Practical No. 1

Title:

To use the Data Definition Language for creating, altering and dropping the table object in a database.

Topic:

Data definition language.
Proposition1: SQL - Structured Query Language
SQL is a database language used for accessing data in a relational database.

Proposition2: DDL - Data Definition Language
DDL commands are the SQL commands you use to create, alter, and drop (delete)

different database objects in an ORACLE database.

Proposition 3: Table Definition
A Database Object is something created and stored in a database. Tables, views, synonyms indexes, sequences, clusters are all types of database objects.

Table: It is a unit of storage that holds data in the form of rows and columns.

CREATE TABLE

1. ALTER TABLE
2. DROP TABLE
3. TRUNCATE TABLE

Create table command

Syntax: -

```
create table < table_name>
(column_name1 datatype(size),
column_name2 datatype(size),
                :              :
column_name n datatype(size));
```

e.g.

To create a table EMP with the column EMPNO, ENAME, JOB, MGR, JOINDATE, SALARY, COMM, DEPTNO.

```
create table emp (
empno number(5),
ename varchar2(25),
job varchar2(15),
mgr number(5),
joindate date,
salary number(7,2),
comm number(7,2),
deptno number(5));
```

The above SQL statement will create a table EMP with the given columns. To view the structure of the table created use the DESCRIBE COMMAND. The result of the command is to see the column names and data types.

Syntax: -

```
desc[ribe] <table_name>;
```

e.g.

```
desc EMP;
```

Restrictions for creating a table:

1. Table names and column names must begin with a letter.
2. Table names and column names can be 1 to 30 characters long.
3. Table names must contain only the characters A-Z, a-z, 0-9, underscore _, $ and #.
4. Table name should not duplicate the name of another database object.
5. Table name must not be an ORACLE reserved word.
6. Column names should not be duplicate within a table definition.

Alter table command:

Syntax: -

Case 1:

```
alter table <table_name>
add ( column_name1 datatype,
column_name2 datatype,
  :          :
,column_name n datatype);
```

Case 2:

```
alter table <table_name>
modify (column_name1 datatype,
 column_name2 datatype,
                :      :
 column_name n  datatype);
```

After you create a table, you may need to change the table structures because you omitted a column or your column definition needs to be changed. You can do this using the ALTER TABLE statement.

You can add columns to a table using the ALTER TABLE statement with the ADD clause.

e.g.

To add a column ADDRESS to the table EMP.

```
alter table emp
add  (address varchar2 (20));
```

You can modify existing column in a table by using the ALTER TABLE statement with the MODIFY clause.

e.g.

To modify the length of the NAME field to 30 in the EMP table.

```
alter table emp

modify  (ename varchar2 (30));
```

You can drop columns of a table using the ALTER TABLE statement with the DROP COLUMN clause.

e.g.

To drop the NAME field in the EMP table.

```
 alter table emp

drop column  (ename);
```

Result of the above alter table commands can be seen by describing the table.

Restrictions:

1. The new column becomes last column by default.
2. You can increase the width or precision of a numeric column.
3. You can change the datatype if the column contains null values (column is empty.
4. Decrease the width of a column if the column contains only null values or if the table has no rows.
5. You can convert a CHAR column to the VARCHAR2 datatype or convert VARCHAR2 column to CHAR datatype if the column contains null values or if you do not change the size.

Drop table command

The drop table statement removes the definition (structure) of an Oracle table. When you drop a table, the database loses all the data in the table and all the indexes associated with it.

Note: Create a dummy table and then use it in the drop table command.

Syntax: -

```
drop table <table_name>;
```

e.g

```
drop table emp;
```

Truncate table command

The truncate table statement is used to remove all rows from a table and to release the storage space used by the table.

Syntax: -

truncate table <table_name>;

e.g

truncate table emp;

Rename table command

Rename statement is used to rename a table, view, sequence, or a synonym.

Syntax: -

Rename <oldtable_name> to <newtable_name>;

e.g.

To change the name of the table EMP to EMPTBL

rename emp to emptbl;

Conclusion: Thus the Data Definition Language for creating, altering and dropping the table object in a database has been studied and implemented successfully.

Lab Exercise:

1. Create a table DEPT_TBL with the following fields.

Column	Datatype
Deptno	Number(5)
Dname	Varchar2(20)
Loc	Varchar2(20)

Note: Do not delete the EMP and DEPT tables since they will be used in all

the remaining LAB practicals.

2. Describe the structure of the DEPT table to identify the column names.
3. Change the size of Dname column to 30 in the DEPT_TBL table.
4. Change the name of DEPT_TBL table to DEPT.
5. What is the difference between DROP and TRUNCATE commands ?

Practical No. 2

Title:

To implement Domain and Entity Integrity Constraints on a Database.

Topic:

Types of Integrity Constraints

- Domain Integrity Constraints
- Entity Integrity Constraints
- Referential Integrity Constraints

Domain Integrity Constraints

It is used to maintain value according to the user specifications.

There are basically two types of domain integrity constraints.

- Not null constraint
- Check constraint

Not null constraint

By default all columns in a table allow Null values. When a 'Not Null' constraint is

Enforced on a column or a set of columns in a table, it will not allow null values.

At the time of table creation: -

Syntax: -

```
create table <table_name>
(column_name1   datatype(size),
 column_name2   datatype(size) not null,
                    :
,column_name n datatype(size));
```

e.g.

```
create table  emp( empno number(5),
ename varchar2(25) not null,
job varchar2(15),
mgr number(5),
joindate date,
salary number(7,2),
comm number(7,2),
deptno number(7,2) not null);
```

After the table creation: -

Syntax: -

```
alter table <table_name>
 modify <column_name> not null;
```

e.g.

```
alter table emp modify mgr not null;
```

Constraint Guidelines:

1. Constraints are easy to reference, if you give them a meaningful name.
2. If you do not name your constraint, Oracle generates a name with the format Sys_Cn where n is an integer to create a unique constraint name.
3. Constraints can be created at the time of table creation or after the table has been created.
4. Zero & Null are not equivalent.

Restriction:

"Not Null' integrity constraint can be defined using alter table command even when the table contains rows.

Check constraints

The Check Constraint defines a condition that each row must satisfy. A single column can have multiple check constraints that reference the column in its definition.

At the time of table creation: -

Syntax: -

```
create table <table_name>
(column_name1 datatype(size)
constraint <constraint_name>  check <condition>,
                    .
,column name n data type(size)) ;
```

e.g.

```
create table  emp(
empno number(5),
ename varchar2(25),
job varchar2(15),
mgr number(5),
joindate date,
salary number(7,2) constraint emp_sal_ck check(salary>500),
comm number(7,2),
deptno number(7,2));
```

After the table creation: -

Syntax: -

```
alter table <table_name>
add constraint <constraint_name>  check<condition>;
```

e.g.

```
alter table emp

add constraint emp_deptno_ch  check(deptno between 10 and 99);
```

Entity Integrity Constraints

An entity is any data recorded in a database. Each database represents a table and each row of a table represents an instance of that entity.

There are two types of entity integrity constraints.

- Unique constraints
- Primary key constraints

Unique constraint

It is used to prevent the duplication of values within the rows of a specified column or set of columns in a table. Columns defined with this constraint can also allow null values.If unique constraint is defined in more than one column i.e. combination of columns, it is said to be composite unique key.

At the time of table creation: -

Syntax: -

```
create table <table_name>

(column_name1  datatype(size),

column_name2  datatype(size) constraint

<constraint _name> unique ,

                    :

,column_name n  datatype(size));
```

e.g.

Table-level constraint

```
create table dept
(deptno  number(5),
dname  varchar2(20),
loc varchar2(20),
constraint dept_dname_uk  unique(dname));
```

Column-level constraint

```
create table dept
(deptno   number(5) constraint dept_dname_uk unique,
dname    varchar2(20),
loc   varchar2(20));
```

After the table creation: -

Syntax: -

```
alter table  <table_name>
add constraint <constraint_name>  unique (column_name);
```

e.g.

```
alter table dept
add constraint dept_deptno_uk  unique(deptno);
```

Composite unique key

e.g.

```
create table dept
(deptno    number (5),
dname      varchar2(20),
loc    varchar2(20),
constraint dept_compkey   unique(deptno, dname));
```

Constraint Guidelines:

1. UNIQUE key constraints can be defined at the column or table level definition.

2. Using the table level definition creates a composite unique key.
3. Maximum combination of columns that a composite unique key can contain is 16.

Primary key constraints

The Primary key constraint avoids duplication of rows and does not allow null values, when enforced on a column or set of columns. If a primary key constraint is assigned to a combination of columns, then it is said to be a composite primary key.

At the time of table creation: -

Syntax: -

create table <table_name>

(column_name1 datatype(size)

constraint <constraint_name> primary key,

column_name2 datatype(size),

:

column_name n datatype(size)) ;

e.g.

Column-level constraint

```
create table  dept
  (deptno number(5) constraint dept_deptno_pk  primary key,
   dname varchar2(20),
   loc varchar2(20));
```

Table-level constraint

```
create table  dept
(deptno  number(5),
dname varchar2(20),
loc varchar2(20),
constraint  dept_deptno_pk  primary  key (deptno));
```

After the table creation: -

Syntax: -

alter table <table_name>

add constraint <constraint_name> primary key (column_name) ;

e.g.

alter table emp

add constraint emp_empno_pk primary key (empno);

Guidelines:

1. Primary key can be defined at the column level or table level.
2. A composite primary key is created by using the table level definition (syntax is same as of unique key constraint).
3. A table can have only one primary key.
4. Primary key constraint cannot be defined in an alter table command when the table contains rows having null values.

Conclusion: Thus the Domain & Entity integrity constraints has been studied and implemented successfully.

Lab Exercise:

1. State True or False.
   - i) The primary key constraint allows null values.
   - ii) By default a table allows null records.
   - iii) A table containing null values can be altered to not null.
   - iv) Unique key allow null values.
2. Add a table level PRIMARY KEY constraint to the EMP table.
3. What is the difference between Not Null constraints and Unique constraints?
4. What is the difference between Unique key and Primary key?

Practical No. 3

Title:

To implement referential integrity constraints on a database.

Topic:

Referential integrity constraint

- Foreign Key

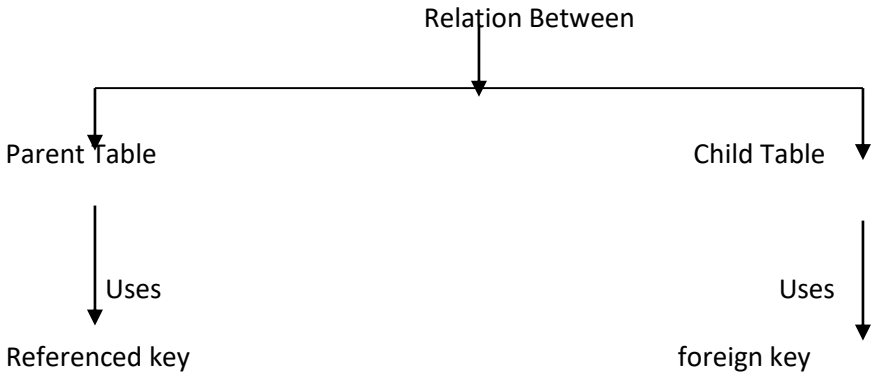
Proposition1: - Foreign key
Foreign key represents relationship between tables. A foreign key is a column (or a group of columns) whose values are derived from the 'primary key'.

