UNIT - I
INTRODUCTION

DEFINITIONS

ATTRIBUTE (COLUMN):

It Represents a Particular Characteristic of aparticular entity. E.x. Rollno, Name, Age of a Student.

ENTITY (ROW):

It represents a record/an instance of aparticular entity.

CELL:

It represents value/data item of an attribute of arecord.

ENTITY SET (TABLE):

It is a collection of multiple entities/records/instance/rows. It is a 2-dstructure which is composed of rows and columns where each column represents a particular characteristic and each row represents a record of a particular entity.

DATABASE:

It is a collection of tables related toparticular subject.

DATABASE MANAGEMENT SYSTEM(DBMS):

It is software (system) which helps the user to manage thedatabase in an efficient manner. It allows the user to select, insert, update and delete records in various tables. It provides various types of commands to perform various types of operations. it also allows user to create views , procedures and so many other things so that it becomes quite easy for the user tohandle the database. It is basically of three types:

- Relational (RDMBS),
- Hierarchical,
- Network.

RELATIONAL DATABASE MANAGEMENT SYSTEM(RDBMS):

It is a type of DBMS in the concept of relations is used .A relation can be in between attributes of a single entity or a relation can be in between two or more than entities bound by a common column. Oracle is also a RDBMS

SQL(STRUCTURED QUERY LANGUAGE):

It is a query language which provides various types of queries to perform various tasks. Three types of SQL queries are as follows:

DDL (DATA DEFINITION LANGUAGE):

This type of query allows us to create and alter various objects like tables and views. e.x, create table, create view, alter table,alter view etc.

DML (data manipulation language)

This type of query allows us to select , insert , update and delete data from various tables.

DCL (DATA CONTROL LANGUAGE):

This type of queryallows to assign and revoke permissions from user for various operations on various objects like table and views. e.x, grant, revoke

DATA TYPES IN ORACLE:

VARCHAR2 (SIZE)/VARCHAR (SIZE):

This data type is used to store variable length alphanumeric data. no of character cannot exceed the specified size. if we enter less no of characters than specified size, spaces are nothappened to the right of string. Its value is feeded inside single quotes.

CHAR(SIZE):

This data type is used to store fixed length alphanumeric data. No of character cannot exceed the specified size. If we enter less no of characters thanspecified size, spaces are appended to the right of string. Its value is feeded inside single quotes

NUMBER(P,S):

This data type allows us to store numeric values. Precision(p) allows to specify maximum digits within a number. scale(s) allows to specify digits afterdecimal point. if value of scale is not mentioned , then its default value is 0. if value of precision is not mentioned ,then its default value is 38. maximum number of digits allowed in a number is 38.4.

LONG:

This data type allows us to store variable length string up to 2GB.

DATE:

This data type allows us to store date in formatdd-mm-yy. It's value is feeded inside Single quotes.

RAW:

This data type is used to store binary data such as imagesupto 255 bytes.

LONG RAW:

This data type is used to store binary data such as images up to 2GB.

VARIOUS COMMONLY USED DDL COMMANDS USED IN ORACLE:

CREATE TABLE:

This command is used to create tableby using this command, we specify name of thetable, name of the columns and data type of the columns.

SYNTAX:

CREATE TABLE tablename (Columnname1 datatype,columnname2 datatype , . , columnnamedatatype)

EXAMPLE:

CREATE TABLE student (rollno number(3) , name varchar2(20) ,age n umber(3) , marks number(4,2));

ALTER TABLE:

This command is used to alter the structure of a table. We can add new columns by using alter table command.

SYNTAX:

ALTER TABLE tablename ADD (columname1 datatype , columnname2 datatype,.., column namedatatype)

EXAMPLE:

ALTER TABLE student ADD (address varchar2(10)). WE CAN ALSO MODIFY COLUMNS DATA TYPES BY USING ALTER TABLE COMMAND.

SYNTAX :

-ALTER TABLE tablename MODIFY (columname1 datatype ,column name2data type ,.., column namedatatype)

EXAMPLE :

ALTER TABLE student MODIFY (address varchar2(20))

LIMITATIONS OF ALTER TABLE COMMAND:

Table name cannot be changed. Column name cannot be changed. Column cannot be dropped (deleted). Size of column cannot be changed if table data exists.

VARIOUS COMMONLY USED DML COMMANDS USED IN ORACLE:

INSERT:

This command is used to insert data in tables.

(a) Values are feeded in the same order in which columnswere created in create table command.

SYNTAX:

INSERT INTO tablename VALUES(value1,value2,..,valuen);

EXAMPLE:

INSERT INTO student VALUES(1,'amit', 21 , 88);

To feed values in limited columns, we must mentionfield names:

SYNTAX:
INSERT INTO student(columnname1,column name2, Columnname N) VALUES(value1, value2 , .. value N)

EXAMPLE

i. INSERT INTO student(rollno,name) VALUES(3, 'raj')
ii. INSERT INTO student(rollno,age,marks) VALUES(5, 25,65)

To feed values in all the columns, then mentioning column names is optional.

EXAMPLE

i. INSERT INTO student(rollno,name,age,marks) VALUES(4, 'aryan',24,74)
ii. INSERT INTO student VALUES(4, 'aryan',24,74)

UPDATE

This command is used to modify table records.where clause is used to specify the criteria of records tobe updated.

SYNTAX

UPDATE tablename SET columnname1=value1,columnname2=value2, columnnamen=value n where condition

EXMPLE

UPDATE student SET name='submit' WHERE rollno=5
UPDATE student SET name='Aman', age=28, marks=55 WHERE rollno=3
UPDATE student SET age=age+2 WHERE rollno=1

IF WHERE CLAUSE IS NOT SPECIFIED , THEN ALL THE RECORDS AREUPDATED.

SYNTAX

UPDATE tablename SET column name1=value1,columnname2=value2, columnnamen=valuen

EXAMPLE

UPDATE student SET age=age+2

DELETE

This command is used to delete table records.Where clause is used to specify the criteria of records tobe deleted.

SYNTAX

DELETE FROM tablename WHERE condition

EXAMPLE

DELETE FROM student WHERE rollno=1

IF WHERE CLAUSE IS NOT SPECIFIED , THEN ALL THE TABLERECORDS ARE DELETED

SYNTAX

DELETE FROM tablename

EXAMPLE

DELETE FROM student

SELECT

This command is used to select data from table.

SYNTAX

SELECT columnname1,columnname2,.., columnnamen FROMtablename

EXAMPLE

SELECT rollno,name,age FROM student

* IS USED TO READ DATA OF ALL THE COLUMNS.

SYNTAX

SELECT * FROM tablename

EXAMPLE

SELECT * FROM student

(B) WHERE CLAUSE IS USED TO SPECIFY THE CRITERIA OF RECORDS TO BE SELECTED.

SYNTAX

SELECT * FROM tablename WHERE criteria

EXAMPLE

SELECT * FROM student WHERE rollno=1

If we do not specify where clause, then all the records oftable are selected. Order by clause is used to sort all the records by aparticular field. by default all records are sorted in ascending order. we use keyword ask to sort records in descending order and keyword desk to sort records in descending order.

SYNTAX

SELECT * FROM tablename ORDER BY columnname

SELECT * FROM tablename ORDER BY columnname [ASC/DESC]

EXAMPLE

SELECT * FROM student ORDER BY marks

SELECT * FROM student ORDER BY marks ASC

SELECT * FROM student ORDER BY marks DESC

(E) WHERE AND ORDER BY CLAUSE CAN ALSO BE USED TOGETHER.

SYNTAX

SELECT * FROM tablename WHERE criteria ORDER BY columnname

EXAMPLE

SELECT * FROM student WHERE rollno>1 ORDER BY marks
DISTINCT CLAUSE IS USED TO READ ONLY DISTINCT VALUES.

SYNTAX

SELECT DISTINCT column name FROM table name

EXAMPLE

SELECT DISTINCT name FROM student

VARIOUS COMMONLY USED DCL COMMANDS USED INORACLE (SECURITY MANAGEMENT IN ORACLE) :-

EXAMPLE

(To perform DML operations on the table createdby another user):-

1. SELECT * FROM system.student
2. INSERT INTO system.student VALUES(9,'Z')
3. DELETE FROM system.student WHERE roll no>10
4. UPDATEsystem.student SET name='Ankit' WHERE roll no=9

GRANT COMMAND:
Grant command helps us to allow one user to access objects of another user variousprivileges which can be given to another user:-

SELECT , INSERT , UPDATE , DELETE , ALTER , INDEX

EXAMPLE
To grant only one privilege to another user):-

GRANT UPDATE ON student TO Scott

EXAMPLE
(To grant multiple privileges to another user):-
GRANT SELECT, INSERT ON student TO Scott

EXAMPLE(To grant all privileges to another user):-

GRANT ALL ON student TO ScottEXAMPLE(TO GRANT PRIVILEGES TO ANOTHER USER AND ALSO ALLOWING HIM TO GRANT SAME PRIVILEGES ON SAME OBJECT TO SOME OTHER USERS):

GRANT ALL ON student TO Scott WITH GRANT OPTION

REVOKE COMMAND:

Revoke command helps us to snatch permissions from oneuser to access objects of another user.

EXAMPLE(TO REVOKE ONLY ONE PRIVILEGE FROM ANOTHER USER):

REVOKE UPDATE ON student FROM Scott

EXAMPLE(TO REVOKE MULTIPLE PRIVILEGES FROM ANOTHERUSER):-

REVOKE SELECT, INSERT ON student FROM Scott

EXAMPLE(TO REVOKE ALL PRIVILEGES FROM ANOTHER USER):-

REVOKE ALL ON student FROM Scott

SOME COMMONLY USED COMMANDS:

TO SEE THE LIST OF ALLTHE TABLE, A USER HAS CREATED:
SELECT * FROM TAB

TO SEE THE STRUCTURE OF A TABLE:
DESCRIBE table name

TO CHANGE THE NAME OF A TABLE:
RENAME oldname TO newname

TO DELETE A TABLE:
DROP TABLE table name

TO PERFORM A MATHEMATICAL CALCULATION:-
SELECT 2*7 FROM DUAL

TO SEE CURRENT DATE:
SELECT SYSDATE FROM DUAL

OPERATORS IN ORACLE:-

MATHEMATICAL OPERATORS:-
/ - DIVISION* - MULTIPLICATION+ - ADDITION- - SUBSTRACTIONEXAMPLE:-
A.SELECT 2*6 FROM DUAL
B.SELECT AGE , AGE+2 FROM STUDENT

RELATIONAL OPERATORS :-
> - GREATER THAN>= - GREATER THAN OR EQUAL TO< - LESS THAN<= - LESS THAN OR EQUAL TO= - EQUAL TO<> - NOT EQUAL TO

LOGICAL OPERATORS:-
AND:-
This operator returns true when both the conditionspecified with and operator returns true. if any one of thecondition specified evaluates to false , then and operatorreturns false.

EXAPMLE:-
SELECT * FROM STUDENT WHERE ROLL NO>2 AND ROLL NO<5
UPDATE STUDENT SET AGE=AGE+2 WHERE ROLL NO>2 ANDROLL NO<5
OR:-this operator returns true when any one condition or both the conditions specified with or operator returns true. if both the conditions specified evaluate to false ,then or operator returns false.

EXAPMLE:-
A.SELECT * FROM STUDENT WHERE ROLL NO=2 OR ROLL NO=5
B.UPDATE STUDENT SET AGE=AGE+2 WHERE ROLL NO=2 ORROLL NO=5
NOT:-This operator negates the result of an expression.
Example:-
select * from student where not(roll no=2 or roll no=5)
update student set age=age+2 where not(roll no=2 or roll no=5)
between: This operator returns true if a value falls inbetween minimum value and maximum value.

SYNTAX:-
SELECT * FROM table name WHERE column name BETWEEN min.valueANDmax.value

EXAMPLE:-
Select * from student where roll no between 2 and 4

Not between:-

This operator returns true if a value doesnot fall in between minimum value and maximum value.

SYNTAX:-
SELECT * FROM table name WHERE column name NOT BETWEEN minvalueANDmaxvalue

EXAMPLE:-
SELECT * FROM student WHERE roll no NOT BETWEEN 2 AND 4
IN:-
THIS OPERATOR RETURNS TRUE IF A VALUE MATCHES TO ANY OF MULTIPLE VALUES SPECIFIED WITH IN OPERATOR.

SYNTAX:-
SELECT * FROM table name WHERE column name IN (value1, value2,.,valuen)

EXAMPLE:-SELECT * FROM student WHERE name IN ('aman','amit','sumit')
NOT IN:-
THIS OPERATOR RETURNS TRUE IF A VALUE DOES NOT MATCH TO ANY OF MULTIPLE VALUES SPECIFIED WITH NOT INOPERATOR.

SYNTAX:-
SELECT * FROM table name WHERE column name NOT IN (value1, value2,.,valuen)

EXAMPLE:-SELECT * FROM student WHERE name NOT IN ('Aman', 'Amit', 'Sumit')
LIKE:-
This operator returns true if a value matches to aspecified pattern of a string. Wildcards used with likeoperator are :-
% - IT DENOTES TO ZERO OR MORE CHARACTERS._ - IT DENOTES TO ONE CHARACTER.

