subprogram is named PL/SQL block that ca accept parameters and be invoked from a calling environment.

Two types of subprograms

- A procedure that performs an action
- A function that computes a value

Benefits of subprograms

- Easy maintenance
- Improved data security and integrity

- Improved performance
- Improved code clarity

Procedure

A procedure is a type of subprogram that performs an action. A procedure can be stored in the database, as a schema object, for repeated execution.

Syntax for creating procedure:

Create [or replace] procedure <procedure_name>

[(parameter1 [mode1] datatype1,parameter2 [mode2] datatype2,..)]

Is| As

PL/SQL Block;

The replace option indicates that if the procedure exists, It will be dropped and replaced with the new version created by the statement. Parameter name of a PL/SQL variable whose value is passed to or populated by the calling environment.

Mode: type of argument

IN, OUT, IN OUT

IN : It is the default mode and value is passed into subprogram.

OUT : It must be specified and is returned to calling environment.

IN OUT: It is passed into subprogram and returned to calling environment.

IN parameter

IN parameters are passed as constants from the calling environment into the procedure.

Example:1

```
create or replace procedure raise_salary
(grade in job_grade.gra%type)
is
begin
update job_grade set lowest_sal=lowest_sal*1.10 where gra= grade;
 end raise_salary;
/
```

```
Procedure created.

SQL> execute raise_salary('a');  // executing procedure

PL/SQL procedure successfully completed.

SQL> select * from job_grade;

GRA    LOWEST_SAL        HIGHEST_SAL
---- ---------- ----------------------------------------------
a          13200                    4000

c          3000                     6000

d          4000                     10000

e          2000                     6000
```

IN, OUT parameter

Example:1

```
  1  create or replace procedure info
  2  (g in job_grade.gra%type,
  3   l_sal out job_grade.lowest_sal%type,
  4   h_sal out job_grade.highest_sal%type)
  5  is
  6  begin
  7  select lowest_sal,highest_sal into l_sal,h_sal from job_grade where gra=g;
  8* end info;
  9  /
SQL> edit g:\oracle\sql\info1.sql
SQL> @g:\oracle\sql\info1
Procedure created.
```

How to view the value of OUT parameters with sql *plus

1.Run the sql script file to generate and compile the source code.

2.Create host variables in sql*plus, using the **variable** command

3.Invoke the procedure, supplying these host variables as the OUT parameters.: reference the host variables in the execute command.

4.To view the values passed from the procedure to the calling environment ,use the print command.

```
SQL> variable g_sal number;

SQL> variable g1_sal number;

SQL> execute info('a',:g_sal,:g1_sal)

PL/SQL procedure successfully completed.

SQL> print g_sal;

     G_SAL

    ----------

     13200

SQL> print g1_sal;

    G1_SAL

    ----------

      4000
```

IN OUT parameter

Example:1

```
  1   create or replace procedure info

  2   (g in  out number)

  3   is

  4   begin

  5   select lowest_sal into g from job_grade where highest_sal=g;

  6*  end info;

  7  /
```

```
Procedure created.

SQL> variable g_sal number;

  1   begin

  2   :g_sal:=4000;

  3*  end;

PL/SQL procedure successfully completed.


SQL> print g_sal;

     G_SAL

    ----------

      4000

SQL> execute info (:g_sal)

PL/SQL procedure successfully completed.


SQL> print g_sal;

     G_SAL

   ----------

     13200
```

Methods for passing parameters

Positional : List actual parameters in the same order as formal parameters

Named : List actual parameters in library order by associating each with its corresponding formal parameter

Combination : List some of the actual parameters as positional and some as named.

Removing procedures

Drop a procedure stored in the database

Syntax:

Drop procedure procedure_name

Example:

Drop procedure raise_salary;

Functions

A function is a named PL/SQL block that returns a value. A function can be stored in the database as a schema object for repeated execution. A function is called as part of an expression.

Syntax:

Create [or replace] function function_name

[(parameter1 [mode1] datatype1,

Parameter2 [mode2] datatype2,

….)]

Return datatype

Is/as

PL/SQL block ;

Example:

```
declare
summation number;
average number;
function summa(m4 number,m5 number) return number is
begin
return(m4+m5);
end;
function aver( summ1 number) return number is
begin
return(summ1/2);
end;
begin
 summation:=summa(&m1,&m2);
 average:=aver(summation);
 dbms_output.put_line('summation is:'||summation);
 dbms_output.put_line('average is:'||average);
end;
```

Removing functions

```
Drop function function_name
```

Example:

Drop function summa;

Packages

Packages bundle are related PL/SQL types, items, and subprograms into one container.