Proposition2: - Referenced key
It is a unique or primary key which is defined on a column belonging to the parent table.

Proposition3: - Child table
This table depends upon the values present in the referenced key of the parent table, which is referred by a foreign key.

Proposition 4: - Parent table
This table determines whether insertion or updation of data can be done in child table. This table would be referred by child's table foreign key.

Referential Integrity Constraint.



Referential integrity constraints

It is used to establish a 'parent-child' relationship between two tables having a common column.

At the time of table creation: -

e.g.

Column-level constraint

```
create table emp
(empno  number(5),
 ename  varchar2(25) not null,
 job   varchar2(15),
 mgr number(5),
 joindate  date,
 salary  number(7,2),
comm  number(7,2),
deptno  number(7,2)
 constraint  emp_deptno_fk references dept(deptno));
```

Table-level constraint

```
create table  emp
(empno  number(5),
ename  varchar2(25) not null,
job  varchar2(15),
mgr  number(5),
joindate  date,
salary  number(7,2),
comm  number(7,2),
deptno  number(7,2) not null,
constraint emp_deptno_fk foreign key (deptno)
references dept (deptno));
```

Guidelines:

1. We can establish a relationship between a primary key or a unique key in the same table or a different table.
2. Foreign key constraint can be defined at the column or table constraint level.
3. A composite foreign key is creating by using the table-level definition**.**

After the table creation: -

e.g.

```
alter  table emp

add constraint emp_deptno_fk foreign key(deptno) references dept(deptno);
```

Viewing Constraints

e.g.

To display all the constraints on the EMP table.

```
select  constraint_name, constraint_type,

search_condition

from user_constraints

where table_name = 'EMP';
```

Result: - (Students will write the result).

e.g.

```
select * from user_constraints

where table_name = 'EMP';
```

Viewing the columns associated with constraints

e.g.

```
select constraint_name, column_name

from user_cons_columns

where table_name = 'EMP';
```

Result: - (Students will write the result).

On delete cascade

It indicates that when the row in the parent table is deleted, the dependent rows in the child table will also be deleted.

At the time of table creation

Syntax: -

```
create table <table_name>
(column_name1 datatype(size),
 column_name2 datatype(size)  constraint <constraint_name>
 references <parent_table_name> (parent_table_column_name)
on delete cascade, ……);
e.g.
create table emp
(empno      number(5),
ename       varchar2(25) not null,
job          varchar2(15),
mgr number(5),
joindate    date,
salary      number(7,2),
comm        number(7,2),
deptno      number(7,2)
constraint fk_dno references dept (deptno) on delete cascade);
```

After the table creation

Syntax: -

```
alter  table <table_name>
 add constraint <constraint_name>
foreign key <child_table_column_name> references
<parent_table_name> (parent_table_column_name) on delete cascade ;
```

e.g. :-

```
alter table emp
  add constraint fk_dno foreign key (deptno)  references
dept (deptno) on delete cascade;
```

Guidelines:

1. When a relation is created between two tables, it is permissible to delete records in a child table. But a vice-versa operation i.e. deleting records from a table when it references child table is not allowed.
2. Rejects an insert or update of a value, if a corresponding value does not currently exist in the primary key table.
3. On delete cascade allows automatic deletion of child records when a parent record is deleted.

Dropping integrity constraints

Dropping a constraint will allow future insertions or other manipulations without affecting existing data in the table.

Syntax: -

alter table <table_name> drop constraint <constraint_name>;

e.g.

A unique key specified on the salary column in the emp table can be dropped as shown below.

```
alter table emp drop constraint emp_sal_ck;
```

Conclusion: Thus the Referential integrity constraints has been studied and implemented successfully.

Lab Exercise:

1. Confirm the constraints were added by querying USER_CONSTRAINTS in the dept table. Note the types and names of the constraints.
2. Confirm the Columns Associated with Constraints were added by querying USER_CONS_COLUMNS in the dept table. Note the names of the columns and names of the constraints.

3. What would happen if a parent record is deleted before the child record is deleted?
4. Why should we name a constraint?

Practical No. 4

Title:

To use the Data Manipulation language for inserting, updating and deleting the data in the table object in a database.

.

Topic:

Types of DataManipulation Commands

- Insert
- Update
- Delete

Insert command

Syntax: -

insert into <table_name>

(column_name1,

column_name 2,

. . . . .

column_name n)

values

(expression1,

expression 2,

. . .

expression n);

Example:

To insert a row into the EMP table created in the previous experiment.

The above SQL statement will insert a single of data in the EMP table. In order to add more data to the EMP table, the insert command can be used with a new set of values each time.

Case 1:

```
insert into emp(empno, ename, job, mgr, joindate,salary,comm,deptno,
address )  values (1001, 'nilesh joshi','clerk',1005,'17-dec-1995',2800,
600,20,'nashik');
```

Case 2:

```
insert into emp values(1002,'avinash pawar', 'salesman',1003,
'20-feb-1996',5000,1200,30,'nagpur');
```

Case 3:

```
insert into emp values(&empno,'&ename','&job',&mgr,'&joindate',
&salary,&comm,&deptno,'&address');
Enter value for empno: 1003
Enter value for ename: amit kumar
Enter value for job: manager
Enter value for mgr: 1004
Enter value for joindate:  02-apr-1995
Enter value for salary: 20000
Enter value for comm: null
Enter value for deptno: 30
Enter value for address: pune
```

Emp Table

EMPNO	ENAME	JOB	MGR	JOINDATE	SALARY	COMM	DEPTNO	ADDRESS
1001	nilesh joshi	clerk	1005	17-dec-95	2800	600	20	nashik
1002	avinash pawar	salesman	1003	20-feb-96	5000	1200	30	nagpur
1003	amit kumar	manager	1004	02-apr-86	2000	-	30	pune
1004	nitin kulkarni	president	-	19-apr-86	50000	-	10	mumbai
1005	niraj sharma	analyst	1003	03-dec-98	12000	-	20	satara
1006	pushkar deshpande	salesman	1003	08-sep-96	6500	1500	30	pune
1007	sumit patil	manager	1004	01-may-91	25000	-	20	mumbai
1008	ravi sawant	analyst	1007	17-nov-95	10000	-	-	amravati

N**ote-** The above rows of data should be inserted in the emp table, since succeeding queries are based on this data.

Update command

Updating of all rows: -

Syntax: -

update <table_name>

set column_name1 = expression1,

column_name2 = expression2,

.

column_name n = expression n;

The update command is used to change or modify data values in a table.

e.g.

To change the commission of employee whose job is 'analyst', following command is used.

```
update emp
set comm = 800
where job = 'clerk';
```

To see result use select command

```
select job, comm from emp;
```

Delete command

Removal of all rows: -

Syntax: -

```
delete from <table_name>;
```

Removal of specified rows:-

```
delete from <table_name>
 where <search_condition>;
```

e.g

To delete rows from EMP_TEMP where empno = 1002,following command is used.

```
delete from emp
where empno = 1002;
```

To see whether rows have been deleted from EMP_TEMP table, use select command.

```
select * from emp_temp;
```

Conclusion: Thus the Data Manipulation Language commands for inserting, updating & deleting data in the table object in a database has been studied and implemented successfully.

Lab Exercise

1. Insert the following rows of data in the DEPART table.

Deptno	Dname	Loc
10	accounting	mumbai
20	research	pune
30	sales	nashik
40	operations	nagpur

2. Add the first row of sample data to the dept table from the list above. Do not list the columns in the INSERT clause.
3. Add the second row of sample data to the dept table from the list above. This time, list the columns explicitly in the INSERT clause.
4. Change the location of Department 40 to bangalore instead of nagpur.
5. Change the name of employee 1003 to ‘nikhil gosavi’.
6. Delete ‘Pushkar Deshpande’ from the EMP table.
7. What is the difference between truncate and delete commands?

Practical No. 5

Title:

To use Data Retrieving Language for selecting the data in the table object in a database.

.

Topic:

Data Retreival language command

- Select

Select command

Syntax: -

select column_name1, … ,column_name n

from <table_name>

OR

select *

from <table_name>;

Note-Meta character asterisk (*) is expanded by oracle to mean all columns in the table.Once data has been inserted into a table we can view what has been entered, with the select command.

e.g.

To see the contents of table EMP

select empno, ename , job, mgr, joindate, salary, comm, deptno, address

from emp ;

OR

select *from emp;

Selecting distinct rows

To prevent the selection of duplicate rows, we include distinct clause in the select command.

e.g.

select distinct deptno from emp;

Filtering table data

While viewing data from a table it is rare that all the data, from the table will be required each time.

The retrieval of specific columns from a table

Syntax: -

select column_name1,

column_name2

from <table_name>;

e.g

To select only the ENAME & EMPNO columns from the EMP table, folllowing command is used.

select empno, ename

from emp;

To retrieve selected rows & all columns from a table: -

ORACLE provides the option of using a 'WHERE' clause in an SQL sentence to apply a filter on the rows in the table.

When a 'WHERE' clause is added to the SQL statement, ORACLE compares each record from the table with the condition specified in the 'WHERE' clause.

Only those rows that satisfy the specified condition will be displayed.

Syntax: -

select * from <table_name>

where <search_condition>;

e.g

To select only those rows from the EMP table where the 'JOB' is 'analyst',

following command is used.

select * from emp

where job = 'analyst';

Note: The values specified in all the columns of a table are always case sensitive e.g :- 'analyst'

Conclusion: Thus the Data Retreival Language for selecting data in the table object in a database has been studied and implemented successfully.

Lab Exercise:

1. Create a query to display all the name, job, joindate and employee number for each employee number appearing first.
2. Create a query to display unique jobs from the EMP table.

Note- For these queries use the DEPART table created earlier.

Practical No. 6

Title:

To create the table from an existing table and to populate a table with the contents of an existing table.

Topic:

Preposition1: - Creating a table from an existing table

It is possible to create a table from an existing table. The created table is called the target table and the table from which the target table is created is called the source table.

The target table is identical to the source table.

Preposition2: - Inserting data into a table from another table

In addition to inserting data one row at a time into a table, it is possible to populate a table with data that already exists in another table.

Select command to create a table

Syntax: -

```
create table  <new_table_name>
(column_name1,…..,column_name n )
as select column_name1,
        .
,column_name n
from <existing_table_name>
[where <condition>]);
```

e.g.