SYNTAX:-
SELECT * FROM table name WHERE column name LIKE pattern

EXAMPLE:-
1. SELECT * FROM student WHERE name LIKE 'a%'
2.SELECT * FROM student WHERE name LIKE 'a%n'
3.SELECT * FROM student WHERE name LIKE 'a___'

ORACLE IN-BUILT FUNCTIONS:-
ORACLE INBUILT FUNCTIONS SERVE THE PURPOSE OFMANIPULATING DATA AND RETURN A VALUE.
ORACLE INBUILTFUNCTIONS ARE OF TWO TYPES:-

AGGREGATE FUNCTIONS/GROUP FUNCTIONS:-

These types of functions act on a set of values and return a single value. Some of the examples of aggregatefunctions are as follows:-

NUMERIC FUNCTIONS:-
A.SUM:-
THIS FUNCTION FINDS THE SUM OF A SET OF VALUES

EXAMPLE:-
SELECT SUM(marks) FROM studentB.
B.COUNT:-
THIS FUNCTION COUNTS A SET OF VALUES

EXAMPLE:- SELECT COUNT(marks) FROM student

C. AVG:-
THIS FUNCTION FINDS THE AVERAGE OF A SET OF VALUES
EXAMPLE:-
SELECT AVG(marks) FROM studentD.
MAX:-
THIS FUNCTION FINDS THE MAXIMUM VALUE FROM A SET OF VALUES

EXAMPLE:- SELECT MAX(marks) FROM studentE.
MIN:-
THIS FUNCTION FINDS THE MINIMUM VALUE FROM A SET OFVALUES

EXAMPLE:-
SELECT MIN(marks) FROM student
NOTE:-
MULTIPLE FUNCTIONS CAN BE USED IN A SINGLE QUERY.

EXAMPLE :-
SELECT SUM(marks) AS SUM_MARKS, COUNT(marks) "COUNT",AVG(marks) ,MAX(marks) , MIN(marks) FROM student

2. SCALAR FUNCTIONS/SINGLE ROW FUNCTIONS:-
These type of functions act on a single value and return asingle value.
some of the examples of scalar functionsare as follows:-

Numeric functions:-
A. POWER:-
THIS FUNCTION IS USED TO FIND THE POWER OF ANUMBER.

EXAMPLE:-i. SELECT POWER(5,3) FROM DUAL
ii. SELECT POWER(roll no,2) FROM studentB.
B.ABS:-
This function always returns a positive value

EXAMPLE:-
i.SELECT age , ABS(age) FROM studentC.
C.ROUND:-
This function rounds off a number uptospecified decimal points.

EXAMPLE:-
i.SELECT ROUND(53.56182,2) FROM DUAL
ii. SELECT ROUND(marks,2) FROM studentD.
D.SQRT:-
This function finds square root of a number.

EXAMPLE:-

i. SELECT SQRT(64) FROM DUAL
ii. SELECT SQRT(marks) FROM student

3. STRING FUNCTIONS:-
LOWER:-
This function converts all the characters of astring to lower case.

EXAMPLE:-
SELECT NAME, LOWER(name) FROM studentB.
UPPER:-
THIS FUNCTION CONVERTS ALL THE CHARACTERS OF ASTRING TO UPPER CASE.

EXAMPLE:-
SELECT NAME, UPPER(name) FROM studentC.
INITCAP:-
This function converts first character ofstring to uppercase and the remaining characters of a string to lowercase.

EXAMPLE:-
SELECT NAME, INITCAP(name) FROM studentD.
LENGTH:-
THIS FUNCTION RETURNS TOTAL NO OF CHARACTERSIN A STRING

EXAMPLE:-
SELECT NAME, LENGTH(name) FROM studentE.
LTRIM:-
This removes a particular character from the left part of a string. If the character to be removed is not specified , then by default , it removes spaces

EXAMPLE:-
i. SELECT NAME , LTRIM(NAME , 'a') FROM student
ii. SELECT LTRIM('aaaaabcdeaa' , 'a') FROM dualiii.SELECT LTRIM(' ABC ') FROM DUALF.

RTRIM:-
This removes a particular character from the right part of a string. if the character to be removed is not specified , then by default , it removes spaces.

EXAMPLE:-
i.SELECT name , RTRIM(name , 'a') FROM student
ii.SELECT RTRIM('aaaaabcdeaa' , 'a') FROM dual
iii.SELECT RTRIM(' abc ') FROM DUAL

LPAD:-
This function pads the string in the left part witha character so that length of string becomes equal to specified no of alphabets. by default, string is padded withspaces.

EXAMPLE:-
i.SELECT LPAD('abcdef',10,'*') FROM DUAL
ii.SELECT LPAD('abcdef',10) FROM DUAL
iii.SELECT LPAD(name,10,'-') FROM studentH.

RPAD:-
This function pads the string in right part with a character so that length of string becomes equal to specified no of alphabets. by default, string is padded with spaces.

EXAMPLE:-
i.SELECT RPAD('abcdef',10,'*') FROM DUAL
ii.SELECT RPAD('abcdef',10) FROM DUAL
iii. SELECT RPAD(name,10,'-') FROM studentI.

SUBSTR:-
This function returns a part of string(substring). We also mention starting position of substring and length of substring.

EXAMPLE:-
SELECT name,SUBSTR(name,2,3) FROM student

UNIT - II
CONSTRAINTS IN ORACLE

CONSTRAINTS IN ORACLE :-

Constraints are limitations or validation rules applied on various columns.
Not null:-
by applying this constraint on a column, user cannot enter null values within a column. if a user tries to enter null value in that column , oracle displays error message.

EXAMPLE(to apply NOT NULL constraint on a column):-

CREATE TABLE student1 (roll noNUMBER(4) NOT NULL , nameVARCHAR2(20))

CHECK :-
By applying this constraint on a column , we can limit the range of values or we can limit the domain of acolumn.

Ex(to apply CHECK constraint at COLUMN LEVEL):-

CREATE TABLE student2 (roll no NUMBER(3) CHECK (roll no BETWEEN 1AND 50) , city VARCHAR2(20) CHECK (city IN ('sirsa' , 'hisar' , 'rohtak')))

EXAMPLE(to apply CHECK constraint at TABLE LEVEL):-
CREATE TABLE student3 (roll no NUMBER(3), city VARCHAR2(20) ,CHECK (roll no BETWEEN 1 AND 50) , CHECK (city IN ('sirsa' , 'hisar' ,'rohtak')))

SPECIFYING DEFAULT VALUES OF A COLUMN:-

EXAMPLE:-
CREATE TABLE student9 (roll noNUMBER(3) , city VARCHAR2(20)DEFAULT 'sirsa');

UNIQUE CONSTRAINT :-
By applying this constraint on a column , User cannot enter duplicate values in a column. Only unique values are allowed. If a user tries to enter duplicate values inthat column, oracle displays error message.

EXAMPLE(to apply UNIQUE CONSTRAINT at COLUMN LEVEL):-

CREATE TABLE student4 (roll no NUMBER(3) UNIQUE , cityVARCHAR2(20))

PRIMARY KEY CONSTRAINT:-

EXAMPLE(TO ADD PRIMARY KEY USING ALTER TABLE COMMAND):-
ALTER TABLE student7 ADD PRIMARY KEY(roll no)

EXAMPLE(TO DELETE PRIMARY KEY USING ALTER TABLECOMMAND):-
ALTER TABLE student7 DROP PRIMARY KEY

EXAMPLE(To create primary key using create table command and giving primary key constraint a name):-

CREATE TABLE student8 (roll noNUMBER(3) CONSTRAINT pk_student8 PRIMARY KEY , name VARCHAR2(20))

EXAMPLE(TO CREATE PRIMARY KEY USING ALTER TABLE COMMAND AND GIVING PRIMARY KEY CONSTRAINT A NAME):-

ALTER TABLE student7 ADD CONSTRAINT pk_student7 PRIMARYKEY(roll no)

EXAMPLE(TO DELETE PRIMARY KEY USING ALTER TABLE COMMANDBY SPECIFYING ITS CONSTRAINT NAME):-

ALTER TABLE student8 DROP CONSTRAINT pk_student8

FOREIGN KEY CONSTRAINT:-

A foreign key is a column whose values are derived from primary key or unique key of other table. The table in which primary key is defined is known as primary table or master table or parent table. The table in which foreignkey is defined is known as foreign table or detail table or child table.

Some of the features of primary key/foreign key relationship:-

1. Primary key cannot accept duplicate values where as foreign key can accept.
2. Primary key cannot accept null values whereas foreignkey can accept
3. Values entered in foreign key of detail table must bepresent in primary key of primary table otherwise whene ver user tries to perform insert or update operation, oracle displays error.
4. To perform update or delete operations on primary key of master table, corresponding values must not bepresent in foreign key of foreign table, otherwise oracle display error.
5. In references clause, we need to mention only name of the parent table. Primary key of primary table is automatically attached to foreign key of foreign table.
6. If on delete cascade option has been set , then delete operation in master table will trigger delete operation inchild table.

EXAMPLE (To create foreign key by referring primarytable):-
CREATE TABLE fees7 (roll noNUMBER(3) REFERENCES student7, feesNUMBER(6,2))

EXAMPLE(To create foreign key by referring primary table, also specify foreign key constraint name):-
CREATE TABLE fees7 (roll noNUMBER(3) CONSTRAINT fk_fees7REFERENCES student7, fees NUMBER(6,2))

EXAMPLE(To delete foreign key using alter table commandby specifying foreign key constraint name):-
ALTER TABLE fees7 DROP CONSTRAINT fk_fees7

EXAMPLE(To create foreign key using alter table commandby referring primary table , also specify foreign keyconstraint name):-
ALTER TABLE fees7 ADD FOREIGN KEY(roll no) REFERENCES student7

EXAMPLE (To create foreign key by referring primary table, also setting on delete cascade option):-
CREATE TABLE fees7 (roll noNUMBER(3) CONSTRAINT fk_fees7REFERENCES student7 ON DELETE CASCADE, fees NUMBER(6,2))

Displaying Table Information

As with most relational databases, there may come a situation where you need to view the underlying metadata and look through the actual table list and ownership of your database. Thankfully, there are multiple ways to perform this relatively simple task in Oracle, so we'll briefly explore each option below to find which best suits your needs.

What are Oracle Data Dictionaries?

A data dictionary in Oracle is a collection of read-only tables that provide useful information about the database including schemas, users, privileges, and even auditing data. The values in these stored dictionaries are updated automatically by Oracle anytime a statement is executed on the server that modifies data.

From there, the read-only dictionaries can be read and queried just like any standard table, which as we'll see below provides some very useful functionality.

Viewing Tables Owned by Current user

At the most basic level, you may wish to view a list of all the tables owned by the current Oracle user. This can be accomplished with a simple SELECT query on the USER_TABLES data dictionary.

Once connected to Oracle, issue this statement:

```
SELECT
Table_name, owner
FROM
User_tables
ORDER BY
owner, table_name
```

Viewing Tables Accessible by Current User

In a situation where you're only interested in what tables the current Oracle user has access to, regardless of ownership, you'll use the ALL_TABLES data dictionary instead.

```
SELECT
table_name, owner

FROM
Alltables

ORDER BY
owner, table_name
```

It’s likely that this query will return far more results than you are interested in since you’re viewing everything even remotely accessible to the user, so you may wish to limit your query by specifying an appropriate owner, like so:

```
SELECT
table_name, owner

FROM
all_tables

WHERE
owner='schema_name'

ORDER BY
owner, table_name
```

Viewing All Tables
Lastly, when you absolutely need to view every table in the system, look no further than the great and
powerful DBA_TABLES data dictionary.

```
SELECT
table_name, owner

FROM
dba_tables

WHERE
owner='schema_name'

ORDER BY
owner, table_name
```

It is important to note that this final DBA_TABLES dictionary may require user privileges beyond what the current user has. If necessary, you may need to be granted the SELECT ANY

DICTIONARY privilege or the SELECT_CATALOG_ROLE role. More information on granting these privileges can be found in the official documentation.

Altering an Existing Table

The Oracle ALTER TABLE statement to add a column, modifies a column, drop a column, rename a column or rename a table (with syntax, examples and practice exercises).

Description

The Oracle ALTER TABLE statement is used to add, modify, or drop/delete columns in a table. The Oracle ALTER TABLE statement is also used to rename a table.

Add column in table

Syntax
To ADD A COLUMN in a table, the Oracle ALTER TABLE syntax is:

```
ALTER TABLE table_name
  ADD column_namecolumn_definition;
```

Example

```
ALTER TABLE customers
  ADD customer_namevarchar2(45);
```

This Oracle ALTER TABLE example will add a column called customer_name to the customers table that is a data type of varchar2(45).

In a more complicated example, you could use the ALTER TABLE statement to add a new column that also has a default value:

```
ALTER TABLE customers
ADD city varchar2(40) DEFAULT 'Seattle';
```

In this example, the column called city has been added to the customers table with a data type of varchar2(40) and a default value of 'Seattle'.
Add multiple columns in table

Syntax
To ADD MULTIPLE COLUMNS to an existing table, the Oracle ALTER TABLE syntax is:

```
ALTER TABLE table_name
ADD (column_1 column_definition,
column_2 column_definition,
     ...column_ncolumn_definition);
```

Example

Let's look at an example that shows how to add multiple columns in an Oracle table using the ALTER TABLE statement.