A package usually has a specification and a body, stored separately in the database.

Package specification

It is the interface to the application. It declares the types, variables, constants, exceptions, cursors and subprograms.

A package specification can exist without a package body, but a package body cannot exist without a package specification.

Syntax

```
Create [or replace] package package_name

is| as

Public type and item declarations

Subprograms specifications

End package_name;
```

Example:

```
create or replace package commp is

g_comm number:=0.10;

procedure reset_comm

(p_comm in number);
```

```
end commp;

Package created.
```

Package body

Syntax

Create [or replace] package body package_name

Is| as

Private type and item declarations

Subprogram bodies

End package_name;

Example

```
create or replace package body commp
is
function validate_comm(p_comm in number)
return boolean
is
v_max_comm number;
begin
select max(lowest_sal) into v_max_comm from job_grade;
if p_comm>v_max_comm then return(false);
else return(true);
```

```
 11  end if;
 12  end validate_comm;
 13  procedure reset_comm(p_comm in number)
 14  is
 15  begin
 16  if validate_comm(p_comm)
 17  then g_comm:=p_comm;
 18  else
 19  raise_application_error(-20210,'invalid commision');
 20  end if;
 21  end reset_comm;
 22 end commp;
 23  /
```

Package body created.

Invoking package constructs:

```
SQL> execute commp.reset_comm(0.15);

PL/SQL procedure successfully completed.

SQL> create or replace package global_con is
  2  a constant number:=2;
  3  b constant number :=3;
  4  end global_con;
  5  /
```

Package created.

```
SQL> execute dbms_output.put_line('20 miles='||20*global_con.a||'km');

20 miles=40km

PL/SQL procedure successfully completed.
```

Referencing a public variable from a stand alone procedure:

```
SQL> ed

Wrote file afiedt.buf

  1  create or replace procedure me( x in number, y out number)

  2  is

  3  begin

  4  y :=x *global_con.a;

  5 end me;

SQL> /

Procedure created.


SQL> variable ya number;

SQL> execute me(3,:ya);

PL/SQL procedure successfully completed.

SQL> print ya;

        YA

----------

         6
```

Removing packages:

drop package package name;

drop package body package_name;

Overloading

It is the use of same name for different subprograms inside a PL/SQL block, a subprogram, or a package.

Example:

```
create or replace package   over
is
procedure add_dept(p_n in emp.ssn%type,p_na in emp.name%type);
procedure add_dept(p_n in emp.ssn%type,p_na in emp.name%type,p_dept in emp.deptno%type);
end over;
/
```

Package created

```
create  package body  overp is
procedure add_dept(p_n emp.ssn%type,p_na emp.name%type)
is
begin
insert into emp (ssn,name) values(p_n,p_na);
end add_dept;
procedure add_dept(p_n emp.ssn%type,p_na emp.name%type,p_dn emp.deptno%type)
is
begin
insert into emp (ssn,name,deptno) values(p_n,p_na,p_dn);
end add_dept;
```

12 end overp;

Trigger

A trigger is aPL/SQL block or a PL/SQL procedure associated with a table ,view, schema, or the database. It executes implicitly whenever a particular event takes place.

It can be:

Application trigger: fires whenever an event occurs with a particular application

Database trigger: fires whenever a data event or system event occurs on a schema or database.

A triggering statement contains:

- Triggering timing
  - For table: BEFORE, AFTER
  - For view: INSTEAD OF
- Triggering event: INSERT, UPDATE, or DELETE
- Table name: on table, view
- Trigger type: row or statement
- When clause: restricting condition
- Trigger body: PL/SQL block

Trigger type

Statement trigger: The trigger body executes once for the triggering event. this is default. A statement trigger fires once, even if no rows are affected at all.

Row trigger: The trigger body executes once for each row affected by the triggering event. A row trigger is not executed if the triggering event affects no rows.

Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
Timing
        Event1 [OR event2 OR event3]
        ON table_name
Trigger _body
```

Example:

```
create trigger ab
 before insert or delete or update on a
 for each row
 begin
 raise_application_error(-20000,'not accessible')
end
```

This program raises an error during insertion and deletion and update operation in a row.