To create a table similar to emp table but with only a few columns from emp table.

```
create table emp_temp
(id_no, name, deptno)
as select       empno,
                ename,
                deptno
from emp;
```

To see contents of emp_temp table

```
select * from emp_temp;
```

Select command to insert Records

Inserting data set into a table from another table.

Syntax: -

```
insert into <table_name>
select column_name1,
       .
 ,column_name n
from <existing_table_name>
[where <condition>];
```

e.g. To insert rows into emp_temp from emp table.

```
insert into emp_temp
select empno, ename, deptno
from emp;
```

- **Note** : - There are two sets of identical rows in the table EMP_TEMP. This is because one set of rows was created during the CREATE table As select command and other set of rows was created by the INSERT command.

Select command for Column Aliases.

The select command can be used to temporarily change a column name, when the query result is displayed. Column aliases come in handy, to shorten column names or if there is a need to hide the actual column from being displayed.

Syntax:

select column_name <alias_name> from table_name;

e.g.

select ename “Employees” from emp;

Conclusion: Thus the creation of a table from an existing table and to populate that table with the contents of an existing table has been studied and implemented successfully.

Lab Exercise:

1. Create a table dept_temp from table dept. Only create the structure not the contents.

Hint :- Use “where” clause in select statement .

2. Insert rows into dept_temp table from dept table.

3. State True or False.

i. We can use an Insert command to copy rows from one table to another.

ii. A table can be created using the Select c ommand.

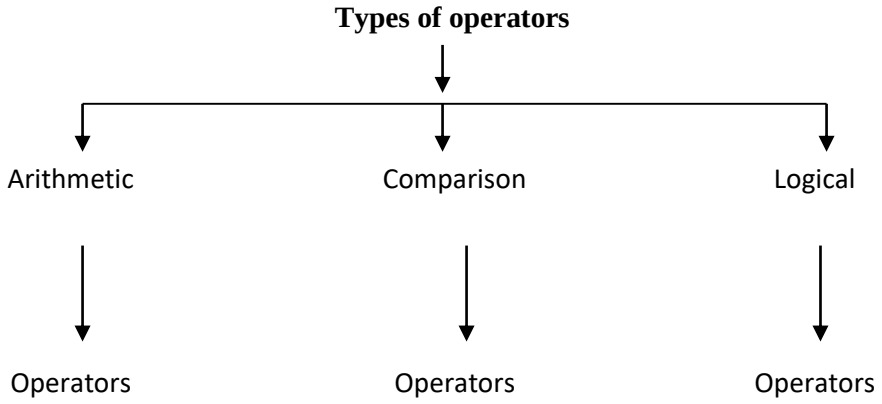
A column alias replaces the column name at display time only

Practical No. 7

Title:

To use arithmetic, comparison and logical operators in a database.

Topic:



Types of operators: -

1. Arithmetic operators.
2. Comparison or Relational operators.
3. Logical operators.

Arithmetic operators

To perform calculations based on number values, we include arithmetic expressions in SQL. An arithmetic expression consists of column names with number datatype and an arithmetic operator connecting them.

Examples: -

1. select salary + 2000 from emp;
2. select ename , salary + comm "Total Salary" from emp where deptno = 20;

Note:

1. * and / have equal higher precedence.
2. + and – have equal lower precedence.

The following example illustrates the precedence of operators.

e.g.

select 1.5 * (salary + comm) from emp;

In the above example, the result obtained from adding commission to salary is multiplied by 1.5

Note:

If parenthesis is omitted, then multiplication will be performed first followed by addition.

Comparison or Relational Operators

Comparison operators are used in conditions to compare one expression with another. Comparison operators are listed as below:

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
< >	Not equal to
! =	Not equal to

Examples: -

1) Display the list of employees whose job is ‘analyst’.

```
select  * from emp where job = ‘analyst’;
```

2) Display the list of employees whose salary is more or equal to 20000.

```
select * from emp where salary >= 20000 ;
```

3) Display the list of employees whose salary is less than 5000.

```
select ename,job,salary from emp where salary <5000;
```

4) Displa the list of employees excluding job title as ‘salesman’.

```
select * from emp where job != ‘salesman’;
```

Logical operators

A logical operator is used to combine the results of two conditions to produce a single result.

AND operator

It is used to combine the result of two conditions and both the conditions must be true for the entire condition to be true.

Examples:-

1) Display the list of employees whose salary is between 5000 and 20000.

```
select  * from emp where salary>= 5000 and salary <= 20000;
```

2) Display the list of employees whose joining date is between 01-apr-1995 and 10-sep-1996.

```
select * from emp

where joindate >= ‘01-apr-1995’ and  joindate <= ’10-sep-1996’;
```

OR operator

It is used to combine the result of two conditions and if one of the condition is true, then entire condition is true.

Examples: -

1) Display the list of employees whose manager is 1003 or department is 30.

```
select * from emp where mgr = 1003 or deptno = 30;
```

2) Display the list of employees who belongs to department 10 or 30.

```
select * from emp where deptno = 10 or deptno=30;
```

NOT operator

It is a negative of a single condition.

Examples: -

1) Display the list of employees whose salary is not less than 5000.

```
select * from emp where not salary < 5000 ;
```

2) Display the list of employees except job title as 'Manager'.

```
select * from emp where not (job = 'manager');
```

Conclusion: Thus the use of arithmetic, comparison and logical operators in adatabase has been studied and implemented successfully.

Lab Exercise:

1. Display the annual salary of all employees.
2. Display the employee details whose job is not Manager.
3. Display the employees whose salary is greater than Rs.5000 and less than or equal to Rs.10000.
4. Find the list of all employees who stay in a city Nashik or city Nagpur.
5. List all the employees who are located in Mumbai.

Practical No. 8

Title:

To use SQL character functions in a Database

Topic:

Proposition1: - To understand Function

A function takes one or more arguments and returns a value.

Proposition2: - to understand & use Dual table

Dual table has one column defined to be of varchar2 datatype and contains only one

row with value 'x'.

. **Single row functions**

A single row function returns only one value for every row queried in the table.

The single row functions can be classified as:

1. Character functions
2. Numeric functions
3. Date functions
4. Conversion functions

Character functions

Character functions accept character input and return either character or number

values.

1. **Initcap** - It returns the input string with initial letter capitalized and all other characters in lowercase.
   Format: - initcap (char)

   e.g.

   i. select initcap('hello') from dual;

   ii. select initcap(ename) from emp;

2. **Lower** - It returns the input string with all letters in lowercase.

Format: - lower (char)

e.g.

select lower('HELLO') from dual;

3. **Upper** - It returns the input string with all letters in uppercase.
   Format: - upper (char)

   e.g.

   select upper('sunny') from dual;

4. **Ltrim** - It trims from the left of character string.

   Format: - ltrim (char, set)

   e.g.

   select ltrim('Nilesh Joshi', 'Nilesh') from dual;

5. **Rtrim** - It trims from the right of character string.

   Format: - rtrim (char, set)

   e.g.

   select rtrim('Nilesh Joshi' , 'Joshi') from dual;

6. **Translate** -
   Format: - translate (char, from, to)

   e.g.

   select translate ('Jack', 'J ' , 'b') from dual;

7. **Replace:** - It returns character string with each occurrence of searchstring replaced with [Repstring].
   Format: - Replace (char, searchstring, [Repstring])

   e.g.

   select Replace ('Jack and Jue', 'J', 'bl') from dual;

8. **Substr** - It returns the substring of character string that stack at m character and is of length n.

   Format: - substr (char, m, n)

   where

   m is position

   n is no. Of characters.

   e.g.

```
select substr('Ravindra', 2, 3) from dual;
```

9. **Lpad:** - It will left-pads string1 with as many iterations of string 2 as is needed to make total result n characters long.

   Format: - lpad(char, length, padding by)

   e.g.

```
select lpad('function', 15 ,'*' )from dual;
```

10. **Rpad** - It will right-pad string1 with as many iterations of string2 as is needed to make result n characters long.

    Format: - rpad (char, length, padding by)

    e.g.

```
select rpad ('function', 15, '*') from dual;
```

11. **Length** - It returns the length of the character string.

    Format: - length (char)

    e.g.

```
select length('Hello') from dual;
```

12. **Concatenation (||) operator-**The concatenation operator is used to merge two or more strings, or a string and a data value together.

    e.g.

```
select ('The address of ' || ename || ' is ' || address)  Address from emp  Where

empno=1004;
```

Conclusion: Thus the use of SQL character functions in adatabase has been studied and implemented successfully.

Lab Exercise:

1. Write a query that will display the employee's name with the first letter capitalized and all other letters lowercase and the length of their names.
2. Create a query to display name and salary for all employees. For the salary to be 15 characters long, left padded with $.
3. What is the difference between Translate and Replace functions?

Practical No. 9

Title:

To use comparison operators in ORACLE for range searching and pattern matching of table data.

Topic:

Comparison operators

Proposition1: - Range searching

In order to select data that is within a range of values, the BETWEEN operator is used

Proposition 2: - Pattern matching

ORACLE provides a LIKE predicate , which allows comparison of one string with another string value which is not identical. This is achieved by using wildcard characters. Two wildcard characters that are available are

For character data types

1) Percent sign (%) :- It matches any string.

2) Underscore (_) :- It matches any single character.

IN and NOT IN Predicates

In case a value needs to be compare to a list of values, then the IN predicate is used. We can check a single value against multiple values by using the IN predicate.

Comparison Operators:

1) IN, NOT IN
2) BETWEEN, NOT BETWEEN
3) LIKE, NOT LIKE
4) IS NULL , IS NOT NULL
5) ANY, ALL

Syntax : -IN

select column_name1, column_name2,…….,column_name n

from < table_name >

where column_name in (‘value1’,’value2’,……,’valuen’);

e.g.

select empno, ename , job

from emp

where job in (‘analyst’, ‘clerk’);

The above example displays employee number, name and job of all employees whose job is ‘analyst ‘ or ‘clerk’.

Syntax: - NOT IN

select column_name1, ……..,column_name n

from < table_name >

where column_name not in (‘value1 ‘,……..,’value n’);

e.g.

select empno, ename, job

from emp

where job not in (‘salesman’, ’clerk’);

The above example will display employee number, name and job where the JOB is not a salesman or clerk.

Guidelines:

1) The IN & NOT IN operators can be used with any datatype.
2) If characters or dates are used in the list, they must be enclosed in single quotation marks (‘ ‘).

Syntax: - LIKE using % (Percentage)

select column_name1,column_name2,…… ,column_name n

from < table_name>

where column_name like ‘value%’ ;

e.g.

select ename, job, salary

from emp

where ename like ‘ni%’;

The above example will display all rows from table EMP where employee name begins with ‘ni ’.