For example:

```
ALTER TABLE customers
ADD (customer_namevarchar2(45),
city varchar2(40) DEFAULT 'Seattle');
```

This Oracle ALTER TABLE example will add two columns, customer_name as a varchar2(45) field and city as a varchar2(40) field with a default value of 'Seattle' to the customers table.

Modify column in table

Syntax

To MODIFY A COLUMN in an existing table, the Oracle ALTER TABLE syntax is:

```
ALTER TABLE table_name
MODIFY column_namecolumn_type;
```

Example

Let's look at an example that shows how to modify a column in an Oracle table using the ALTER TABLE statement.

For example:

```
ALTER TABLE customers
MODIFY customer_namevarchar2(100) NOT NULL;
```

This Oracle ALTER TABLE example will modify the column called customer_name to be a data type of varchar2(100) and force the column to not allow null values.

In a more complicated example, you could use the ALTER TABLE statement to add a default value as well as modify the column definition:

```
ALTER TABLE customers
MODIFY city varchar2(75) DEFAULT 'Seattle' NOT NULL;
```

In this example, the ALTER TABLE statement would modify the column called city to be a data type of varchar2(75), the default value would be set to 'Seattle' and the column would be set to not allow null values.

Modify Multiple columns in table

Syntax

To MODIFY MULTIPLE COLUMNS in an existing table, the Oracle ALTER TABLE syntax is:

```
ALTER TABLE table_name
MODIFY (column_1 column_type,
column_2 column_type,
    ...
column_ncolumn_type);
```

Example

Let's look at an example that shows how to modify multiple columns in an Oracle table using the ALTER TABLE statement.

For example:

```
ALTER TABLE customers
MODIFY (customer_namevarchar2(100) NOT NULL,
city varchar2(75) DEFAULT 'Seattle' NOT NULL);
```

This Oracle ALTER TABLE example will modify both the customer_name and city columns. The customer_name column will be set to a varchar2(100) data type and not allow null values. The city column will be set to a varchar2(75) data type, its default value will be set to 'Seattle', and the column will not allow null values.

Drop column in table

Syntax

To DROP A COLUMN in an existing table, the Oracle ALTER TABLE syntax is:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example

Let's look at an example that shows how to drop a column in an Oracle table using the ALTER TABLE statement.

For example:

```
ALTER TABLE customers
DROP COLUMN customer_name;
```

This Oracle ALTER TABLE example will drop the column called customer_name from the table called customers.

Rename column in table

Syntax

To RENAME A COLUMN in an existing table, the Oracle ALTER TABLE syntax is:

```
ALTER TABLE table_name
RENAME COLUMN old_name TO new_name;
```

Example

Let's look at an example that shows how to rename a column in an Oracle table using the ALTER TABLE statement.

For example:

```
ALTER TABLE customers
RENAME COLUMN customer_name TO cname;
```

This Oracle ALTER TABLE example will rename the column called customer_name to cname.

Rename table

Syntax
To RENAME A TABLE, the Oracle ALTER TABLE syntax is:

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

Example

Let's look at an example that shows how to rename a table in Oracle using the ALTER TABLE statement.

For example:

```
ALTER TABLE customers
RENAME TO contacts;
```

This Oracle ALTER TABLE example will rename the customers table to contacts.

Practice Exercise #1:
Based on the departments table below, rename the departments table to depts.

```
CREATE TABLE departments
(department_id number(10) NOT NULL,
department_namevarchar2(50) NOT NULL,
CONSTRAINT departments_pk PRIMARY KEY (department_id)
);
```

Solution for Practice Exercise #1:
The following Oracle ALTER TABLE statement would rename the departments table to depts:

```
ALTER TABLE departments
RENAME TO depts;
```

Practice Exercise #2:
Based on the employees table below, add a column called bonus that is a number(6) data type.

```
CREATE TABLE employees
( employee_number number(10) NOT NULL,
employee_namevarchar2(50) NOT NULL,
department_id number(10),
CONSTRAINT employees_pk PRIMARY KEY (employee_number)
);
```

Solution for Practice Exercise #2:
The following Oracle ALTER TABLE statement would add a bonus column to the employees table:

```
ALTER TABLE employees
ADD bonus number(6);
```

Practice Exercise #3:
Based on the customers table below, add two columns - one column called contact_name that is a varchar2(50) datatype and one column called last_contacted that is a date datatype.

```
CREATE TABLE customers
(customer_id number(10) NOT NULL,
customer_namevarchar2(50) NOT NULL,
address varchar2(50),
city varchar2(50),
state varchar2(25),
zip_codevarchar2(10),
 CONSTRAINT customers_pk PRIMARY KEY (customer_id)
);
```

Solution for Practice Exercise #3:
The following Oracle ALTER TABLE statement would add the contact_name and last_contacted columns to the customers table:

```
ALTER TABLE customers
ADD (contact_namevarchar2(50),
last_contacted date);
```

Practice Exercise #4:
Based on the employees table below, change the employee_name column to a varchar2(75) data type.

```
CREATE TABLE employees
(employee_number number(10) NOT NULL,
employee_namevarchar2(50) NOT NULL,
department_id number(10),
CONSTRAINT employees_pk PRIMARY KEY (employee_number)
);
```

Solution for Practice Exercise #4:
The following Oracle ALTER TABLE statement would change the datatype for the employee_name

column to varchar2(75):

```
ALTER TABLE employees
MODIFY employee_namevarchar2(75);
```

Practice Exercise #5:

Based on the customers table below, change the customer_name column to NOT allow null values and change the state column to a varchar2(2) data type.

```
CREATE TABLE customers
( customer_id number(10) NOT NULL,
customer_namevarchar2(50),
address varchar2(50),
city varchar2(50),
state varchar2(25),
zip_codevarchar2(10),
CONSTRAINT customers_pk PRIMARY KEY (customer_id)
);
```

Solution for Practice Exercise #5:

The following Oracle ALTER TABLE statement would modify the customer_name and state columns accordingly in the customers table:

```
ALTER TABLE customers
MODIFY (customer_namevarchar2(50) NOT NULL,
state varchar2(2));
```

Practice Exercise #6:

Based on the employees table below, drop the salary column.

```
CREATE TABLE employees
(employee_number number(10) NOT NULL,
employee_namevarchar2(50) NOT NULL,
department_id number(10),
salary number(6),
CONSTRAINT employees_pk PRIMARY KEY (employee_number)
);
```

Solution for Practice Exercise #6:

The following Oracle ALTER TABLE statement would drop the salary column from the employees table:

```
ALTER TABLE employees
DROP COLUMN salary;
```

Practice Exercise #7:

Based on the departments table below, rename the department_name column to dept_name.

```
CREATE TABLE departments
(department_id number(10) NOT NULL,
department_namevarchar2(50) NOT NULL,
CONSTRAINT departments_pk PRIMARY KEY (department_id)
);
```

Solution for Practice Exercise #7:

The following Oracle ALTER TABLE statement would rename the department_name column to dept_name in the departments table:

```
ALTER TABLE departments
RENAME COLUMN department_name TO dept_name;
```

GROUPING:-

Grouping allows us to group records based on distinct values of a particular column. Having clause allows usto filter summary of groups. Condition in having clause should be based on either aggregate functions or on thecolumn by which groups have been created.

EXAMPLE(to perform grouping and using an aggregate function):-
SELECT deptno,SUM(sal) FROM EMP GROUP BY deptno

EXAMPLE(to perform grouping and using multiple aggregate function):-
SELECT deptno,SUM(sal),COUNT(sal), AVG(sal), MAX(sal),MIN(sal) FROM EMP GROUP BY deptno

EXAMPLE(to perform grouping and using an aggregate function and to perform filtering of groups based on same aggregate function in having clause):-
SELECT deptno,SUM(sal) FROM EMP GROUP BY deptno HAVINGSUM(sal)>9000

EXAMPLE(to perform grouping and using an aggregate function and to perform filtering of groups based on different aggregate function in having clause):-

SELECT deptno,SUM(sal) FROM EMP GROUP BY deptno HAVINGCOUNT(sal)<6

EXAMPLE(to perform grouping and using an aggregatefunction and to perform filtering of groups based on thecolumn by which grouping has been performed in havingclause):-
SELECT deptno,SUM(sal) FROM EMP GROUP BY deptno HAVING deptno=20

SUBQUERIES:-

A subquery is a SQL query which is placed inside another sql query. the statement containing a sub query is knownas parent statement.parent query.

EXAMPLE:-
```
SELECT * FROM student7 WHERE roll no IN (
SELECT roll no FROM fees7
)
```

SET OPERATORS

UNION/INTERSECT/MINUS CLAUSE:-
UNION:-

by using union clause . multiple queries are put together and their output is combined. Output of union clause = records only in query 1 +records in only query 2 + a single set of records which iscommon in both the queries.

EXAMPLE:-
```
SELECT roll no FROM student6 UNION SELECT roll no FROM student7
```

INTERSECT:-

By using intersect clause. Multiple queries are puttogether and their output is combined. Output of intersect clause = a single set of records whichis common in both the queries.

Example:-

Select roll no from student6 intersect select roll no from student7

Minus:-

By using union clause. Multiple queries are put togetherand their output is combined. Output of minus clause = records in query 1 - records inquery 2(only those records are displayed from query 1 whicharenot present in query 2)

Example:-select roll no from student6 minus select roll no from student7

UNIT – III
JOINS

JOINS:-

Joins are used to combine the data of two or more than two tables. Joins provide a great flexibility to the user tosee the records of multiple tables related to each other. With the help of joins, summary from various tables caneasily be obtained.
joins are of multiple types:-

INNER JOIN:-

Inner joins are used to combine the data oftwo or more than two tables in which only those recordsof both the tables are displayed which satisfy a specifiedcondition

EXAMPLE:-SELECT student.rollno,name, fees.rollno,fees FROM student INNER JOIN feesONstudent.roll no=fees.roll no

LEFT OUTER JOIN:-

Left outer joins are used to combine thedata of two or more than two tables in which all therecords of left table are displayed and only thoserecords of right table are displayed which satisfy aspecified condition.

EXAMPLE:-
SELECT student.rollno,name, fees.rollno,fees FROM student LEFT OUTER JOINfees ON student.roll no=fees.roll no
RIGHT OUTER JOIN:-
right outer joins are used to combinethe data of two or more than two tables in which all therecords of right table are displayed and only thoserecords of left table are displayed which satisfy aspecified condition.

EXAMPLE:-
SELECT student.rollno,name, fees.Rollno,fees FROM student RIGHT OUTERJOIN fees ON student.roll no=fees.roll no
FULL OUTER JOIN:-
FULL OUTER JOINS ARE USED TO COMBINE THEDATA OF TWO OR MORE THAN TWO TABLES IN WHICH ALL THERECORDS OF BOTH TABLES ARE DISPLAYED
EXAMPLE:-
SELECT student.rollno,name, fees.rollno,fees FROM student FULL OUTERJOIN fees ON student.roll no=fees.roll no
SELF JOIN :-

In case of joins, multiple instance of same tableare opened in memory with different alias. And then a joinoperation is performed between those two instance ofsame table by using their alias. So, when a table is joinedto itself is known as self-join
.
EXAMPLE:-(to display details of those employees which have amanager)

1. SELECT emp_1.empno,emp_1.name,emp_1.mngrno,emp_2.name managerFROM employee emp_1,employee emp_2 WHEREemp_1.mngrno=emp_2.empno

2. SELECT emp_1.empno,emp_1.name,emp_1.mngrno,emp_2.name managerFROM employee emp_1 INNER JOIN employee emp_2 ONemp_1.mngrno=emp_2.empno(TO DISPLAY DETAILS OF THOSE EMPLOYEES WHICH HAVE AMANAGER OR NOT)

3. SELECT emp_1.empno,emp_1.name,emp_1.mngrno,emp_2.name managerFROM employee emp_1 LEFT OUTER JOIN employee emp_2 ONemp_1.mngrno=emp_2.empno

VIEWS:-

Views are based on a table. They provide the user flexibility to have a limits access on table. it can be usedto prevent users to access all columns of data. views canbe used to obtain summary from various table by using clauses like union, group by etc. views can be created onsingle table or multiple tables. Views itself have no data. They obtain data from tables at run time on which theyare based. Views which can be used to modify the recordsof base table are known as updateable views. Views which cannot be used to modify the records ofbasetableareknownasreadonlyviews.

EXAMPLE(TO CREATE A VIEW BASED ON A SINGLE TABLE(INCLUDING PRIMARY KEY)):-
CREATE VIEW v1 AS SELECT empno,Ename ,sal FROM emp
In this case , all insert , update and delete operations canbe performed on emp table with the help of view v1.

EXAMPLE(TO CREATE A VIEW BASED ON A SINGLE TABLE(EXCLUDING PRIMARY KEY)):-
CREATE VIEW v2 AS SELECT E name ,sal FROM empIN
This case, update and delete operations can beperformed on emp table with the help of view v1 but insert operation cannotbe performed because view does notinclude primary key empno of table emp.

Examples(to create a read only view based on a singletable using aggregate function or various clauses likeunion, intersect etc.):-

1. CREATE VIEW v3 AS SELECT deptno,SUM(sal) SUM_SAL FROM EMPGROUP BY deptno

2. CREATE VIEW v4 AS SELECT SUM(sal) SUM_SAL, COUNT(sal)COUNT_SAL,AVG(sal) AVG_SAL, MAX(sal) MAX_SAL,MIN(sal) MIN_SALFROM emp

3. CREATE VIEW v5 AS SELECT roll no FROM student6 UNION SELECTroll no FROM student7
In these cases in which aggregate functions or clauses like union, intersect , minus , group by , distinct are beingused. These types of views are read only. Wecan notperform insert , update or delete operation on these views.

EXAMPLE(TO CREATE A VIEW BASED ON MULTIPLE TABLES ANDTHOSE TABLE DO NOT HAVE PRIMARY KEY-FOREIGN KEYRELATIONSHIP):-

CREATE VIEW v6 AS SELECT student.roll no "s_rollno",name, fees.rollno"f_rollno",fees FROM student INNER JOIN fees ON student.roll no=fees.roll no

If a view is based on multiple tables and those tables donot have primary key-foreign key relationship , then wecan not perform insert , update or delete operation oncreated view. If a view is based on multiple tables and those tableshave primary key-foreign key relationship then:-

1. Insert operation is not allowed on view.
2. Delete operation performed on view affect only childtable records
3. Update operation performed on view affect only childtable records.

If try to update columns of primary tablein update command, then oracle displays error message\.

EXAMPLE(TO DELETE A VIEW):-
DROP VIEW v1;

INDEXES:-

An index is an ordered list of contents of a column orgroup of columns of a table..indexing involves formation of an index table independent of the base table on which index has been created. an index table has two columns:-

1. First column will hold stored data in sorted order ofthe column of base table on which index is created.
2. Second column identifies the location of record in oracle database. This address field is called rowid.indexing is an strategy to search and sort records in the table fastly. It is a technique to improve the speed, without indexing, sequentotal search operation is performed on table which degrades the performance ofsystem.

TOO MANY INDEXES ON A TABLE:-

Each time, a record isinserted in a table, oracle engine inserts a record both indata files and index tables. In index table , records aremaintained in ascending order. if too many indexes arecreated on a table , it will take longer to insert a recording a table. Although indexes enhance the performance of select command, it degrades the performance of insertcommand. so a proper balance between these two need tobe maintained.Categories of index based on uniqueness of indexedcolumn:-

1. DUPLICATE INDEXES:- DUPLICATE INDEXES ALLOW DUPLICATEVALUES FOR INDEXED COLUMN.

EXAMPLE:-CREATE INDEX idx_rn_fees ON fees(roll no);

2. UNIQUE INDEXES:- UNIQUE INDEXES ALLOW ONLY UNIQUEVALUES FOR INDEXED COLUMN.

EXAMPLE:-CREATE UNIQUE INDEX idx_rn_fees ON fees(roll no);

CATEGORIES OF INDEX BASED ON NO OF COLUMNS IN AN INDEX:-

1. SIMPLE INDEX:- AN INDEX CREATED ON SINGLE COLUMN IS SIMPLEINDEX.
EXAMPLE:-
CREATE INDEX idx_rn_fees ON fees(roll no);

2. COMPOSITE INDEX:- AN INDEX CREATED ON MULTIPLE COLUMNSIS COMPOSITE INDEX.
EXAMPLE:-
CREATE INDEX idx_rn_fees ON fees(class,roll no);

EXAMPLE(TO DELETE AN INDEX):-
DROP INDEX idx_rn_fees

SEQUENCES:-

Sequence is an object that can generate numeric values. Sequences are used while inserting data in a column oftable. it automatically increments a values in a column.it also provides the option to insert only unique values.by using sequences , we can also mention minimum valueand maximum value entered inside a column.

SYNTAX:-
CREATE SEQUENCE seq_name[INCREMENT BY valueSTART WITH valueMINVALUE value / NOMINVALUEMAXVALUE value / NOMAXVALUECYCLE/NOCYCLECACHE value /NOCACHE]DESCRIPTION:-

INCREMENT BY:-
It specifies the interval between two sequence numbers. Minvalue:-
It specifies minimum sequence number. if wedon't want to specify any minimum value then ,nominvalue clause is used.

MAXVALUE:-
It specifies maximum sequence number. if wedon't want to specify any maximum value then ,nomaxvalue clause is used.

CYCLE / NOCYCLE:-

IT SPECIFIES WHETHER SEQUENCE NUMBERSARE REPEATED AFTER MAXIMUM VALUE IS ENTERED.

CACHE:-

It specifies how many sequence numbers are pre-allocated in memory for faster access. it we do not wantto pre-allocate any sequence number , then no cache clause is used.

EXAMPLE(TO CREATE A SEQUENCE):-
CREATE SEQUENCE seq_rn INCREMENT BY 1 START WITH 8 MINVALUE1 MAXVALUE 10 CYCLE CACHE 2

EXAMPLE(TO INSERT A VALUE IN A TABLE USING SEQUENCE):-
INSERT INTO student VALUES(seq_rn.NEXTVAL , 'a')

EXAMPLE(TO ALTER A SEQUENCE):-
ALTER SEQUENCE seq_rn INCREMENT BY 2 MINVALUE 1 MAXVALUE20

EXAMPLE(TO DROP A SEQUENCE):-
DROP SEQUENCE seq_rn

Oracle Data Types

CHAR Data type

The CHAR datatype stores fixed-length character strings. When you create a table with a CHAR column, you must specify a string length (in bytes or characters) between 1 and 2000 bytes for the CHAR column width. The default is 1 byte. Oracle then guarantees that:

When you insert or update a row in the table, the value for the CHAR column has the fixed length. If you give a shorter value, then the value is blank-padded to the fixed length.If a value is too large, Oracle Database returns an error.

VARCHAR2 and VARCHAR Datatypes

The VARCHAR2 datatype stores variable-length character strings. When you create a table with a VARCHAR2 column, you specify a maximum string length (in bytes or characters) between 1 and 4000 bytes for the VARCHAR2 column. For each row, Oracle Database stores each value in the column as a variable-length field unless a value exceeds the column's maximum length, in which case Oracle
Database returns an error. UsingVARCHAR2 and VARCHAR saves on space used by the table.

For example, assume you declare a column VARCHAR2 with a maximum size of 50 characters. In a single-byte character set, if only 10 characters are given for the VARCHAR2 column value in a particular row, the column in the row's row piece stores only the 10 characters (10 bytes), not 50.

Overview of Numeric Datatypes

The numeric datatypes store positive and negative fixed and floating-point numbers, zero, infinity, and values that are the undefined result of an operation (that is, is "not a number" or NAN).

NUMBER Data type
Floating-Point Numbers

The NUMBER data type stores fixed and floating-point numbers. Numbers of virtually any magnitude can be stored and are guaranteed portable among different systems operating Oracle Database, up to 38 digits of precision.
The following numbers can be stored in a NUMBER column:
Positive numbers in the range 1 x 10-130 to 9.99...9 x 10125 with up to 38 significant digits

Negative numbers from -1 x 10-130 to 9.99...99 x 10125 with up to 38 significant digits

For numeric columns, you can specify the column as:
column_name NUMBER
Optionally, you can also specify a precision (total number of digits) and scale (number of digits to the right of the decimal point):

column_name NUMBER (precision, scale)

If a precision is not specified, the column stores values as given. If no scale is specified, the scale is zero.

Floating-Point Numbers

Oracle Database provides two numeric data types exclusively for floating-point numbers: BINARY_FLOAT and BINARY_DOUBLE. They support all of the basic functionality provided by the NUMBER data type. However, while NUMBER uses decimal precision, BINARY_FLOAT and BINARY_DOUBLE use binary precision. This enables faster arithmetic calculations and usually reduces storage requirements.

BINARY_FLOAT and BINARY_DOUBLE are approximate numeric data types. They store approximate representations of decimal values, rather than exact representations. For example, the value 0.1 cannot be exactly represented by either BINARY_DOUBLE or BINARY_FLOAT. They are frequently used for scientific computations. Their behavior is similar to the data types FLOAT and DOUBLE in Java and XMLSchema.

Overview of DATE Data type

The DATE data type stores point-in-time values (dates and times) in a table. The DATE data type stores the year (including the century), the month, the day, the hours, the minutes, and the seconds (after midnight).

Oracle Database can store dates in the Julian era, ranging from January 1, 4712 BCE through December 31, 9999 CE (Common Era, or 'AD'). Unless BCE ('BC' in the format mask) is specifically used, CE date entries are the default.

Oracle Database uses its own internal format to store dates. Date data is stored in fixed-length fields of seven bytes each, corresponding to century, year, month, day, hour, minute, and second.

For input and output of dates, the standard Oracle date format is DD-MON-YY, as follows:

'13-NOV-92'

You can change this default date format for an instance with the parameter NLS_DATE_FORMAT. You can also change it during a user session with the ALTER SESSION statement. To enter dates that are not in standard Oracle date format, use the TO_DATE function with a format mask:

TO_DATE ('November 13, 1992', 'MONTH DD, YYYY')

Oracle Database stores time in 24-hour format—HH:MI:SS. By default, the time in a date field is 00:00:00 A.M. (midnight) if no time portion is entered. In a time-only entry, the date portion defaults to the first day of the current month. To enter the time portion of a date, use the TO_DATE function with a format mask indicating the time portion, as in:

```
INSERT INTO birthdays (bname, bday) VALUES
('ANDY',TO_DATE('13-AUG-66 12:56 A.M.','DD-MON-YY HH:MI A.M.'));
```

A foreign key constraint (also called a referential integrity constraint) designates a column as the foreign key and establishes a relationship between that foreign key and a specified primary or unique key, called the referenced key. A composite foreign key designates a combination of columns as the foreign key.

Database System Architectures
There are a number of database system architectures presently in use.

One must examine several criteria:
Where do the data and DBMS reside?
Where are the application program executed (e.g., which CPU)? This may include the user interface.
Where are business rules (applications logic) enforced?

Traditional Mainframe Architecture
Database (or files) resides on a mainframe computer.
Applications are run on the same mainframe computer. e.g., COBOL programs or JCL scripts that access the database.

Business rules are enforced in the applications running on the mainframe.
Multiple users access the applications through simple terminals (e.g., IBM 3270 terminals or VT220 terminals) that have no processing power of their own. User interface is text-mode screens.

Advantages:
Excellent security and control over applications
High reliability - years of proven MF technology
Relatively low incremental cost per user (just add a terminal)

Disadvantages:
Unable to effectively serve advanced user interfaces
Users unable to effectively manipulate data outside of standard applications
Personal Computer - Stand-Alone Database
Database(or files) reside on a PC - on the hard disk.

Applications run on the same PC and directly access the database. In such cases, the application is the DBMS.

Business rules are enforced in the applications running on the PC.
A single user accesses the applications.
File Sharing Architecture
PCs are connected to a local area network (LAN).
A single file server stores a single copy of the database files.
PCs on the LAN map a drive letter (or volume name) on the file server.
Applications run on each PC on the LAN and access the same set of files on the file server. The application is also the DBMS.

Business rules are enforced in the applications - Also, the applications must handle concurrency control. Possibly by file locking.

Each user runs a copy of the same application and accesses the same files.

Example: Sharing MS Access files on a file server.

Advantages:
(limited) Ability to share data among several users
Costs of storage spread out among users
Most components are now commodity items - prices falling

Disadvantages:
Limited data sharing ability - a few users at most
Classic Client/Server Architecture

Client machines:
Run own copy of an operating system.
Run one or more applications using the client machine's CPU, memory.
Application communicates with DBMS server running on server machine through a Database Driver
Database driver (middleware) makes a connection to the DBMS server over a network.

Examples of clients: PCs with MS Windows operating system. Forms and reports developed e.g. Oracle Developer/2000, etc.

Server Machines:
Run own copy of an operating system.
Run a Database Management System that manages a database.

Provides a Listening daemon that accepts connections from client machines and submits transactions to DBMS on behalf of the client machines.

Examples: Sun Sparc server running UNIX operating system. RDBMS such as Oracle Server, Sybase, Informix, DB2, etc.
PC with Windows NT operating system.
Middleware:
Small portion of software that sits between client and server.

Establishes a connection from the client to the server and passes commands (e.g., SQL) between them.

Examples:
For Oracle: SQL*Net (or Net8) running on both client and server.
For Sybase: Sybase Open Client and Open Server.
For IBM DB2: Client Application Enablers
Business rules may be enforced at:
The client application - so called "Fat Clients".
Entirely on the database server - so called "Thin Clients"
A Mix of both.

We can also call this a "Two Tier" or "Two Level" Client/Server Architecture

Advantages of client/server:
Processing of the entire Database System is spread out over clients and server.
DBMS can achieve high performance because it is dedicated to processing transactions (not running applications).
Client Applications can take full advantage of advanced user interfaces such as Graphical User Interfaces.

Disadvantages of client/server:
Implementation is more complex because one needs to deal with middleware and the network.
It is possible the network is not well suited for client/server communications and may become saturated.
Additional burden on DBMS server to handle concurrency control, etc.

As more business rule logic is programmed into the client side applications, they can become unwieldy. Stored procedures and triggers can help in this case.