Syntax:-NOT LIKE using %(Percentage)

select column_name1,column_name2,……..,column_name n

from < table_name>

where column_name not like ‘value%’;

Note:- ‘value’ is a string literal.

e.g.

select ename, job, salary

from emp

where ename not like ‘ni%’;

The above example will display all rows where the employee name does not begin with ‘ni ’ from the EMP table.

Syntax: -Like using ‘_’ (underscore)

```
select  column_name1,……..,column_name n

from < table_name>

where column_name like  ‘__value%’ ;
```

e.g.

```
select empno, ename,address

from emp

where address like ‘_une’ ;
```

The above example will display all rows where the address starts with any character but ends with ‘une’.

Syntax: -BETWEEN

```
select column_name1,…….,column_name n

from < table_name>

where column_name between lowerbound and upperbound ;
```

You can display rows based on a range of values using BETWEEN operator . The range that you specify contains a lower bound and upper bound.

e.g.

```
select  ename,salary,comm

from emp

where comm  between 1000 and 3000;
```

Syntax :- NOT BETWEEN

```
select    column_name1,…….,column_name n

from < table_name>

where column_name not between lowerbound and upperbound;

e.g.

select  ename, salary, comm

from  emp

where  comm  not between 1000 and 3000;
```

Guidelines:

1) Values specified with the BETWEEN operator are inclusive.
2) You must specify the lower limit first.

Syntax :-IS NULL

```
select column_name1,…….,column_name n

from < table _name >

where column_name is null;
```

The IS NULL Operator tests for values that are Null. A Null value means the value is unavailable , unassigned, unknown,unequal to any value or zero length string.

e.g.

Display all the records from EMP table where mgr is Null.

```
select empno, ename,mgr, job

from  emp

where mgr is null;
```

Syntax:- IS NOT NULL

```
select column_name1,…….,column_name n

from< table_name>

where column_name is not null;
```

e.g.

```
select empno, ename, mgr, job
from emp
where mgr is not null;
```

ANY Operator

The ANY operator compares a value to each value returned by a subquery.

< ANY means less than the maximum.

>ANY means more than the minimum.

= ANY is equivalent to IN.

ALL Operator

The all operator compares a value to every value returned by a subquery.

>ALL means more than the maximum.

< ALL means less than the minimum.

- **Note:** ANY and ALL operators are covered in the subquery Topics.

Conclusion: Thus the use of comparison operators in ORACLE for range searching and pattern matching of table data.in adatabase has been studied and implemented successfully.

Lab Exercise:

1. Display all those employees whose job title is either “Manager” or “Analyst”.
2. Display the employee name and department number of all employees in departments 10 and 30.
3. List the names of employees not in the city of Pune.
4. List the name and salary of employees who earn more than Rs.5000 and are in department 10 or 30.
5. Display the employee number, name, job and commission of all those employees who do not get any commission.
6. Display the name, job and start date of employees joined between February 20,1996, and December 1, 1996.
7. Display the name and salary for all employees whose salary is not in the range of Rs.5000 and Rs.10000.

8. Find all names and joindate of employees whose names start with letter “a”.
9. Find the names of all employees having “ i “ as the second letter in their names.

10. Find out the employees who stay in a city whose second letter is not “a”.

Practical No. 10

Title:

To use order by, group by, and having clause in a database

Topic:

Proposition1**: -** Sorting of Oracle table data.

The order of rows returned in a query result is undefined. Oracle server may not fetch rows in the same order for the same query twice. To prevent this from happening, the ORDER BY clause is used.

Proposition2: - Grouping rows in a table.

The rows of data in an Oracle table can be divided into groups by using the GROUP BY clause.The HAVING clause can be used to restrict groups from being displayed.

Information :

1. Order by clause
2. Group by clause
3. Having clause

Order by clause

Syntax : -

select <expr>

from <table_name>

[where condition (s)]

[order by {column, expr} [asc | desc]];

where order by - specifies the order in which the rows are displayed.

asc - orders the rows in ascending order.

This is the default order.

desc - orders the rows in descending order.

e.g.

```
select ename, job, deptno, joindate
from emp
order by joindate desc;
```

The above example sorts the result by the employees joining date.

Guidelines:

1. The ORDER BY clause should be placed last in the query.
2. The default sort order is ascending.

Sorting by column aliases:

You can use a column alias in the ORDER BY clause.

e.g.

```
select empno, ename, salary*12  annsal
from emp
order by annsal;
```

The above example sorts the result by the alias annsal which represents the annual salary.

Note: You can sort query results by more than one column. The sort limit is the number of columns in the given table.

Group by clause

Syntax: -

select column, group_function (column)

from <table_name>

[where condition]

[group by group_by_expression]

[order by column];

where

Group by expression - specifies columns whose values determine the basis for grouping rows.

The GROUP BY clause can be used to divide the rows in a table into groups. We can then use the group functions to return summary information for each group.

e.g.

select deptno, avg (salary)

from emp

group by deptno;

Note : avg (sal) is a group function which calculates the average salary for each department since the rows are being grouped by department number.

The GROUP BY column does not have to be in SELECT list

e.g.

```
select avg (salary)
from emp
group by  deptno;
```

In the above e.g. the Group By column deptno is not in using the GROUP BY

clause on multiple columns.

e.g.

```
select deptno, job, sum(salary)
from  emp
group by deptno, job;
```

In the above example we are having a Group by on multiple columns.

First the rows are grouped by department number.

Second, within the department number groups, the rows are grouped by job title.

Having clause

Syntax: -

```
select column, group_function
from table
[where condition]
[group by group_by_expression]
[having group_condition]
[order by column];
```

where

group condition - restricts the groups of rows returned to those groups for which the specified condition is TRUE.

You use the HAVING clause to specify which groups are to be displayed.

The Oracle server performs the following steps when you use the HAVING clause.

1. Rows are grouped.
2. The group function is applied to the group.
3. The groups that match the criteria in the HAVING clause are displayed.

e.g.

```
select  deptno, max (salary)
from emp
group by deptno
having max (salary) > 10000;
```

Guidelines:

1. If you include group function in a SELECT clause, you cannot select individual results as well unless the individual column appears in the GROUP BY clause. Any column or expression in the SELECT list that is not an aggregate function must be in GROUP BY clause.
2. You cannot use the column alias in the GROUP BY clause.
3. By default rows are sorted by ascending order of the columns included in the Group By clause.
4. You cannot use the WHERE clause to restrict groups.
5. You use the Having clause to restrict groups.
6. The HAVING clause can precede the GROUP BY clause, but it is recommended that you place the GROUP BY clause first because it is more logical.

Conclusion: Thus the use of order by, group by, and having clause in a database has been studied and implemented successfully.

Lab Exercise:

1. Display the average salaries for each department.
2. Display the department numbers and average salary for those departments whose minimum salary is greater than Rs.3000.
3. Display all the employee information in the ascending order of joining date.
4. Display the minimum, maximum, sum and average salary for each job type.

Practical No. 11

Title:

To use SQL Numeric functions in a database.

Topic:

Proposition1**: -** Study of SQL Numeric Functions

Numeric Functions

Numeric function accepts the numeric value as the input and it returns the numeric values as the output.

The numeric functions are as follows.

1. **Abs** - It returns the absolute value of the number.
   Format:- abs(n)

   where n is number.

   e.g. select abs(-15) from dual;

2. **Cos** - It returns cosine of the number.
   Format: - cos(n)

   e.g. select cos(180) from dual;

3. **Cosh** - It returns the hyperbolic cosine of the number.
   Format: - cosh (n)

   e.g. select cosh(1) from dual;

4. **Exp** - It returns the e raised to the power of the number.
   Format: - exp(n)

   e.g. select exp(4) from dual;

5. **Ceil** - It returns the smallest integer that is larger than the number.
   Format: - ceil (n)

   e.g. select ceil(44.778) from dual;

6. **Floor** - It returns the largest integer that is smaller than the number.
   Format: - floor(n)

   e.g. select floor(100.2) from dual;

7. **Round** - it will round m to n number of digits after the decimal point.
   Format: - round(m,n)

   where m - is number

   n- is number of digits after decimal point

   e.g. select round (100.256,2) from dual;

8. **Trunc** - It truncates m to n places after the decimal point.
   Format: - trunc(m,n)

   Where m is number

   n is no of digits after decimal point.

   e.g. select trunc (100.256,2) from dual;

9. **Power** - It returns m raised to the n power.
   Format: - power(m,n)

   where m,n are numbers.

   e.g. select power(5,2) from dual;

10. **Mod** - It returns the remainder when m is divided by n.
    where m,n are numbers.

    e.g. select mod(10,3) from dual;

11. **Sqrt** - it returns the positive square root of a positive number.
    Format: - sqrt(n)

    e.g. select sqrt(4) from dual;

12. **Sin** - It returns the sine of the number.
    Format: - sin(n)

    e.g. select sin(1) from dual;

13. **Sinh** - It returns the hyperbolic sine of the number.
    Format: - sinh (n)

    e.g. select sinh(1) from dual;

14. **Tan** - It returns the tangent of the number.
    Format: - tan(n)

    e.g. select tan(1) from dual;

Conclusion: Thus the use of SQL Numeric functions in a database has been studied and implemented successfully.

Lab Exercise:

1. Display the absolute value of the following numbers.

   i) –25 ii) –45 iii) –105

2. Display sine, cosine and tangent values of the following numbers.

   i) 5 ii) 9 iii) 25

3. Display the cube of the following numbers.

   i) 5 ii) 6 iii) 8.

4. Display the exponential value of the following numbers.

   i) 5 ii) 7 iii) 9

Practical No. 12

Title:

To use group functions in a database.

Topic:

Group functions

A group function returns a result based on a group of rows.

The group functions are listed below:

1. **Avg** - It returns the average value of the specified column of number

   data type.

   Format: - avg (column_name)

   e.g. select avg (salary) from emp;

2. **Sum** - It returns the summation of the specified column of number datatype.
   Format: - sum (column_name)

   e.g. select sum (salary) from emp;

3. **Min** - It returns the lowest value from the specified column of number

   datatype.

   Format: - min (column_name)

   e.g. select min(salary) from emp;

4. **Max** - It returns the highest value of the specified column of number datatype.

   Format: - max (column_name)

   e.g. select max (salary) from emp;

5. **Count** - Count function is used to count the number of rows.
   Count (*) - It counts all rows, inclusive of duplicate and nulls.

   e.g. select count (*) from emp;

   Count (column_name)

   It is used to count the number of values present in the specified column without including nulls.

   e.g select count (comm) from emp;

   Count (distinct column_name)

   It is used to eliminate the duplicate and null values in the specified column.

   e.g. select count (distinct deptno) from emp;

Conclusion: Thus the use of group functions in a database has been studied

and implemented successfully.

Lab Exercise:

Display the highest, lowest, sum and average salary of all employees.

Label the columns Maximum, Minimum, Sum and Average respectively.