Examples:
Oracle Server RDBMS running on a server.
Oracle Developer/2000 running on a client PC (connected with SQL*Net as the middleware).
Oracle Server RDBMS running on a server.
Sybase-PowerSoft PowerBuilder running on a client PC.
Oracle Server RDBMS running on a server.
MS Visual Basic application running on a client PC.
Sybase Adaptive Server RDBMS running on a server.

C++ application running on a UNIX workstation.
Three-Tier Client Server
Same general situation as traditional client/server.
Difference is the enforcement of business rules (applications logic) is done in a "middle" layer. Sometimes called "application logic" server.

Another option is to aggregate transactions from multiple users with a Transaction Monitor
Is "web friendly". The web browser becomes the user interface on the client.

Advantages:
Centralize applications logic (one place to make changes)
Relieves clients from having to load up on applications logic (the "fat client" approach).
Frees up DBMS server to efficiently process transactions

Disadvantages:
System complexity - extremely complex to program and debug
Security issues

Examples:
Oracle Web Applications Server running on the "Middle tier"
M Internet Explorer (web browser) running on a client PC.
Sybase Adaptive Server RDBMS running on a server.

UNIT - IV
FUNDAMENTALS OF PL/SQL

Fundamentals of PL/SQL

Like other programming languages, PL/SQL has a character set, reserved words, punctuation, data types, and fixed syntax rules.

Character Sets and Lexical Units

PL/SQL programs are written as lines of text using a specific set of characters:

- ■ Upper- and lower-case letters A .. Z and a ..z
- ■ Numerals 0 .. 9
- ■ Symbols ( ) + – * / <>= ! ~ ^ ; : . ' @ % , " # $ & _ | { } ? [ ]
- ■ Tabs, spaces, and carriage returns

PL/SQL keywords are not case-sensitive, so lower-case letters are equivalent to corresponding upper-case letters except within string and character literals.

■ Delimiters (simple and compound symbols) – A delimiter is a simple ompound symbol
or c
that has a special meaning to PL/SQL. For example, you use delimiters to represent arithmetic operations such as addition and subtraction.

■ Identifiers(which include reserved words) – We use identifiers to name PL/SQL program items and units, which include constants, variables, exceptions, cursors, cursor variables, subprograms, and packages.

■ Literals – A literal is an explicit numeric, character, string, or BOOLEAN value not represented by an identifier.

■ Comments – The PL/SQL compiler ignores comments, but you should not. Adding comments to your program promotes readability and aids understanding. Generally, you use comments to describe the purpose and use of each code segment. PL/SQL supports two comment styles: single-line and multi-line.

Oracle PL/SQL Data Types: Character, Number, Boolean, Date, LOB

What is PL/SQL Data types?

A data type is associated with the specific storage format and range constraints. In Oracle, each value or constant is assigned with a data type.

Basically, it defines how the data is stored, handled and treated by Oracle during the data storage and processing.

The main difference between PL/SQL and SQL data types is, SQL data type are limited to table column while the PL/SQL data types are used in the PL/SQL blocksCHARACTER Data Type

- ➢ NUMBER Data Type
- ➢ BOOLEAN Data Type
- ➢ DATE Data Type
- ➢ LOB Data Type
- ➢ CHARACTER Data Type:

This data type basically stores alphanumeric characters in string format.

The literal values should always be enclosed in single quotes while assigning them to CHARACTER data type.

This character data type is further classified as follows:

- CHAR Data type (fixed string size)
- VARCHAR2 Data type (variable string size)
- VARCHAR Data type
- NCHAR (native fixed string size)
- NVARCHAR2 (native variable string size)
- LONG and LONG RAW

Data TypeDescriptionSyntax

CHAR This data type stores the string value, and the size of the stri ng is fixed at the time of declaring the variable.

Oracle would be blank-padded the variable if the variable didn't occu py the entire size that has been declared for it, Hence Oracle will allocate the memory for declared size even if the variable didn't occupy it fully.

The size restriction for this data type is 1-2000 bytes.
CHAR data type is more appropriate to use where ever fixed the size of data will be handled.

```
grade CHAR;
manager CHAR (10):= 'guru99';
```

Syntax Explanation:
The first declaration statement declared the variable 'grade' of CHAR data type with the maximum size of 1 byte (default value).

The second declaration statement declared the variable 'manager' of CHAR data type with the maximum size of 10 and assigned the value 'guru99' which is of 6 bytes. Oracle will allocate the memory of 10 bytes rather than 6 bytes in this case.

VARCHAR2
This data type stores the string, but the length of the string is not fixed. The size restriction for this data type is 1-4000 bytes for table column size and 1-32767 bytes for variables.

The size is defined for each variable at the time of variable declaration.But Oracle will allocate memory only after the variable is defined, i.e., Oracle will consider only the actual length of the string that is stored in a variable for memory allocation rather than the size that has been given for a variable in the declaration part.

It is always good to use VARCHAR2 instead of CHAR data type to optimize the memory usage.

```
manager VARCHAR2(10) := ‘guru99';
```

Syntax Explanation:

The above declaration statement declared the variable 'manager' of VARCHAR2 data type with the maximum size of 10 and assigned the value 'guru99' which is of 6 bytes. Oracle will allocate memory of only 6 bytes in this case.

VARCHAR

This is synonymous with the VARCHAR2 data type.It is always a good practice to use VARCHAR2 instead of VARCHAR to avoid behavioral changes.manager VARCHAR(10) := ‘guru99';

Syntax Explanation:

The above declaration statement declared the variable 'manager' of VARCHAR data type with the maximum size of 10 and assigned the value 'guru99' which is of 6 bytes. Oracle will allocate memory of only 6 bytes in this case. (Similar to VARCHAR2)

NCHARThis data type is same as CHAR data type, but the character set will of the national

Character set.
This character set can be defined for the session using NLS_PARAMETERS.
The character set can be either UTF16 or UTF8.
The size restriction is 1-2000 bytes.
native NCHAR(10);
Syntax Explanation:

The above declaration statement declares the variable 'native' of NCHAR data type with the maximum size of 10.

The length of this variable depends upon the (number of lengths) per byte as defined in the character set.

NVARCHAR2 This data type is same as VARCHAR2 data type, but the character set will be of the national character set.

This character set can be defined for the session using NLS_PARAMETERS.
The size restriction is 1-4000 bytes.
Native varNVARCHAR2(10):='guru99';

Syntax Explanation:

The above declaration statement declares the variable 'Native_var' of NVARCHAR2 data type with the maximum size of 10.

LONG and LONGRAW

This data type is used to store large text or raw data up to the maximum size of 2GB.These are mainly used in the data dictionary.LONG data type is used to store character set data, while LONG RAW is used to store data in binary format.LONG RAW data type accepts media objects, images, etc. whereas LONG works only on data that can be stored using character set.

Large_text LONG;
Large_raw LONG RAW;
Syntax Explanation:

The above declaration statement declares the variable 'Large_text' of LONG data type and 'Large_raw' of LONG RAW data type.

Note: Using LONG data type is not recommended by Oracle. Instead, LOB data type should be preferred.

NUMBER Data Type:

This data type stores fixed or floating point numbers up to 38 digits of precision. This data type is used to work with fields which will contain only number data. The variable can be declared either with precision and decimal digit details or without this information. Values need not enclose within quotes while assigning for this data type.
A NUMBER(8,2);
B NUMBER(8);
C NUMBER;

Syntax Explanation:

In the above, the first declaration declares the variable 'A' is of number data type with total precision 8 and decimal digits 2.

The second declaration declares the variable 'B' is of number data type with total precision 8 and no decimal digits.

The third declaration is the most generic, declares variable 'C' is of number data type with no restriction in precision or decimal places. It can take up to a maximum of 38 digits.

BOOLEAN Data Type:

This data type stores the logical values. It represents either TRUE or FALSE and mainly used in conditional statements. Values need not enclose within quotes while assigning for this data type.

Var1 BOOLEAN;
Syntax Explanation:

In the above, variable 'Var1' is declared as BOOLEAN data type. The output of the code will be either true or false based on the condition set.

DATE Data Type:

This data type stores the values in date format, as date, month, and year. Whenever a variable is defined with DATE data type along with the date it can hold time information and by default time information is set to 12:00:00 if not specified. Values need to enclose within quotes while assigning for this data type.

The standard Oracle time format for input and output is 'DD-MON-YY' and it is again set at NLS_PARAMETERS (NLS_DATE_FORMAT) at the session level.

```
newyear DATE:='01-JAN-2015';
current_date DATE:=SYSDATE;
```

Syntax Explanation:

In the above, variable 'newyear' is declared as DATE data type and assigned the value of Jan 1st, 2015 date.

The second declaration declares the variable current_date as DATE data type and assigned the value with current system date.
Both these variable holds the time information.

LOB Data Type:

This data type is mainly used to store and manipulate large blocks of unstructured data's like images, multimedia files, etc. Oracle prefers LOB instead of the a LONG data type as it is more flexible than the LONG data type. The below are the few main advantage of LOB over LONG data type.
The number of column in a table with LONG data type is limited to 1, whereas a table has no restriction on a number of columns with LOB data type.
The data interface tool accepts LOB data type of the table during data replication, but it omits LONG column of the table. These LONG columns need to be replicated manually.

The size of the LONG column is 2GB, whereas LOB can store up to 128 TB.
Oracle is constantly improvisingthe LOB data type in each of their releases according to the modern requirement, whereas LONG data type is constant and not getting many updates.
So, it is always good to use LOB data type instead of the LONG data type. Following are the different LOB data types. They can store up to the size of 128 terabytes.

Declarations

You can declare variables and constants in the declarative part of any PL/SQL block, subprogram, or package. Declarations allocate storage space for a value, specify its data type, and name the storage location so that you can reference it.

Some examples follow:

```
DECLARE
birthday DATE;
emp_count SMALLINT := 0;
```

Constants – To declare a constant, put the keyword CONSTANT before the type specifier.

Example –

```
DECLARE
credit_limit CONSTANT REAL := 5000.00;
```

Using DEFAULT
You can use the keyword DEFAULT instead of the assignment operator to initialize variables. For example, the declaration
blood_type CHAR := 'O';
can be rewritten as follows:
blood_type CHAR DEFAULT 'O';
Using NOT NULL –
Besides assigning an initial value, declarations can impose the NOT NULL constraint:

DECLARE
acct_id INTEGER(4) NOT NULL := 9999;
Using the %TYPE Attribute
The %TYPE attribute provides the data type of a variable or database column. This is particularly useful when declaring variables that will hold database values. For example, assume there is a column named last_name in a table named employees.
To declare a variable named v_last_name that has the same data type as column title, use dot notation and the %TYPE attribute, as follows:
v_last_nameemployees.last_name%TYPE;
Using the %ROWTYPE Attribute
The %ROWTYPE attribute provides a record type that represents a row in a table or view. Columns in a row and corresponding fields in a record have the same names and data types. However, fields in a %ROWTYPE record do not inherit constraints, such as the NOT NULL or check constraint, or default values.

DECLARE
dept_recdepartments%ROWTYPE; -- declare record variable
Restrictions on Declarations

PL/SQL does not allow forward references. You must declare a variable or constant before referencing it in other statements, including other declarative statements.

DECLARE
-- Multiple declarations not allowed.
-- i, j, k, l SMALLINT;
-- Instead, declare each separately.
i SMALLINT;
j SMALLINT;
-- To save space, you can declare more than one on a line.
k SMALLINT; l SMALLINT;
RAW,NCHAR,NVARCHAR2,RAW,ROWID,STRING,VARCHAR,VARCHAR2, Composite TypesTABLE, VARRAY, RECORDLOB TypesBFILE, BLOB, CLOB, NCLOBReferenceTypesREF CURSORBOOLEAN, DATEDBMS_OUTPUT.PUT_LINE:It is a pre-defined package that prints the message inside the parenthesis

ANONYMOUS PL/SQL BLOCK.

The text of an Oracle Forms trigger is an anonymous PL/SQL block. It consists of three sections:

- A declaration of variables, constants, cursors and exceptions which is optional.
- A section of executable statements.
- A section of exception handlers, which is optional.

ATTRIBUTES

Allow us to refer to data types and objects from the database.PL/SQL variables and Constants can haveattributes. The main advantage of using Attributes is even if you Change the data definition, you don’tneed to change in the application.

%TYPE

It is used when declaring variables that refer to the database columns Using %TYPE to declare variable has two advantages. First, you need not know the exact data type of variable. Second, if the database definition of variable changes, the data t ype of variable changesaccordingly at run time.

%ROWTYPE

The %ROWTYPE attribute provides a record type that represents a row in a table (or view). The recordcan store an entire row of data selected from the table or fetched from a cursor or strongly typedcursor variable.

Control Structures in PL/SQL

Procedural computer programs use the basic control structures.

Control structures

The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is true or false. A condition is any variable or expression that returns a BOOLEAN value (TRUE or FALSE).The iteration structure executes a sequence of statements repeatedly as long as a condition holds true.The sequence structure simply executes a sequence of statements in the order in which they occur.

Testing Conditions: IF and CASE Statements

The IF statement executes a sequence of statements depending on the value of a condition. There are three forms of IF statements: IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSIF.

The CASE statement is a compact way to evaluate a single condition and choose between many alternative actions. It makes sense to use CASE when there are three or more alternatives to choose from.

Using the IF-THEN Statement

The simplest form of IF statement associates a condition with a sequence ofstatements enclosed by the keywords THEN and END IF (not ENDIF)

The sequence of statements is executed only if the condition is TRUE. If the condition is FALSE or NULL, the IF statement does nothing. In either case, control passes to the next statement.

Example: Using a Simple IF-THEN Statement

```
DECLARE
sales NUMBER(8,2) := 10100;
quota NUMBER(8,2) := 10000;
bonus NUMBER(6,2);
emp_id NUMBER(6) := 120;

BEGIN
IF sales > (quota + 200) THEN
bonus := (sales - quota)/4;
UPDATE employees SET salary = salary + bonus WHERE employee_id = emp_id;
END IF;
END;
```

35

```
/
```

Using CASE Statements

Like the IF statement, the CASE statement selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is an expression whose value is used to select one of several alternatives.

Example: Using the CASE-WHEN Statement

```
DECLARE
grade CHAR(1);
BEGIN
grade := 'B';
CASE grade
WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
END CASE;
END;
/
```

Controlling Loop Iterations: LOOP and EXIT Statements

LOOP statements execute a sequence of statements multiple times. There are three forms of LOOP statements: LOOP, WHILE-LOOP, and FOR-LOOP.

36

Using the LOOP Statement

The simplest form of LOOP statement is the basic loop, which encloses a sequence of statements between the keywords LOOP and END LOOP, as follows:

```
LOOP
sequence_of_statements
END LOOP;
```

With each iteration of the loop, the sequence of statements is executed, then control resumes at the top of the loop. You use an EXIT statement to stop looping and prevent an infinite loop. You can place one or more EXIT statements anywhere inside a loop, but not outside a loop. There are two forms of EXIT statements: EXIT and
EXIT-WHEN.

Using the EXIT Statement

The EXIT statement forces a loop to complete unconditionally. When an EXIT statement is encountered, the loop completes immediately and control passes to the next statement.

Using the EXIT-WHEN Statement

The EXIT-WHEN statement lets a loop complete conditionally. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop completes and control passes to the next statement after the loop.

Labeling a PL/SQL Loop

Like PL/SQL blocks, loops can be labeled. The optional label, an undeclared identifier enclosed by double angle brackets, must appear at the beginning of the LOOP statement. The label name can also appear at the end of the LOOP statement. When you nest labeled loops, use ending label names to improve readability.

Using the WHILE-LOOP Statement

The WHILE-LOOP statement executes the statements in the loop body as long as a condition is true:

```
WHILE condition LOOP
sequence_of_statements

END LOOP;
```

Using the FOR-LOOP Statement

Simple FOR loops iterate over a specified range of integers. The number of iterations is known before the loop is entered. A double dot (..) serves as the range operator. The range is evaluated when the FOR loop is first entered and is never re-evaluated. If the lower bound equals the higher bound, the loop body is executed once.

Example: Using a Simple FOR..LOOP Statement

```
DECLARE
p NUMBER := 0;
BEGIN
FOR k IN 1..500 LOOP -- calculate pi with 500 terms
p := p + ( ( (-1) ** (k + 1) ) / ((2 * k) - 1) );
END LOOP;
p := 4 * p;
DBMS_OUTPUT.PUT_LINE( 'pi is approximately : ' || p ); -- print result
```

```
END;
/
```

Sequential Control: GOTO and NULL Statements

The GOTO statement is seldom needed. Occasionally, it can simplify logic enough to warrant its use. The NULL statement can improve readability by making the meaning and action of conditional statements clear.

Overuse of GOTO statements can result in code that is hard to understand and maintain. Use GOTO statements sparingly. For example, to branch from a deeply nested structure to an error-handling routine, raise an exception rather than use a GOTO statement.

Using the GOTO Statement

The GOTO statement branches to a label unconditionally. The label must be unique within its scope and must precede an executable statement or a PL/SQL block. When executed, the GOTO statement transfers control to the labeled statement or block. The labeled statement or block can be down or up in the sequence of statements.

Example : Using a Simple GOTO Statement

```
DECLARE
p VARCHAR2(30);
n PLS_INTEGER := 37; -- test any integer > 2 for prime
BEGIN
FOR j in 2..ROUND(SQRT(n)) LOOP
IF n MOD j = 0 THEN -- test for prime
p := ' is not a prime number'; -- not a prime number
GOTO print_now;
END IF;
END LOOP;
p := ' is a prime number';
<<print_now>>
DBMS_OUTPUT.PUT_LINE(TO_CHAR(n) || p);
END;
/
```

Using the NULL Statement

The NULL statement does nothing, and passes control to the next statement. Some languages refer to such an instruction as a no-op (no operation).

Example: Using the NULL Statement to Show No Action

```
DECLARE
v_job_idVARCHAR2(10);
v_emp_id NUMBER(6) := 110;
BEGIN
SELECT job_id INTO v_job_id FROM employees WHERE employee_id = v_emp_id;
IF v_job_id = 'SA_REP' THEN
UPDATE employees SET commission_pct = commission_pct * 1.2;
```

```
ELSE
NULL; -- do nothing if not a sales representative
END IF;
END;
/
```

Oracle PL/SQL Insert, Update, Delete & Select Into [Example]

we are going to learn how to use SQL in PL/SQL. SQL is the actual component that takes care of fetching and updating of data in the database whereas PL/SQL is the component that processes these data. Further, in this article, we will also discuss how to combi ne the SQL within the PL/SQL block.

- Data Insertion
- Data Update
- Data Deletion
- Data Selection
- DML Transactions in PL/SQL

DML stands for Data Manipulation Language. These statements are mainly used to perform the manipulation activity. It deals with the below operations.

- Data Insertion
- Data Update
- Data Deletion
- Data Selection

In PL/SQL, we can do the data manipulation only by using the SQL commands.

Data Insertion

In PL/SQL, we can insert the data into any table using the SQL command INSERT INTO. This command will take the table name, table column and column values as the input and insert the value in the base table.

The INSERT command can also take the values directly from another table using 'SELECT' statement rather than giving the values for each column. Through 'SELECT' statement, we can insert as many rows as the base table contains.

Syntax:

```
BEGIN
INSERT INTO <table_name>(<column1 >,<column2>,...<column_n>)
VALUES(<valuel><value2>,...:<value_n>);
END;
```

The above syntax shows the INSERT INTO command. The table name and values are a mandatory fields, whereas column names are not mandatory if the insert statements have values for all the column of the table.

The keyword 'VALUES' is mandatory if the values are given separately as shown above.

Syntax:

```
BEGIN
INSERT INTO <table_name>(<columnl>,<column2>,...,<column_n>)
SELECT <columnl>,<column2>,.. <column_n> FROM <table_name2>;
END;
```

The above syntax shows the INSERT INTO command that takes the values directly from the <table_name2> using the SELECT command.
The keyword 'VALUES' should not be present in this case as the values are not given separately.

Data Update
Data update simply means an update of the value of any column in the table. This can be done using 'UPDATE' statement. This statement takes the table name, column name and value as the input and updates the data.
Syntax:

```
BEGIN
UPDATE <table_name>
SET <columnl>=<VALUE1>,<column2>=<Yalue2>,<column_n>=<value_n>
WHERE <condition that uniquely identifies the record that needs to be update>;
END;
```

The above syntax shows the UPDATE. The keyword 'SET' instruct that PL/SQL engine to update the value of the column with the value given.

'WHERE' clause is optional. If this clause is not given, then the value of the mentioned column in the entire table will be updated.

Data Deletion
Data deletion means to delete one full record from the database table. The 'DELETE' command is used for this purpose.

Syntax:

```
BEGIN
DELETE
FROM
<table_name>
WHERE <condition that uniquely identifies the record that needs to be update>;
END;
```

The above syntax shows the DELETE command. The keyword 'FROM' is optional and with or without 'FROM' clause the command behaves in the same way.'WHERE' clause is optional. If this clause is not given, then the entire table will be deleted.

Data Selection
Data projection/fetching means to retrieve the required data from the database table. This can be achieved by using the command 'SELECT' with 'INTO' clause. The 'SELECT' command

will fetch the values from the database, and 'INTO' clause will assign these values to the local variable of the PL/SQL block.

Below are the points that need to be considered in 'SELECT' statement.

'SELECT' statement should return only one record while using 'INTO' clause as one variable can hold only one value. If the 'SELECT' statement returns more than one value than 'TOO_MANY_ROWS' exception will be raised.

'SELECT' statement will assign the value to the variable in the 'INTO' clause, so it needs to get at least one record from the table to populate the value. If it didn't get any record, then the exception 'NO_DATA_FOUND' is raised.

The number of columns and their data type in 'SELECT' clause should match with the number of variables and their data types in the 'INTO' clause.
The values are fetched and populated in the same order as mentioned in the statement.
'WHERE' clause is optional that allows to having more restriction on the records that are going to be fetched.

'SELECT' statement can be used in the 'WHERE' condition of other DML statements to define the values of the conditions.

The 'SELECT' statement when using 'INSERT', 'UPDATE', 'DELETE' statements should not have 'INTO' clause as it will not populate any variable in these cases.

Syntax:
```
BEGIN
SELECT <columnl>,..<column_n> INTO <vanable 1 >,..<variable_n>
FROM <table_name>
WHERE <condition to fetch the required records>;
END;
```

The above syntax shows the SELECT-INTO command. The keyword 'FROM' is mandatory that identifies the table name from which the data needs to be fetched.

'WHERE' clause is optional. If this clause is not given, then the data from the entire table will be fetched.
Example 1: In this example, we are going to see how to perform DML operations in PL/SQL. We are going to insert the below four records into emp table.

EMP_NAME	EMP_NO	SALARY	MANAGER
BBB	1000	25000	AAA
XXX	1001	10000	BBB
YYY	1002	10000	BBB
ZZZ	1003	7500	BBB

Autonomous Transaction in Oracle PL/SQL: Commit, Rollback

What are TCL Statements in PL/SQL?

TCL stands for Transaction Control Statements. It will either save the pending transactions or roll back the pending transaction. These statements play the vital role because unless the transaction is saved the changes through DML statements will not be saved in the database. Below are the different TCL statements.

COMMIT	Saves all the pending transaction
ROLLBACK	Discard all the pending transaction
SAVEPOINT	Creates a point in the transaction till which rollback can be done later
ROLLBACK TO	Discard all the pending transaction till the specified <save point>

The transaction will be complete under the following scenarios.

- When any of the above statements is issued (except SAVEPOINT)
- When DDL statements are issued. (DDL are auto-commit statements)
- WHEN DCL statements are issued. (DCL are auto-commit statements)
- What is Autonomous Transaction

In PL/SQL, all the modifications done on data will be termed as a transaction. A transaction is considered as complete when the save/discard is applied to it. If no save/discard is given, then the transaction will not be considered as complete and the modifications done on the data will not be made permanent on the server.

Irrespective of some modifications done during a session, PL/SQL will treat the whole modification as a single transaction and saving/discard this transaction affects to the entire pending changes in that session. Autonomous Transaction provides functionality to the developer in which it allows to do changes in a separate transaction and to save/discard that particular transaction without affecting the main session transaction.

This autonomous transaction can be specified at subprogram level.To make any subprogram to work in a different transaction, the keyword 'PRAGMA AUTONOMOUS_TRANSATION' should be given in the declarative section of that block.

It will instruct that compiler to treat this as the separate transaction and saving/discarding inside this block will not reflect in the main transaction.

Issuing COMMIT or ROLLBACK is mandatory before going out of this autonomous transaction to the main transaction because at any time only one transaction can be active.

So once we made an autonomous transaction we need to save it and complete the transaction then only we can move back to the main transaction.

Syntax:

```
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
<executin_part>
[COMMIT|ROLLBACK]
END;
/
```

In the above syntax, the block has been made as an autonomous transaction.
Example 1: In this example, we are going to understand how the autonomous transaction is working.
SQL in PL/SQL

```
DECLARE
L_salary NUMBER;
  IS
  PRAGMA AUTONOMOUS_TRANSACTION;
  BEGIN
  UPDATE emp
  SET salary=salary+15000
  WHERE emp_no=1002;
  COMMIT;
 E ND;

BEGIN
SELECT salary INTO l_salary FROM emp WHERE emp_no=1001;
Dbms_output.put_line(‘Before: Salary of 1001 is‘||l_salary);
SELECT salary INTO l_salary FROM emp WHERE emp_no=1002;
Dbmsoutput.put_line(‘Before: Salary of 1002 is‘||1_salary):
UPDATE emp
SET salary=salary+5000
WHERE emp_no=1001;
Nested_block;
ROLLBACK;
SELECT salary INTO l_salary FROM emp WHERE emp_no=1001;
Dbms_output.put_line(‘After: Salary of 1001 is‘||l_salary);
SELECT salary INTO l_salary FROM emp WHERE emp_no=1002;
Dbms_output.put_line(‘After: Salary of 1002 is‘||l_salary);
END:
/
```

Output

```
Before:Salary of 1001 is 15000
Before:Salary of 1002 is 10000
After:Salary of 1001 is 15000
```

```
After:Salary of 1002 is 25000
```

Code Explanation:
Code line 2: Declaring l_salary as NUMBER.
Code line 3: Declaring nested_block procedure
Code line 4: Making nested_block procedure as 'AUTONOMOUS_TRANSACTION'.
Code line 7-9: Increasing the salary for employee number 1002 by 15000.
Code line 10: Committing the transaction.
Code line 13-16: Printing the salary details of employee 1001 and 1002 before changes.
Code line 17-19: Increasing the salary for employee number 1001 by 5000.
Code line 20: Calling the nested_block procedure;
Code line 21: Discarding the main transaction.
Code line 22-25: Printing the salary details of employee 1001 and 1002 after changes.