1. Write a query to display the number of people with the same job.
2. Determine the number of managers without listing them. Label the column number of managers.
3. Write a query that will display the difference between the highest and lowest salaries. Label the column DIFFERENCE.
4. Display the employee name that is first and employee that is last in an alphabetized list of all employees.
5. Display the number of employees in department 30 who can earn a commission.

Practical No. 13

Title:

To use SQL date functions in a database.

Topic:

Different Single row functions (Date functions)

1. MONTH_BETWEEN
2. ADD_MONTHS
3. NEXT-DAY
4. LAST_DAY
5. ROUND
6. TRUNC

Proposition1: - Date functions

Date functions operate on ORACLE dates. All date functions return a value of date datatype except months_between , which returns a numeric value

Proposition2: - Sysdate

Sysdate is a pseudo column that contains the current date and time. It requires no arguments when selected from the table Dual and returns the current date.

Months_between

Months_between finds the number of months between d1 & d2.

If date d1 is later than date d2 the result is positive.

If date d1 is earlier than date d2 the result is negative.

Format:

months_between (d1, d2)

where d1 & d2 are dates

e.g.

select months_between ('02-feb-1993', '02-jan-1993')

“months" from dual;

Add_months

Returns date after adding the number of months specified with the function.

Format:

add_months (d,n)

where d - is the date

n- number of months to be added to the date

e.g.

select add_months (sysdate, 4)

from dual;

Next_day

Returns the date of the first weekday named 'char' that is after the date named by date.

Format:

next_day (date, char)

where

char - day of the week

e.g.

select next_day ('04-feb-1998', ‘Friday’)

"next day" from dual;

Last_day

Returns the last day of the month that contains date 'd'.

Format:

Last_day (d)

where d-date

e.g.

select last_day (sysdate) "last"

from dual;

Round

Returns date rounded to the unit specified by the format model "fmt".If the format model 'fmt' is omitted, date is rounded to the nearest date.

Format:

round (date [,'fmt'])

e.g.1

select round (sysdate, 'month') "round"

from dual;

e.g. 2

select round (sysdate,'year') "round"

from dual;

Trunc

Returns date with the time portion of the day truncated to the unit specified by the format model fmt. If the format model fmt is omitted, date is truncated to the nearest day

Format:

trunc (date [, 'fmt'])

e.g. 1.

select trunc (sysdate,'month') "trunc"

from dual;

e.g. 2

select trunc (sysdate,'year') "trunc"

from dual;

Conclusion: Thus the use of group functions in a database has been studied and implemented successfully.

Lab Exercise:

1. Write a query to display the current date. Label the column Date.
2. Display the employee name and calculate the number of months between today and the date the employee was joined. Label the column MONTHS_WORKED. Round the number of months up to the closest whole number.
3. Display the name, joindate and day of the week on which the employee started. Label the column DAY.
4. Display the current date and the last day of the current month.

Practical No. 14

Title:

To use data conversion in ORACLE using the various data type conversions functions

in SQL.

Topic:

Proposition1: - Conversion of datatypes in Oracle

SQL provides three functions to convert a value from one datatype to another

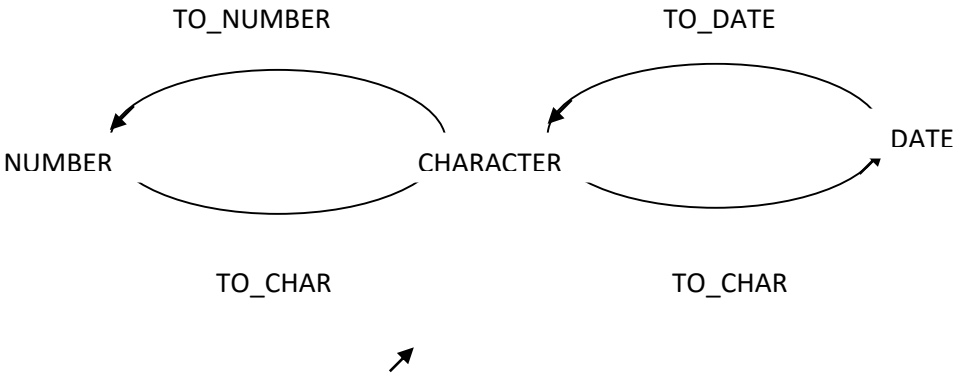
1. To_char
2. To_number
3. To_date

To_char function - It converts a number or date value to a VARCHAR2 character string with format model.

To_number function - converts a character string containing digits to a number.

To_date function - converts a character string representing a date to a date value according to the format specified.

Explicit datatype conversion:



To_char function with numbers

Syntax: -

To_char (number, ['fmt'])

where fmt - is the format model used for the conversion

e.g.

```
select to_char (comm, '$99,999')  comm
 from emp
 where job = 'salesman';
```

Guidelines:

1. The ORACLE server displays a string of pound sign (#) in place of a whole number whose digits exceed the number of digits provided in the format model.
2. The ORACLE server rounds the stored decimal value to the number of decimal spaces provided in the format model.

To_char function with Dates

Syntax: -

To_char (date, 'fmt')

Fmt - format

Previously all ORACLE date values were displayed in the 'DD-MON-YY' format. The TO_CHAR function allows you to convert a date from this default format to one specified in the 'fmt'.

e.g.

```
select empno, to_char (joindate, 'mm/yyyy') month_hired
from emp;
```

Date Format Model Elements:

YYYY - Full year in numbers

Year - Year spelled out

MM - 2- digit value for month

MONTH - Full name of the month

DY - 3-letter abbreviation of the day of the week

DAY - Full name of the day

Time Format :

AM or PM - Meridian Indicator

A.M or P.M - Meridian Indicator with periods

HH or HH12 or HH24 - Hour of day or hour (1-12) or hour (0-23)

MI - Minute (0-59)

SS - Second (0-59)

e.g. 2

Select empno,to_char (joindate, 'dd month yyyy') joindate

from emp;

Guidelines:

1. The format model must be enclosed in single quotation marks and is case sensitive.
2. The format model can include any valid date format element. Be sure to separate the date value from the format model by a comma.
3. The names of days and months in the output are automatically padded with blanks.
4. To remove the padded blanks or to suppress leading zeros, use the fill mode fm element.

To_number

Syntax: -

To_number (char)

Converts a 'char', a character value containing a number, to a value of NUMBER datatype.

e.g.

update emp

set comm =comm + to_number (substr ('$100', 2,3));

The above example will add 100 to the comm of every employee in the EMP table.

To see results use

select empno, comm from emp;

Conclusion: Thus the use data conversion in ORACLE using the various data type conversions functions in SQL has been studied and implemented successfully.

Lab Exercise:

1. Display the joindate of all the employees in the EMP table showing day, month, year & time of joining.
2. Display the names and joining dates of all employees who joined in the month of April.
3. Display the system date to appear in the format similar to “03 rd of APRIL 1975”.
4. Display the joining date of all employees to appear in the format similar to “Sunday, the Seventh of September, 1995.”

5. Display “20th OCTOBER 1999” in the date format.

Practical No. 15

Title:

To relate data through join concept in a Database .

Topic:

Proposition1: - Join

Join is used to combine the data spread across tables. A join is performed by the 'where' clause which combines the specified rows of tables.

Types of Joins

1. Equi - join
2. Non equi-join
3. Self Join
4. Outer Join

Equi -join

A join, which is based on equalities, is called equi-join. In equi-join comparison operator equal to (=) is used to perform a join. It retrieves rows from tables having a common column. It is also called simple join.

Syntax : -

```
select table1.column , table1.column,
table2.column,….,
from table1, table2
where table1.column1 = table2.column2;
```

e.g

```
Select emp.empno,emp.ename,emp.deptno,
dept.deptno,dept.loc
from emp,dept
where    emp.deptno = dept.deptno;
```

Guidelines:

1. Rows in one table can be joined to rows in another table according to the common values existing in corresponding columns, that are usually primary and foreign key columns.
2. When writing a select statement that joins tables, precede the column name with the table name.
3. If the same column name appears in more than one table, the column name must be prefixed the table name.
4. If the column names are unique, then we need not prefix it with the table name.

Table aliases

Table aliases are used to make multiple table queries shorter and more readable. As a result, we give an alias name or short name to the table in the 'from' clause. The alias can be used instead of the table name throughout the query.

e.g.

```
select e.empno, e.ename, e.deptno,
d.deptno, d.loc
from emp e, dept d
where e.deptno = d.deptno;
```

Note: - The above example is same as example of equi join but uses table aliases where 'e' refers to emp table and 'd' refers to dept table.

Non equi-join

A join that specifies the relationship between columns belonging to different tables by making use of the relational operators (<, >, <=, >=, ! =) other than '=' operator is called as non equi-join.

To use non equi-join create the following table.

Table - Salgrade

Column-name	Datatype
Grade	Number (4)
Losal	Number (8)
Hisal	Number (8)

Note - Students will enter the following records in a given table.

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

To relate emp and salgrade tables using non-equi join.

e.g

```
select e.ename, e.salary, s.grade
from emp e, salgrade s
where e.salary
between s.losal and s.hisal;
```

OR

```
select e.ename, e.salary, s.grade
from emp e, salgrade s
where e.salary >= s.losal and
        e.salary <= s.hisal;
```

Self join

Joining of a table to itself is known as self join. i.e. it joins one row in a table to another. It can compare each row of the table to itself and also with other rows of the same table.

e.g.

To find the name of each employee's manager you need to join EMP table to itself.

```
select worker.ename "employee", manager.ename "manager"
from emp worker, emp manager
where worker.mgr = manager.empno;
```

Note: The above example joins the emp table to itself. To stimulate two tables in the FROM clause, there are two aliases, namely WORKER and MANAGER, for the same table, EMP.

Outer Join

An outer join returns all the rows returned by simple join or equi join as well as those rows from one table that do not match any row from the other table.The symbol (+) represents outer join.

Syntax: -

Case 1
```
select table.column, table.column
from table1,table2
where table1.column (+) = table2.column;
```

Case 2
```
select table.column, table.column
from table1, table2
where table1.column = table2.column (+);
```

e.g

```
select e.ename, d.deptno, d.dname
from emp e, dept d
where e.deptno (+) = d.deptno
order by e.deptno;
```

The above example displays numbers and names for all the departments. The operations department, which does not have any employees, is also displayed.

Guidelines :

1. The outer join operator can appear only on one side of the expression. The side that has information missing. It returns those rows from one table that has no direct match in the other table.
2. A condition involving an outer join cannot use the IN operator or be linked to another condition by the OR operator.

Conclusion: Thus the join concept in a Database has been studied and implemented successfully.

Lab Exercise:

1. Display employee Ravi's employee number, name, department number, and department location.

2. Display the list of employees who work in the Sales department.

3. Display the list of employees who do not work in the Sales department.

Practical No. 16

Title:

Usage of sub queries in a database.

Topic:

Proposition1: - Subquery

Subquery is a SELECT statement that is embedded in a clause of another SELECT statement. i.e. nesting of queries or query within query.

Types of subqueries

1. Single row subqueries
2. Multiple row subqueries
3. Multiple column subqueries

Single row subqueries

A Single row subquery is one that returns one row from the inner SELECT statement. This type of subquery uses a single row operator that are listed below:

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

e.g.

1. Display the employee details whose job title is the same as that of employee 1005.

```
select empno, ename, job, salary, deptno
from emp
where job = (select job from
emp where empno = 1005);
```

2. Display the employee details whose salary is equal to the minimum salary.

```
select empno, ename, job, salary
from emp
where salary = (select min(salary)  from emp);
```

3. Display all the departments that have a minimum salary greater than that of department 20.

```
select deptno, min(salary)
from emp
group by deptno
having min(salary) > (select min(salary) from emp where deptno = 20);
```

Guidelines:

1. A SELECT statement can be considered as a query block.
2. Main select statement is called as outer query block.
3. Subquery is called as inner query block.
4. The inner query block executed first.
5. The outer query block is then processed and uses the values returned by the inner queries to complete its search conditions.
6. Inner queries return single values, so this SQL statement is called a single row subquery.
7. The outer and the inner queries can get data from different tables.
8. We can display data from a main query by using a group function in a subquery to return a single row.
9. We can use subqueries not only in the where clause, but also in the HAVING clause.

Multiple-row subqueries

Subqueries that return more than one row are called multiple-row subqueries. We use a multiple-row operator instead of a subquery, with a multiple row subquery. These operators are listed below:

Operator	Meaning
IN	Equal to any member in the list..
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery.

e.g

1. Find the employees who earn the same salary as the minimum salary for departments.

```
Select empno, ename, salary, deptno

From emp

Where salary in (select min (salary) from emp group by deptno);
```

Multiple column subqueries

Query that returns the values from more than one column are called multiple column subqueries.

Syntax:

```
select column_name1, column_name2, ……

from <table_name>

where (column_name, column_name,….) in

(select column_name, column_name,….from <table_name>

where <condition> );
```

e.g.

1. Display the name, department number, salary and commission of any employee whose salary and commission matches both the commission and salary of any employee in department 10.

```
select ename, deptno, salary, comm from  emp
where (salary, comm) in (select salary, comm from emp
where deptno = 30);
```

Common guidelines for all types of subqueries:

1. A subquery must be enclosed in parenthesis.
2. A subquery must appear on the right side of the comparison operator.
3. Subqueries cannot contain an ORDER BY clause for a SELECT statement and if specified it must be the last clause in the main SELECT statement.
4. Two classes of comparison operators are used in subqueries:
    1. Single–row operators
    2. Multiple-row operators

Conclusion: Thus the usage of sub queries in a database .has been studied and implemented successfully.

Lab Exercise:

1. Write a query to display the employee name and joindate for all employees in the same department as Sumit Patil. Exclude Sumit Patil.

2. Create a query to display the employee number and name for all employees who earn more than the average salary. Sort the results in descending order of salary.
3. Display the employee name, department name and job title for all employees whose department location is Nashik.
4. Display the employee name and salary of all employees who report to Amit Kumar.

Practical NO : 17

Title:

Introduction to PL/SQL and Study PL/SQL data Types

About PL/SQL

Procedural Language/SQL (PL/SQL) is Oracle Corporation's procedural language extension to SQL, the standard data access language for relational databases. PL/SQL offers modern software engineering features such as data encapsulation, exception handling, information hiding, and object orientation, and so brings state-of-the-art programming to the Oracle Server and Toolset.

PL/SQL incorporates many of the advanced features made in programming languages designed during the 1970s and 1980s. It allows the data manipulation and query statements of SQL to be included in block-structured and procedural units of code, making PL/SQL a powerful transaction processing language. With PL/SQL, you can use SQL statements to finesse Oracle data and PL/SQL control statements to process the data.

1.1

1.1.1 Block structure

DECLARE - Optional

Variables, cursors, user-defined exceptions

BEGIN - Mandatory

SQL statements, PL/SQL statements

EXCEPTION - Optional

Actions to perform when errors occur

END; - Mandatory

Section	Description	Inclusion
Declarative	Contains all variables, constants, cursors, and user-defined exceptions that are referenced in the executable and declarative sections	Optional
Executable	Contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block	Mandatory
Exception handling	Specifies the actions to perform when errors and abnormal conditions arise in the executable section	Optional

1.111Executing Statements and PL/SQL Blocks from SQL*Plus

Place a semicolon (;) at the end of a SQL statement or PL/SQL control statement.

- Use a slash (/) to run the anonymous PL/SQL block in the SQL*Plus buffer. When the block is executed successfully, without unhandled errors or compile errors, the message output should be as follows:

  PL/SQL procedure successfully completed

Note: In PL/SQL, an error is called an *exception.*

Section keywords DECLARE, BEGIN, and EXCEPTION are not followed by semicolons. However, END and all other PL/SQL statements do require a semicolon to terminate the statement.

Use of Variables

- **Temporary storage of data**

Data can be temporarily stored in one or more variables for use when validating data input for processing later in the data flow process.

- **Manipulation of stored values**

Variables can be used for calculations and other data manipulations without accessing the database.

- **Reusability**

Once declared, variables can be used repeatedly in an application simply by referencing them in other statements, including other declarative statements.

- **Ease of maintenance**

When using %TYPE and %ROWTYPE, you declare variables, basing the declarations on the definitions of database columns. PL/SQL variables or cursor variables previously declared in the current scope may also use the %TYPE and %ROWTYPE attributes as datatype specifiers. If an underlying definition changes, the variable declaration changes accordingly at runtime. This provides data independence, reduces maintenance costs, and allows programs to adapt as the database changes to meet new business needs.

Types of Variables

Scalar datatypes hold a single value. The main datatypes are those that correspond to column types in Oracle Server tables; PL/SQL also supports Boolean variables.

Composite datatypes such as records allow groups of fields to be defined and manipulated in PL/SQL blocks.

Reference datatypes hold values, *called pointers,* that designate other program items.

LOB datatypes hold values, called *locators,* that specify the location of large objects (graphic images for example) that are stored out of line.

Declaring PL/SQL Variables

Syntax

Identifier [CONSTANT] *datatype* [NOT NULL] [: = | DEFAULT *expr]*;

Where

Identifier is the name of the variable

CONSTANT constrains the variable so that its value cannot change; constants must be initialized

Datatype is a scalar, composite, reference, or LOB datatype

NOT NULL constrains the variable so that it must contain a value (NOT NULL variables must be initialized.)

Expr is any PL/SQL expression that can be a literal, another variable, or an expression involving operators and functions

Examples

```
DECLARE
v_hiredate      DATE;
v_deptno        NUMBER(2) NOT NULL := 10;
v_location      VARCHAR2(13) := 'Atlanta';
c_comm.         CONSTANT NUMBER : = 1400;
```

Note: Two variables can have the same name, provided they are in different blocks.

The variable name (identifier) should not be the same as the name of table columns used in the block.

Base Scalar Datatypes

Data Type	**Description**
VARCHAR2 *(maximum length}*	Base type for variable-length character data up to 32,767 bytes. There is no default size for VARCHAR2 variables and constants.
NUMBER *[{precision, scale)}*	Base type for fixed and floating-point numbers.
DATE	Base type for dates and times. DATE values include the time of day in seconds since midnight. The range for dates is between 4712 B.C. and 9999 A.D.
CHAR *[(maximum length}}*	Base type for fixed-length character data up to 32,767 bytes. If you do not specify a *maximum length,* the default length is set to 1.
LONG	Base type for variable-length character data up to 32,760 bytes. The maximum width of a LONG database column is 2,147,483,647 bytes.
LONG RAW	Base type for binary data and byte strings up to 32,760 bytes. LONG RAW data is not interpreted by PL/SQL.
BOOLEAN	Base type that stores one of three possible values used for logical calculations: TRUE, FALSE, or NULL.
BINARYJNTEGER	Base type for integers between -2,147,483,647 and 2,147,483,647.
PLSJNTEGER	Base type for signed integers between -2,147,483,647 and 2,147,483,647. PLS INTEGER values require less storage and are faster than NUMBER and BINARYJNTEGER values.

Bind Variables

A bind variable is a variable that you declare in a host environment and then use to pass runtime values, either number or character, into or out of one or more PL/SQL programs, which can use it as they would use any other variable. You can reference variables declared in the host or calling environment in PL/SQL statements, unless the statement is in a procedure, function, or package. This includes host language variables declared in pre-compiler programs, screen fields in Oracle Developer Forms applications, and SQL*Plus bind variables.

Creating Bind Variables

To declare a bind variable in the SQL*Plus environment, you use the command VARIABLE.

For example, you declare a variable of type NUMBER and VARCHAR2 as follows:

VARIABLE return_code NUMBER VARIABLE return_msg VARCHAR2(30)

Commenting Code

Prefix single-line comments with two dashes (--).

Place multi-line comments between the symbols/* and */.

Order of Operations

The operations within an expression are done in a particular order depending on their precedence.

The following table shows the default order of operations from top to bottom:

Operator	**Operation**
**,NOT	Exponentiation, logical negation
+5-	Identity, negation
V	Multiplication, division
+, -, ||	Addition, subtraction, concatenation
=!, =, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Comparison
AND	Conjunction
OR	Inclusion

Code Naming Conventions

The following table shows a set of prefixes and suffixes to distinguish identifiers from other identifiers, from database objects, and from other named objects.

Identifier	**Naming Convention**	**Example**
Variable	v *name*	v_sal
Constant	c *name*	c company name
Cursor	*name* cursor	emp_cursor
Exception	e *name*	e_too_many
Table type	«a»je_table_rype	amount_table_rype
Table	*name* table	order_total_table
Record type	«a»je_record_rype	emp_record_type
Record	*name* record	customer_record
SQL*Plus substitution variable (also referred to as substitution parameter)	p *name*	p_sal
SQL*Plus global variable (also referred to as host or bind variable)	g *name*	g_year_sal

SELECT Statements in PL/SQL

The INTO clause is required

Example

```
DECLARE
        v_deptno   NUMBER(2);
        v_loc      VARCHAR2(15);
BEGIN
SELECT deptno, loc  INTO  v_deptno, v_loc
FROM      dept
WHERE       dname  =  'SALES';
END;
```

INTO Clause

The INTO clause is mandatory and occurs between the SELECT and FROM clauses. It is used to specify the names of variables to hold the values that SQL returns from the SELECT clause. You must give one variable for each item selected, and their order must correspond to the items selected.You use the INTO clause to populate either PL/SQL variables or host variables. Queries Must Return One and Only One Row. PL/SQL deals with these errors by raising standard exceptions, which you can trap in the exception section of the block with the NO_DATA_FOUND and TOO_MANY_ROWS exceptions.

Manipulating Data Using PL/SQL

Make changes to database tables by using DML commands:

INSERT adds new rows of data to the table.

UPDATE modifies existing rows in the table.