The salary increase for employee number 1001 is not reflected because the main transaction has been discarded. The salary increase for employee number 1002 is reflected because that block has been made as a separate transaction and saved at the end.

So irrespective of the save/discard at main transaction the changes at autonomous transaction has been saved without affecting the main transaction changes.

UNIT – V
EXCEPTION

What is Exception Handling in PL/SQL?

An exception occurs when the PL/SQL engine encounters an instruction which it cannot execute due to an error that occurs at run-time. These errors will not be captured at the time of compilation and hence these needed to handle only at the run-time.

For example, if PL/SQL engine receives an instruction to divide any number by '0', then the PL/SQL engine will throw it as an exception. The exception is only raised at the run-time by the PL/SQL engine.
Exceptions will stop the program from executing further, so to avoid such condition, they need to be captured and handled separately. This process is called as Exception-Handling, in which the programmer handles the exception that can occur at the run time.

you will learn the following topics-
Exception-Handling Syntax
Types of Exception
Predefined Exceptions
User-defined Exception
PL/SQL Raise Exception
Important points to note in Exception

Exception-Handling Syntax

Exceptions are handled at the block, level, i.e., once if any exception occurs in any block then the control will come out of execution part of that block. The exception will then be handled at the exception handling part of that block. After handling the exception, it is not possible to resend control back to the execution section of that block.

The below syntax explains how to catch and handle the exception.
Exception Handling in PL/SQL

```
BEGIN
<execution block>
.
.
EXCEPTION
WHEN <exceptionl_name>
THEN
<Exception handling code for the "exception 1 _name'' >
WHEN OTHERS
THEN
<Default exception handling code for all exceptions >
END;
```

Syntax Explanation:

In the above syntax, the exception-handling block contains series of WHEN condition to handle the exception.

Each WHEN condition is followed by the exception name which is expected to be raised at the run time.When any exception is raised at runtime, then the PL/SQL engine will look in the exception handling part for that particular exception. It will start from the first 'WHEN' clause and, sequentially it will search.

If it found the exception handling for the exception which has been raised, then it will execute that particular handling code part.

If none of the 'WHEN' clause is present for the exception which has been raised, then PL/SQL engine will execute the 'WHEN OTHERS' part (if present). This is common for all the exception.

After executing the exception, part control will go out of the current block.
Only one exception part can be executed for a block at run-time. After executing it, the controller will skip the remaining exception handling part and will go out of the current block.

Note: WHEN OTHERS should always be at the last position of the sequence. The exception handling part present after WHEN OTHERS will never get executed as the control will exit from the block after executing the WHEN OTHERS.

Types of Exception
There are two types of Exceptions in Pl/SQL.

- Predefined Exceptions
- User-defined Exception

Predefined Exceptions

Oracle has predefined some common exception. These exceptions have a unique exception name and error number. These exceptions are already defined in the 'STANDARD' package in Oracle. In code, we can directly use these predefined exception name to handle them.

Below are the few predefined exceptions

Exception	Error Code	Exception Reason
ACCESS_INTO_NULL	ORA-06530	Assign a value to the attributes of uninitialized objects
CASE_NOT_FOUND	ORA-06592	None of the 'WHEN' clause in CASE statement satisfied and no 'ELSE' clause is specified
COLLECTION_IS_NULL	ORA-06531	Using collection methods (except EXISTS) or accessing collection attributes on a uninitialized collections
CURSOR_ALREADY_OPEN	ORA-06511	Trying to open a cursor which is already opened
DUP_VAL_ON_INDEX	ORA-00001	Storing a duplicate value in a database column that is a constrained by unique index
INVALID_CURSOR	ORA-01001	Illegal cursor operations like closing an unopened cursor

INVALID_NUMBER	ORA-01722	Conversion of character to a number failed due to invalid number character
NO_DATA_FOUND	ORA-01403	When 'SELECT' statement that contains INTO clause fetches no rows.
ROW_MISMATCH	ORA-06504	When cursor variable data type is incompatible with the actual cursor return type
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Referring collection by an index number that is larger than the collection size
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Referring collection by an index number that is outside the legal range (eg: -1)
TOO_MANY_ROWS	ORA-01422	When a 'SELECT' statement with INTO clause returns more than one row
VALUE_ERROR	ORA-06502	Arithmetic or size constraint error (eg: assigning a value to a variable that is larger than the variable size)
ZERO_DIVIDE	ORA-01476	Dividing a number by '0'

User-defined Exception

In Oracle, other than the above-predefined exceptions, the programmer can create their own exception and handle them. They can be created at a subprogram level in the declaration part. These exceptions are visible only inthat subprogram. The exception that is defined in the package specification is public exception, and it is visible wherever the package is accessible. <

Syntax: At subprogram level

```
DECLARE
<exception_name> EXCEPTION;
BEGIN
<Execution block>
EXCEPTION
WHEN <exception_name> THEN
<Handler>
END;
```

In the above syntax, the variable'exception_name' is defined as 'EXCEPTION' type.
This can be used as in a similar way as a predefined exception.
Syntax:At Package Specification level

```
CREATE PACKAGE <package_name>
 IS
<exception_name> EXCEPTION;
.
.
END <package_name>;
```

In the above syntax, the variable 'exception_name' is defined as 'EXCEPTION' type in the package specification of <package_name>.
This can be used in the database wherever package 'package_name' can be called.

```
AS
BEGIN
<Execution block>
EXCEPTION
WHEN <exception_name> THEN
<Handler>
RAISE;
END;
```

Syntax Explanation:

In the above syntax, the keyword RAISE is used in the exception handling block. Whenever program encounters exception "exception_name", the exception is handled and will be completed normally

But the keyword 'RAISE' in the exception handling part will propagate this particular exception to the parent program.

Note: While raising the exception to the parent block the exception that is getting raised should also be visible at parent block, else oracle will throw an error.

We can use keyword 'RAISE' followed by the exception name to raise that particular user-defined/predefined exception. This can be used in both execution part and in exception handling part to raise the exception.

Exception Handling in PL/SQL

```
CREATE [ PROCEDURE | FUNCTION ]
AS
BEGIN
<Execution block>
RAISE <exception_name>
EXCEPTION
WHEN <exception_name> THEN
<Handler>
END;
```

Syntax Explanation:

In the above syntax, the keyword RAISE is used in the execution part followed by exception "exception_name".

This will raise this particular exception at the time of execution, and this needs to be handled or raised further.

Example 1: In this example, we are going to see

How to declare the exception

How to raise the declared exception and

How to propagate it to the main block

Exception Handling in PL/SQL
Exception Handling in PL/SQL

```
DECLARE
Sample_exception EXCEPTION;
PROCEDURE nested_block
IS
BEGIN
Dbms_output.put_line(‘Inside nested block’);
Dbms_output.put_line(‘Raising sample_exception from nested block’);
RAISE sample_exception;
EXCEPTION
WHEN sample_exception THEN
Dbms_output.put_line (‘Exception captured in nested block. Raising to main block’);
RAISE,
END;

BEGIN
Dbms_output.put_line(‘Inside main block’);
Dbms_output.put_line(‘Calling nested block’);
Nested_block;
EXCEPTION
WHEN sample_exception THEN
Dbms_output.put_line (‘Exception captured in main block');
END:
/
```

Code Explanation:
Code line 2: Declaring the variable 'sample_exception' as EXCEPTION type.
Code line 3: Declaring procedur enested_block.
Code line 6: Printing the statement "Inside nested block".
Code line 7: Printing the statement "Raising sample_exception from nested block."
Code line 8: Raising the exception using 'RAISE sample_exception'.
Code line 10: Exception handler for exception sample_exception in the nested block.
Code line 11: Printing the statement 'Exception captured in nested block. Raising to main block'.
Code line 12: Raising the exception to main block (propagating to the main block).
Code line 15: Printing the statement "Inside the main block".
Code line 16: Printing the statement "Calling nested block".
Code line 17: Calling nested_block procedure.
Code line 19: Exception handler for sample_exception in the main block.
Code line 20: Printing the statement "Exception captured in the main block."

Important points to note in Exception

In function, an exception should always either return value or raise the exception further. else Oracle will throw 'Function returned without a value' error at run-time.

Transaction control statements can be given at exception handling block.SQLERRM and SQLCODE are the in-built functions that will give the exception message and code.
If an exception is not handled then by default all the active transaction in that session will be rolled back.RAISE_APPLICATION_ERROR (-<error_code>, <error_message>) can be used instead of RAISE to raise the error with user code and message. Error code should be greater than 20000 and prefixed with '-'.

EXCEPTION
An Exception is raised when an error occurs. In case of an error then normal execution stops and thecontrol is immediately transferred to the exception handling part of the PL/SQL Block.Exceptions are designed for runtime handling, rather than compile time handling. Exceptions improve readability by letting you isolate error-handling routines.When an error occurs, an exception is raised. That is, normal execution stops and control transfers tothe exception-handling part of your PL/SQL block or subprogram. Exception Types

Predefined Exceptions
An internal exception is raised implicitly whenever your PL/SQL program violates an Oracle rule orexceeds a system-dependent limit. Every Oracle error has a number, but exceptions must be handledby name. So, PL/SQL predefines some common Oracle errors as exceptions. For example, PL/SQLraises the predefined exception NO_DATA_FOUND if a SELECT INTO statement returns no rows.

User – Defined exceptions
User – defined exception must be defined and explicitly raised by the user

EXCEPTION_INIT
A named exception can be associated with a particular oracle error. This can be used to trap the errorspecifically.

PRAGMA EXCEPTION_INIT
(exception name, Oracle_error_number);The pragma EXCEPTION_INIT associates an exception name with an Oracle, error number. That allowsyou to refer to any internal exception by name and to write a specific handler

RAISE_APPLICATION_ERROR
The procedure raise_application_error lets you issue user-defined error messages from storedsubprograms. That way, you can report errors to your application and avoid returning unhandledexceptions.

Using SQLCODE and SQLERRM
For internal exceptions, SQLCODE returns the number of the Oracle error. The number that SQLCODEreturns is negative unless the Oracle error is no data found, in which case SQLCODE returns +100.SQLERRM returns the corresponding error message. The message begins with the Oracle error code.

Unhandled Exceptions
PL/SQL returns an unhandled exception error to the host environment, which determines the outcome.
When Others
It is used when all exception are to be trapped.

CURSORS

Oracle allocates an area of memory known as context area for the processing of SQL statements. Thepointer that points to the context area is a cursor.

Merits
1] Allowing to position at specific rows of the result set.2] Returning one row or block of rows from the current position in the result set.3] Supporting data modification to the rows at the current position in the result set.

TYPES
1] STATIC CURSOR
SQL statement is determined at design time.
A] EXPLICIT CURSOR

Multiple row SELECT statement is called as an explicit cursor.To execute a multi-row query, Oracle opens an unnamed work area that stores processinginformation. To access the information, you can use an explicit cursor, which names the workarea.Usage - If the SELECT statement returns more that one row then explicit cursor should beused.Steps

- Declare a cursor
- Open a cursor
- Fetch data from the cursor
- Close the cursorEXPLICIT CURSOR ATTRIBUTES
- %FOUND (Returns true if the cursor has a value)
- %NOTFOUND (Returns true if the cursor does not contain any value)
- %ROWCOUNT (Returns the number of rows selected by the cursor)
- %ISOPEN (Returns the cursor is opened or not)

CURSOR FOR LOOP

The CURSOR FOR LOOP lets you implicitly OPEN a cursor, FETCH each row returned by the queryassociated with the cursor and CLOSE the cursor when all rows have been processed.

SYNTAX

FOR <RECORD NAME> IN <CURSOR NAME> LOOPSTATEMENTSEND LOOP;To refer an element of the record use <record name. Column name>

Parameterized Cursor

A cursor can take parameters, which can appear in the associated query wherever constants canappear. The formal parameters of a cursor must be IN parameters. Therefore, they cannot returnvalues to actual parameters. Also, you cannot impose the constraint NOT NULL on a cursor parameter.The values of cursor parameters are used by the associated query when the cursor is opened.

B .IMPLICIT CURSOR

An IMPLICIT cursor is associated with any SQL DML statement that does not have aexplicitcursor associated with it.
This includes:

- All INSERT statements
- All UPDATE statements
- All DELETE statements
- All SELECT ..

INTO statementsIMPLICIT CURSOR ATTRIBUTES"SQL%FOUND (Returns true if the DML operation is valid)"SQL%NOTFOUND (Returns true if the DML operation is invalid)"SQL%ROWCOUNT (Returns the no. of rows affected by the DML operation)

2] DYNAMIC CURSOR

Dynamic Cursor can be used along with DBMS_SQL package .A SQL statement is dynamic, if it isconstructed at run time and then executed.

3] REF CURSOR

Declaring a cursor variable creates a pointer, not an item. In PL/SQL, a pointer has data type REF X,where REF is short for REFERENCE and X stands for a class of objects. Therefore, a cursor variable hasdata type REF CURSOR.To execute a multi-row query, Oracle opens an unnamed work area that stores processing information.To access the information, you can use an explicit cursor, which names the work area.

ORACLE 9IAdvanced Explicit Cursor Concepts
1. FOR UPDATE WAIT Clause
Lock the rows before the update or delete.
Use explicit locking to deny access for the duration of a transaction.SyntaxCursor eipc1 is select * from emp for update [ofcolumn_reference] [NOWAIT]Column ReferenceIt is column in the table against which the query is performed[A list of column may also be used]NOWAITReturns an Oracle Error if the rows are locked by another session.• The SELECT... FOR UPDATE statement has been modified to allow the user to specify howlong the command should wait if the rows being selected are locked.• If NOWAIT is specified, then an error is returned immediately if the lock cannot be obtained.

Example of Using FOR UPDATE WAIT Clause
1. SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID = 10 FOR UPDATE WAIT 20
2. DECLARECURSOR EMP_CURSOR ISSELECT EMPNO, ENAME, DNAME FROM EMP,DEPT

WHEREEMP.DEPTNO=DEPT.DEPTNO AND EMP.DEPTNONO=80FOR UPDATE OF SALARY NOWAIT;[Retrieve the Employees who work in department 80 and update their Salary

2. The WHERE CURRENT OF Clause
Use cursors to update or delete the cursor row.
Include the FOR UPDATE clause in the cursor query to lock the row first.
Use the WHERE CURRENT OF clause to refer the current row from an explicit cursor.SyntaxWHERE CURRENT OF <cursor_name>Cursor_Name -It is the name of a declared cursor. [The cursor have beendeclared with the FOR UPDATE clause]

Example of Using FOR WHERE CURRENT OF Clause
1. DECLARECURSOR EMP_CURSOR IS SELECT EMPNO, ENAME, DNAME FROM E.EMP,D.DEPT WHEREEMP.DEPTNO=DEPT.DEPTNO AND EMP.DEPTNONO=80 FOR UPDATE OF SALARY NOWAIT;BEGINFOR I IN EMP_CURSOR;LOOPIF I.SAL<5000 THENUPDATE EMP SET SALARY=I.SAL*1.10 WHERE CURRENT OF EMP_CURSOR;END IF;END LOOP;END;

The Example loops through each employee in department 80, and checks whether the salary isless than 5000.If salary is less than , the salary is raised by 10%. The where current of clause in theUPDATE statement refers to the currently fetched records.]

3. CURSORS WITH SUB QUERIES
Sub queries are often used in the WHERE clause of select statement. It can be used toFROM clause, creating a temporary data source for the query.
DECLAREbonusREAL;BEGINFORemp_rec IN (SELECT empno, sal, comm FROM emp)LOOPbonus := (emp_rec.sal * 0.05) + (emp_rec.comm * 0.25);INSERT INTO bonuses VALUES (emp_rec.empno, bonus);END LOOP;COMMIT;END;

Modify routines online without interfering with other users
Modify one routine to affect multiple applications
Improved data security and integrity

PL/SQL COMPOSITE DATA TYPES

PL/SQL data types has been broadly classified in two category - Scalar data types and Composite data types. In previous post we discussed about scalar data type which cover NUMBER , CHAR, VARCHAR2, LONG, etc types.

To discuss one of the composite types in PL/SQL - collections and in next post another composite type - Record will be discussed.

A composite data type stores values that have internal components and internal components can be either scalar or composite.Internal components can be of same data type and different data type. PL/SQL allows us to define two kinds of composite data types :

Collection

The internal components must have the same data type and we can access each element of a collection variable by its unique index, with this syntax: variable_name(index).

Record

The internal components can have different data types and we can access each field of a record variable by its name, with this syntax: variable_name.field_name. Detailed discussion of Records in PL/SQL.

Collections in PL/SQL
Oracle provides three types of collections.
Index-by Table(associate array),
Nested Tables, and
VARRAY.

VARRAY:

It is variable-size array and element counts in it can vary from 0 to declared maximum size.Characteristics of VARRAY:

Elements of VARRAY can be accessed by variable_name(index).VARRY index starts from 1 (lowest_index = 1) and it can go up to maximum size of VARRAY.

As contrast to associative array, it can be persisted in database table and order of elements (indexes and element order) remain stable.

VARRAY has constructor support as contrast to Associative array that does not support collection constructor. A collection constructor is a system-defined function with the same name as a collection type,which returns a collection of that type. Syntax of a constructor invocation is:Collection_type ([values,...]), values are optional. If no value is passed constructor returns emplty collection.

VARRAY is stored as a single object in a column in database table.(if size of object is more than 4KB then it is stored separately but in same namespace). Following diagram depicts how VARRAY is stored in database table: Highlighted column refers to VARRAY type and stored in database column as other scalar type.

VARRAY creation and its initialization:-
Syntax of VARRAY creation is as follows - varray_type_def with collection

```
-- size_limit: upper limit of VARRAY(maximum that many elements can be stored)
TYPE typeIS  { VARRAY | [ VARYING ] ARRAY } ( size_limit )
OF data type[ NOT NULL ]
```

Consider following sample program which creates a VARRY to store address information of employees and initialize it with constructor. Here ADDRESS is VARRAY type with upper limit of container 3 and using constructor collection of type ADDRESS created is returned to emp_address.

```
DECLARE
  -- VARRAY type declaration of type VARCHAR, upperlimit 3
  TYPE ADDRESS IS VARRAY(3) OF VARCHAR2(45);
  -- varray variable initialized with constructor of type ADDRESS
emp_address ADDRESS := ADDRESS('HYD,IND', 'NY,USA','BANG,IND');
BEGIN
   DBMS_OUTPUT.PUT_LINE('VARRAY elements count is '
       || emp_address.COUNT);
    DBMS_OUTPUT.PUT_LINE('Address display - Iteration over VARRAY');
    --emp_address.FIRST= 1 and emp_address.LAST = 3
    FOR i IN emp_address.FIRST..emp_address.LAST LOOP
      DBMS_OUTPUT.PUT_LINE(i || '. address is ' || emp_address(i));
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Modify emp_address VARRAY ');
emp_address(1) := 'Sydeny, AUS';
    DBMS_OUTPUT.PUT_LINE('Accessing VARRAY based on index,modified addr ess is  '
       ||emp_address(1)); -- notice modified value here.
    --emp_address.DELETE(2);--Delete operation on VARRAY is not allowed.

END;
```

```
==============Sample output==================
VARRAY elements count is 3
Address display - Iteration over VARRAY
1. address is HYD,IND
2. address is NY,USA
3. address is BANG,IND
Modify emp_address VARRAY
Accessing VARRAY based on index,modified address is Sydeny, AUS
==========================================
```

Where do we use VARRAY:- If we have prior info of maximum number of elements and we want sequential access of collection. It is not not good idea to use VARRAY when collection size is very large, because VARRAY is retrieved at once from database.

Nested Tables:

It is a table (with rows and columns) that is stored in database table as data of a column in no particular order.When that table is retrieved form database in PL/SQl context, PL/SQL indexes all rows starting from 1 and based on index we can access each row of nested table using method: nested_table_var(index). Following diagram shows how Nested tables is stored in database table. Highlighted inner table in CUSTOMER_DETAILS column refers to Nested table type and stored as part of column data.

Nested table creation and its initialization:- Syntax of Nested table creation is as follows, (nested_table_type_def with collection) :

TYPE type IS {TABLE OF data type[NOT NULL] }

Consider following scenario to understand how Nested table type is created in and stored in database.Lets say we have an Customer_detail_objectisa Object TYPE and it stores customer details and nested table is collection of that object- each row of nested table is customer_detail_object.

For the time being just assume it is a container which can store different data types. Follow following steps and execute query in sequence :

Step 1: Create Object type having fields CustID, cust_name, cust_address, execute below query to create Object named Customer_detail_object.

```
--create ObjectCustomer_detail_object : Created in schema level.
create type Customer_detail_object as object
(
CustIDNUMBER(14),
cust_namevarchar2(25),
cust_addressvarchar2(100)
);
```

Step 2: Now nested table type CUSTOMER_DETAILS of object type Customer_detail_object.888888

```
--Create TABLE of object Customer_detail_object: Created in schema level
create type CUSTOMER_DETAILS as Table of Customer_detail_object;
```

Step 3: Create a table, PRODUCTS_CUSTOMRS_DETAILS, in database with a fields of type CUSTOMER_DETAILS (while creating table we specify about CUSTOMER_DETAILS as nested table).

```
--create table in database , NESTED TABLE clause is mandatory to append
create table PRODUCTS_CUSTOMRS_DETAILS
(
product_id number(5),
product_namevarchar2(30),
CUSTOMER_DETAILS HR.CUSTOMER_DETAILS
) NESTED TABLE CUSTOMER_DETAILS STORE AS CUSTOMRS_OBJECTS;
```

Step 4: Insert rows in table. We have created two rows and each row has CUSTOMER_DETAILS table with two rows. If constructor used is empty, nested table will be empty not NULL.

```
 --insert data into table
insert into PRODUCTS_CUSTOMRS_DETAILS
values(1,'P1',
CUSTOMER_DETAILS(
Customer_detail_object(1,'RSQ','BANG,INDIA'),
Customer_detail_object(2,'RTA','AUSTIN,USA')
 ));
insert into PRODUCTS_CUSTOMRS_DETAILS
values(2,'P2',
CUSTOMER_DETAILS(
Customer_detail_object(1,'RSQ','BANG,INDIA'),
Customer_detail_object(2,'BAC','NY,USA')
 ));
commit;
```

Now we have completed set-up to query database and see the stored result from PL/SQL program.

```
declare
customerDetails_Tab CUSTOMER_DETAILS;
begin
--insert a record in database table with nested table data
insert into products_CUSTOMRS_DETAILS
values(3,'P3',
CUSTOMER_DETAILS(
Customer_detail_object(1,'ACV','HYD,INDIA'),
Customer_detail_object(2,'ERT','AUSTIN,USA')
 ));
commit;
 --select record and store nested table value in customerDetails_Tab
select CUSTOMER_DETAILS into customerDetails_Tab
fromproducts_CUSTOMRS_DETAILS
whereproduct_id = 1;
 --update nested table column in database
updateproducts_CUSTOMRS_DETAILS set CUSTOMER_DETAILS = customerDetails_Tab
whereproduct_id = 3;
commit;
end;
```

Here we played around with DML statements and treating inner table as atomic value(Insert, select or update nested table in column).

We can deal with individual row of nested table using TABLE command as follows:

```
select * from table ( select CUSTOMER_DETAILS from
products_CUSTOMRS_DETAILS where product_id = 1);
```

Above query executes and it displays nested tables corresponding to row with product_id = 1, as follows :

Where do we use Nested tables:- Nested table finds it's usage when index values are not consecutive, maximum number of elements storage is not fixed (as contrast to VARRAY).

- Nested table finds extensive use when we want to access refcursor output in SQL and PL/SQL table structure cannot be directly be used SQL. So, a table of SQL object is created at schema level.

```
create or replace type t_emptype as table of emptype;  -- emptype is SQL Object not plsql record
```

How internal storage of VARRAY and Nested table type are different ?VARRY type are stored as part of database table(as column data) until its size reaches 4KB(then it is stored separately from database table), however in Nested table data is stored in a separate store table, a system-generated database table.When we access Nested table database joins this system table with Nested table that's why nested table is suitable for querying and updating(restricted to part of it at a time).

Sample PL/SQL program

PL/SQL Program for Prime Number

```
declare
        n number;
        i number;
        flag number;

begin
        i:=2;
        flag:=1;
        n:=&n;
        fori in 2..n/2
        loop
                if mod(n,i)=0
                then
                        flag:=0;
                        exit;
                end if;
        end loop;
        if flag=1
        then
                dbms_output.put_line('prime');
        else
                dbms_output.put_line('not prime');
        end if;
end;
```

Output

```
Enter value for n: 12
old 9: n:=&n;
new 9: n:=12;
not prime
```

PL/SQL Program to Print Table of a Number

```
declare
        n number;
        i number;
begin
        n:=&n;
        fori in 1..10
        loop
```

```
                dbms_output.put_line(n||' x '||i||' = '||n*i);
        end loop;
end;
```

Output

```
Enter value for n: 5
old 6: n:=&n;
new 6: n:=5;
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Pl/SQL Program for Palindrome Number

A number is called palindrome number if its reverse is equal to itself. For example 12321 is palindrome while 123 is not palindrome.

```
declare
n number;
m number;
rev number:=0;
r number;

begin
n:=12321;
m:=n;
while n>0
loop
r:=mod(n,10);
rev:=(rev*10)+r;
n:=trunc(n/10);
end loop;
if m=rev
then
dbms_output.put_line('number is palindrome');
else
dbms_output.put_line('number is not palindrome');
end if;
end;
```

Output

```
number is palindrome
```