DELETE removes unwanted rows from the table.

Inserting Data

Add new employee information to the EMP table.

Example

```
BEGIN

INSERT INTO emp(empno, ename, job, deptno)

VALUE S (empno_sequence. NEXTVAL, 'HARDING' 'CLERK', 10);

END;
```

Updating Data

Increase the salary of all employees in the EMP table who are Analysts.

Example

```
DECLARE
v_sal_increase    emp.sal%TYPE := 2000;
BEGIN
U P D A T E   e m p   S  E  T    sal + v_sal_increase
W H E R E   j o b 'ANALYST'";
END;
```

Deleting Data

Delete rows that belong to department 10 from the EMP table.

Example

```
DECLARE
v_deptno   emp.deptno%TYPE := 10;
BEGIN
DELETE FROM   emp
WHERE        deptno = v_deptno;
END;
```

Deleting Data

Delete in a specified order.

```
DECLARE
v_ordid  ord.ordid%TYPE := 605;
BEGIN
DELETE FROM  item   WHERE       ordid = v_ordid;
 END;
```

1.Create and execute a PL/SQL block that accepts 2 numbers through substitutions variables.

a. use the define command to provide 2 values
   define p_num1 = 2 define p_num2 = 4

b. Pass the 2values defined in step a above to the pl/sql block through substitution variables. The first number should be divided by the second number and add the second number added to the result.

2.Build a PL/SQL block that computes the total compensation (annual salary + annual bonus) for one year.

a. The annual salary and the annual bonus percentage values are defined using the DEFINE command.

b. Pass the values defined in the above step to the PL/SQL block through substitution variables.

The bonus must be converted from number to a decimal number. If salary or bonus is

null, set it to zero before computing the total compensation. (Hint: Use NVL function)

Syntax Of Commands Used :

1. DEFINE <*variable-name*>
2. VARIABLE *<variable_name> <datatype>*
3. NVL (expr1, expr2)

NVL function returns expr2 if expr1 is NULL. If expr1 is not null, then expr1 is returned.

Conclusions:

This experiment teaches the use of PL/SQL and non PL/SQL variables.

Lab Exercise:

1) Try to pass values to variables at run time.

2) Describe the structure of anonymous PL/SQL block.

3) What are host variables ?

4) What are bind variables ?

5) What are substitution variables ?

Practical No: 18

Title: To study various control structures available in PL/SQL.

IF Statements

There are three forms of IF statements:

IF-THEN-END IF

IF-THEN-ELSE-END IF

IF-THEN-ELSIF-END IF

1.1.2 Syntax : IF-THEN-END IF

IF *condition* THEN

statements;

[ELSIF *condition* THEN

statements ;

[ELSE

statements ;]

END IF;

IF-THEN-ELSE Statement Execution Flow

If the condition is FALSE or NULL, you can use the ELSE clause to carry out other actions.

Example:

```
IF condition THEN
    Statement1;
ELSE
Statement2;
END IF;
```

Nested IF Statements

Either set of actions of the result of the first IF statement can include further IF statements before specific actions are performed. The THEN and ELSE clauses can include IF statements. Each nested IF statement must be terminated with a corresponding END IF.

```
IF conditionl THEN

Statement1;

ELSE

    IF      condition2      THEN
        statement2;

    END IF;

END IF;
```

IF-THEN-ELSIF Statements

When possible, use the ELSIF clause instead of nesting IF statements. The code is easier to read and understand, and the logic is clearly identified. If the action in the ELSE clause consists purely of another IF statement, it is more convenient to use the ELSIF clause. This makes the code clearer by removing the need for nested END IPs at the end of each further set of conditions and actions.

Syntax

```
IF conditionl THEN

        statementl;

ELSIF condition2 THEN

        statement2;

ELSIF conditions THEN

        statements;

END IF;
```

FOR Loop

Syntax

FOR *counter* in [REVERSE]

lower_bound..upper_bound LOOP

statement!; statement2;

……..

END LOOP;

In the syntax

counter is an implicitly declared integer whose value automatically increases or decreases (decreases if the REVERSE keyword is used) by 1 on each iteration of the loop until the upper or lower bound is reached

REVERSE causes the counter to decrement with each iteration from the upper bound to the lower bound (Note that the lower bound is still referenced first.)

lower_bound specifies the lower bound for the range of counter values

upper_bound specifies the upper bound for the range of counter values

Do not declare the counter; it is declared implicitly as an integer.

Example

```
DECLARE
v_lower    NUMBER  :=  1;
v_upper    NUMBER  :=  100;
BEGIN
FOR  i  IN  v_lower..v_upper  LOOP
END  LOOP;
END;
```

WHILE Loop

You can use the WHILE loop to repeat a sequence of statements until the controlling condition is no longer TRUE. The condition is evaluated at the start of each iteration. The loop terminates when the condition is FALSE. If the condition is FALSE at the start of the loop, then no further iterations are performed.

Syntax

WHILE *condition* LOOP

statement!;

statement2;

END LOOP;

1. Write a PL/SQL block to satisfy the following conditions accepting the route_id as the user input.

A If the distance is less than 500 then update the fare to be 290.00.

B If the distance is between 501 – 1000 then update the fare to be 900.00.

C If the distance is greater than 1000 then display a message.

2. Write a PL/SQL block to display the ticket_no, route_id, fleet_id, total_fare when the total fare is greater than 1000.

3. (a) Write a PL/SQL block to display the reverse of numbers between 1 & 100.

(b) Write a loop control that will increment the value of an integer to 250 and exit from the loop and print the value of the integer on the standard output.

Conclusions:

This experiment teaches the use of various control structures.

Lab Exercise:

1.Write a loop control that will increment the vale of an integer to 250 and exit from the loop and print the value of the Integer on the standard output.

2. Try using CASE statement.

3. What are the other control statements available with PL/SQL.

4. How are NULL values handled.

5. Write a PL/SQL block to display the reverse of numbers between 1 and 25.

```
declare
  a number := 100;
begin
 loop
 a := a+25;
exit when a = 250;
end loop;
dbms_output.put_line(to_char(a));
end;
```

1. sql>set serveroutput on;

```
sql> declare

a number (7,2);

b number (4);

c number (5);

begin

c: = & temp;

select fare, distance into a,b from route_header

where route_id = c;

if b < 500 then

update route header set fare = 290 where route_id = c;

elsif b between 501 and 1000 then

update route header set fare = 900 where route_id = c;

elsif b > 1000 then

dbms_output.put_line('Error – distance greater then 1000');

end if;

commit

end;
```

2. declare

a number(5);

b number(5);

c number(5);

d number(7,2);

e number(5);

begin

e := &temp;

select ticket_no,route_id, fleet-id,total_fare into a,b,c,d from

ticket_header where ticket_no =e;

while d > 100 loop

dbms_output_put_line(a||' '||b||''||c||' '||d);

d :=99;

end loop;

end;

3. declare

a number(5);

b number(15);

c char(20);

d char(15);

e number(5);

begin

```
e := &temp
select place_id,place_name,place_address,bus_station
into a,b,c,d from place_header where place_id = e;
if e = '01' then
update place_header set bus_station = 'n' where place_id = e;
elsif e = '05' then
update place_header set bus_station = 'n' where place_id = e;
end if;
commit;
end;
```

```
declare
a number;
begin
for a in reverse 1..25 loop
dbms_output.put_line(to_char(a));
end loop;
end;
```

Practical No: 19

AIM: To study CURSORS and Exceptions.

Oracle allocates an area of memory known as context area for the processing of SQL statements. The context area contains information necessary to complete the processing, including the number of rows processed by the statement, a pointer to the parsed representation of the statement.

A cursor is a handle or pointer to the context area. Through the cursor, a PL/SQL program can control the context area what happens to it as the statement is processed.

Cursor Type	Description
Implicit	Implicit cursors are declared by PL/SQL implicitly for all DML and PL/SQL SELECT statements, including queries that return only one row.
Explicit	For queries that return more than one row. Explicit cursors are declared and named by the programmer and manipulated through specific statements in the block's executable actions.

Syntax:

There are 4 steps to be followed:

1) Declare a cursor

```
CURSOR cursor_name IS
        select_statement ;
```

2) Open the cursor

```
OPEN cursor_name ;
```

3) Fetch into the cursor.

```
FETCH cursor_name INTO [ variable1, variable2,……]
                    | record_name ] ;
```

4) Close the cursor.

```
CLOSE cursor_name ;
```

Explicit Cursor Attributes: -

Attribute	Type	Description
%ISOPEN	Boolean	Evaluates to true if cursor is open
%NOTFOUND	Boolean	Evaluates to true if the most recent fetch does not return a row
%FOUND	Boolean	Evaluates to true if the most recent fetch returns a row
%ROWCOUNT	Number	Evaluates to the total number of rows returned so far.

Sample Code Explaining Important Commands:

In the following example, emp_cursor is a explicit cursor used to retrieve the employee_id and last_name columns from the employees table into variables v_name, v_ename.

```
DECLARE
    v_empno employees.employee_id%type;
    v_ename employees.last_name%type;
    ……..

CURSOR emp_cursor IS
     select employee_id, last_name from employees;
```

```
BEGIN
    ……….
    open emp_cursor;
    loop
      fetch emp_cursor into v_empno, v_name ;
      exit when ….. ;
      …….
    end loop;
    ……….
    close emp_cursor ;
END;
```

2 Example 1: Retrieve the first 10 employees one by one.

```
DECLARE
v_empno employees. EMPLOYEE_ID%TYPE;
v_ename employees.first_name%TYPE;
CURSOR emp_cursor IS
SELECT  EMPLOYEE_ID, first_name
FROM   employees;
 BEGIN
OPEN emp_cursor;
 FOR i IN 1..10 LOOP
FETCH emp_cursor INTO v_empno, v_ename;
DBMS_OUTPUT.PUT_LINE(v_ename);
END LOOP;
END ;
```

Example 2: Retrieve the first 10 employees one by one.

```
DECLARE
v_empno   employees.EMPLOYEE_ID% TYPE;
 v_ename  employees.first_name% TYPE;
CURSOR  emp_cursor IS
SELECT  EMPLOYEE_ID, first_name
FROM employees;
BEGIN
OPEN  emp_cursor; LOOP
FETCH  emp_cursor  INTO v_empno,v_ename;
EXIT WHEN  emp_cursor%ROWCOUNT  >  10
OR  emp_cursor%NOTFOUND;
END   LOOP;
 CLOSE  emp_cursor;
 END  ;
```

Write a PL/SQL block to give details of passengers.

Write a PL/SQL block updating the nonstop to 'n' wher the route_ids are between 105 and115.

Title: To study EXCEPTION HANDLING in PL/SQL.

An exception is an identifier in PL/SQL, raised during the execution of a block that terminates its main body of actions. A block always terminates when PL/SQL raises an exception, but you specify an exception handler to perform final actions.

Exception Types

You can program for exceptions to avoid disruption at runtime. There are three types of exceptions.

Exception	Description	Directions for Handling
Predefined Oracle Server error	One of approximately 20 errors that occur most often in PL/SQL code	Do not declare and allow the Oracle Server to raise them implicitly
Non-predefined Oracle Server error	Any other standard Oracle Server error	Declare within the declarative section and allow the Oracle Server to raise them implicitly
User-defined error	A condition that the developer determines is abnormal.	Declare within the declarative section *and* raise explicitly

Syntax :

```
EXCEPTION
  WHEN exception1 [ OR exception2 . . . ] THEN
        statements;
  [ WHEN exception3 [ OR exception4 . . . ] THEN
        statements; ]
  [ WHEN OTHERS THEN
       statements; ]
```

In the syntax:

exception	is the standard name of a predefined exception or the name of a user-defined exception declared within the declarative section
statement	is one or more PL/SQL or SQL statements
OTHER	is an optional exception-handling clause that traps unspecified exceptions

Predefined Exceptions

Exception Name	Oracle Server Error Number	Description
ACCESS_INTO_NULL	ORA-06530	Attempted to assign values to the attributes of an uninitialized object
COLLECTION_IS_NULL	ORA-06531	Attempted to apply collection methods other than EXISTS to an uninitialized nested table or varray
CURSOR_ALREADY_OPEN	ORA-06511	Attempted to open an already open cursor
DUP_VAL_ON_INDEX	ORA-00001	Attempted to insert a duplicate value
INVALID_CURSOR	ORA-01001	Illegal cursor operation occurred
INVALID_NUMBER	ORA-01722	Conversion of character string to number fails
LOGIN_DENIED	ORA-01017	Logging on to Oracle with an invalid username or password
NO_DATA_FOUND	ORA-01403	Single row SELECT returned no data
NOT_LOGGED_ON	ORA-01012	PL/SQL program issues a database call without being connected to Oracle
PROGRAM_ERROR	ORA-06501	PL/SQL has an internal problem
ROWTYPE_MISMATCH	ORA-06504	Host cursor variable and PL/SQL cursor variable involved in an assignment have incompatible return types
STORAGE_ERROR	ORA-06500	PL/SQL ran out of memory or memory is corrupted
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Referenced a nested table or varray element using an index number larger than the number of elements in the collection
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Referenced a nested table or varray element using an index number that is outside the legal range (-1 for example)
TIMEOUT_ON_RESOURCE	ORA-00051	Time-out occurred while Oracle is waiting for a resource
TOO_MANY_ROWS	ORA-01422	Single-row SELECT returned more than one row
VALUE_ERROR	ORA-06502	Arithmetic, conversion, truncation, or size-constraint error occurred
ZERO_DIVIDE	ORA-01476	Attempted to divide by zero

Example:

```
ACCEPT p_deptno PROMPT 'Please enter the department number:
ACCEPT p_loc PROMPT 'Please enter the department location:

DECLARE
    e_invalid_dept    EXCEPTION;
    v_deptno     dept.deptno%TYPE := &p_deptno;

    BEGIN
    UPDATE dept
    SET    loc = '&p_loc'
    WHERE  deptno = v_deptno;
    IF SQL%NOTFOUND THEN raise
        e_invalid_dept;
    END IF;
   COMMIT;

EXCEPTION
    WHEN e_invalid_dept THEN
        :g_message := 'Department '|| TO_CHAR(v_deptno) ||
                                ' is an invalid department';
END;
```

Write a PL/SQL block that prints the number of employees who earn plus or minus 100 of the salary value set for an SQL*PLUS substitution variable. Use DEFINE to provide the salary value. Pass this value to the PL/SQL block.

(a) If there is no employee within that salary range, print a message to the user indicating that is the case. Use an exception for this case.
(b) If there are one or more employees within that range, the message should indicate how many employees are in that salary range.
(c) Handle any other exception with an appropriate exception handler. The message should indicate that some other error occurred.

CONCLUSIONS:

This experiment teaches the use of cursors, parameterized cursors and cursor attributes and how exceptions can be caught in pl/sql.

Lab Exercise:

1) What are implicit cursors and explicit cursors?

2) Try using the FOR UPDATE clause.

3) How are user defined exceptions trapped?

4) When is RAISE_APPLICATION_ERROR used.

Practical No.: 20

Title: To study PROCEDURES and FUNCTIONS in PL/SQL.

A procedure is a subprogram that performs a specific function.

Syntax:

CREATE [OR REPLACE] PROCEDURE *procedure_name*

[Parameter list]

IS | AS pl/sql block;

The parameter list can hold any of the following modes:

In parameter: This mode is used to pass the values to the subprograms when invoked.

Out Parameter: This mode is used to return values to the caller of the subprograms.

In Out Parameter: This mode is used to pass initial values to the sub program when invoked

& it also returns updated values to the caller.

Sample Code Explaining Important Commands:

In the example below, procedure add_dept takes two parameters i.e department name and location id from the user and inserts it into departments table.

```
CREATE OR REPLACE PROCEDURE add_dept
        (p_name IN departments.departmrnt_name%type DEFAULT 'unknown' ,
         p_loc IN departments.location_id%type DEFAULT 1700)
IS
BEGIN
   insert into departments(department_id,departments.name,departments.location_id)
   values (departmrnts_seq.nextval,p_name,p_loc);
END add_dept;
```

Q1. Write a procedure to accept the ticket_no as input and display an error message if the total_fare is null otherwise display the ticket_header.

Q2. Write a procedure to accept the route_id as input and update the capacity to be 25 if the cat_code is 1 and 50 if the cat_code is 2.(use route_header)

Q3 Create a procedure called UPD_JOB to modify a job in the JOBS table.

(d) Create a procedure called UPD_JOB to update the job title. Provide the job id and a new title, using two parameters. Include the necessary exception handling if no update occurs.
(e) Compile the code, invoke the procedure to change the job title of the job id IT_DBA to Data Administrator. Query the JOBS table to view the results.

Expected Output:

JOB_ID JOB_TITLE MIN_SALARY MAX_SALARY

IT_DBA Database Administrator

A function is a subprogram that computes a value.

Syntax:

```
CREATE OR [REPLACE FUNCTION] function_name
 [argument]   RETURN datatype
IS | AS
  pl/sql block;
```

where argument is in,out,inout.

Sample Code Explaining Important Commands:

In this example, get_sal is a function with one IN parameter and returns a number.

```
CREATE OR REPLACE FUNCTION get_sal
        ( p_id IN employees.employee_id%type)
        RETURN number
IS
        V_salary employees.salary%type := 0;
BEGIN
        select salary
        into v_salary
        from employees
        where employee_id = p_id ;
        return v_salary;
END get_sal;
```

Write a function, which will accept the ticket_no as the input and return total_fare as the output.(use ticket_header)

Write a PL/SQL block to create a function called ANNUAL_COMP to return the annual salary by accepting 2 parameters: an employee's monthly salary and commission. The function should address NULL values.

Create and invoke the function by passing both values. Even if one / both values are null, the function should still return an annual salary which is not null.

Use the formula: (salary * 12) + (commission_pct*12).

Expected Output:

EMPLOYEE_ID	LAST_NAME	ANNUAL COMPENSATION
145	Russell	235200
146	Partners	210600

. . . .

Conclusions:

This experiment teaches the use of procedures functions in pl/sql.

Lab Exercises And Review Questions:

1) What are the different modes in which you can pass parameters to a procedure?

2) How are exceptions handled in procedures.

3) Give some important features of subprogr

4) Create a stored function and call it in SQL statement.

5) When is a procedure generally used. When is a function generally used

Practical No :21

Title: Introduction to TRIGGERS

A database trigger is a stored procedure that is fired when an insert, update or delete statement is issued against the associated table. Database triggers can be used for following purposes:

- To generate data automatically.
- To enforce complex interiority constraints.
- To customize complex security authorizations.
- To maintain replicate tables.
- To audit data modifications.

Syntax :

CREATE [OR REPLACE] TRIGGER *trigger_name*

[before/after] [insert/update/delete]

ON *table_name [for each statement/for each row]*

[WHEN (condition)]

trigger_body

where

trigger_name: is the name of trigger

before/after: trigger is fired before or after executing the trigger statement.

for each statement/for each row: specifies that the trigger fires for each statement/once per row.

WHEN : specifies the trigger restriction.

trigger_body : is the trigger body that defines the action performed by the trigger, beginning with either DECLARE or BEGIN, ending with END, or a call to a procedure.

Sample Code Explaining Important Commands:

In the following example a trigger is created that allows only certain employees to earn a salary of more than 15,000.

```
CREATE OR REPLACE TRIGGER restrict_salary
   BEFORE INSERT OR UPDATE OF SALARY ON EMPLOYEES
     FOR EACH ROW
     BEGIN
          if not ( :NEW.job_id in ('AD_PRES', 'AD_VP')) and :NEW.salary > 15000 then
                raise_application_error ( -20202, 'Employee cannot earn this amount');
          end if;
END;
```

Write a PL/SQL block where Employees should receive a automatic increase in salary if minimum salary for a job is increased. Implement the requirement through a trigger on JOBS table.

(a) Create a stored procedure UPD_EMP _SAL to update the salary amount. This procedure accepts two parameters: the job ID for which the salary has to be updated and the new minimum salary. This procedure is executed from the trigger on the JOBS table.
(b) Create a row trigger named UPDATE_EMP_SALARY on the JOBS table that invokes the procedure UPD_EMP_SAL when the minimum salary in the JOBS table is updated for a specified job ID.

Expected Output:

(a) Query the EMPLOYEES table to see the current salary for employees who are programmers.

(b) Increase the minimum salary for the programmer job from 4,000 to 5,000.

(c) Employee LORENTZ had a salary of less than 4,500. Verify that her salary has been increased to the new minimum 5,000.

Conclusions:

This experiment teaches the use of row triggers.

Lab Exercises And Review Questions:

1) Describe different types of triggers
2) Describe database triggers and their use
3) Describe database firing rules.
4) How do you remove database triggers.

Practice Problems

1] Write an PL/SQL block to increase salary of employee by 10% if salary is >1500.

2] Write an PL/SQL block to calculate 15 odd elements of Fibonacci series.

3] Write an PL/SQL block to display names of those employees gating salary >2000

4] What is cursor? Explain types of cursor with example.

5] Write A program for update and insertion for bank transaction by using cursor.

6] What is an exception Explain exceptions with example

7] Write an PL/SQL block to calculate factorial of a no.

8] Explain functions and Procedures with example.

9] Write an trigger on update after updating sal col. For every employee.

10] Write an trigger on insert and update on table.

11] Explain structure of PL/SQL.

12] Write and explain any on inbuilt exception.

13] Explain different control statement of PL/SQL block.

14] Explain functions and procedures/

15] What is user define exception? Explain any one with example.

16] What is trigger? Explain types of trigger with example.