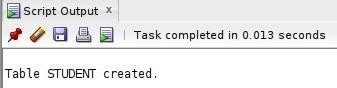
INDEX

|  |  |  |
| --- | --- | --- |
| **S.No** | **Program** | **Page No.** |
| 1 | Write the queries for Data Manipulation and Data  Definition Language. | 2 |
| 2 | Write SQL queries for single row functions and  general functions. | 4 |
| 3 | Write SQL queries for evaluating group functions. | 12 |
| 4 | Write SQL queries using group by clause. | 18 |
| 5 | Write SQL queries using constraints. | 19 |
| 6 | Write SQL queries to demonstrate joins and views  and subqueries. | 20 |
| 7 | Write programs by the use of PL/SQL | 26 |
| 8 | Write programs on procedures and functions | 42 |
| 9 | Write programs on cursors | 45 |
| 10 | Write programs on exception handling and triggers | 49 |



# EXPERIMENT-1: DDL AND DML STATEMENTS

**AIM:** Write the queries for Data Manipulation and Data Definition Language.

# THEORY:

**DDL:** A **data definition language** or **data description language** (**DDL**) is syntax similar to a computer programming language for defining data structures, especially database schemas.-

Commands in DDL are:

1. CREATE
2. DROP
3. RENAME
4. ALTER

# DDL COMMANDS:

**SYNTAX:**

CREATE Statement: CREATE TABLE tablename (attribute\_1 data type, attribute\_2 data type, …… attribute\_n data type);

DROP Statement:DROP TABLE table\_name;

RENAME Statement:RENAME table\_name to new\_name; ALTER Statement:

Add column to Table: ALTER TABLE table\_name ADD column\_name

column-definition;

Modify column in Table: ALTER TABLE table\_name MODIFY column\_name column\_type;

Drop column in Table: ALTER TABLE table\_name DROP COLUMN column\_name;

# DDL QUERIES:

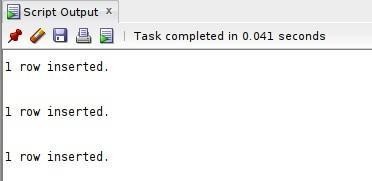
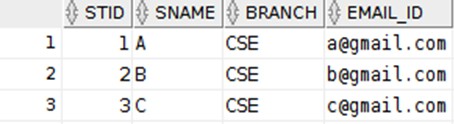
**Q1.Write a query to create a table student1 with sid, sname, and ssec.**

create table student1

( sid varchar(10),sname varchar(20),ssec varchar(4) );

# Output:

Q2. Write a query to insert 5 rows

insert into Student values(1,'A','CSE','a@gmail.com.in'); insert into Student values(2,'B','CSE','b@gmail.com.in''); insert into Student values(3,'C','CSE','c@gmail.com.in');

Aim: Write a query to delete the tuple Code:

delete from Student where name='Unknown' ;

Output:



**Experiment No.** 2 Single row functions

**AIM:** Write the queries for evaluating single row functions.

# THEORY:

Single row functions are the one who work on single row and return one output per row. For example, length and case conversion functions are single row functions. Single row functions can be character functions, numeric functions, date functions, and conversion functions. Note that these functions are used to manipulate data items. These functions require one or more input arguments and operate on each row, thereby returning one output value for each row. Argument can be a column, literal or an expression. Single row functions can be used in SELECT statement, WHERE and ORDER BY clause.

Single row functions are of five types:

1. Character Functions
2. Number Functions
3. Date Functions
4. Conversion functions
5. General Functions

# Character Functions:

These are subdivided into – Case Manipulation Functions and Character Manipulation Functions.

Case Manipulation Functions - Accepts character input and returns a character value. Functions under the category are UPPER, LOWER and INITCAP.

* UPPER function converts a string to upper case.
* LOWER function converts a string to lower case.
* INITCAP function converts only the initial alphabets of a string to upper case.

Character Manipulation Functions – Accepts character input and returns number or character value. Functions under the category are CONCAT, LENGTH, SUBSTR, INSTR, LPAD, RPAD, TRIM and REPLACE.

* CONCAT function concatenates two string values.
* LENGTH function returns the length of the input string.
* SUBSTR function returns a portion of a string from a given start point to an end point.
* INSTR function returns numeric position of a character or a string in a given string.
* LPAD and RPAD functions pad the given string upto a specific length with a given character.
* TRIM function trims the string input from the start or end.
* REPLACE function replaces characters from the input string with a given character.

# Number Functions :

Accepts numeric input and returns numeric values. Functions under the category are ROUND, TRUNC, and MOD.

* ROUND and TRUNC functions are used to round and truncate the number value.
* MOD is used to return the remainder of the division operation between two numbers.

# Date Functions :

Date arithmetic operations return date or numeric values. Functions under the category are MONTHS\_BETWEEN, ADD\_MONTHS, NEXT\_DAY, LAST\_DAY, ROUND

and TRUNC.

* MONTHS\_BETWEEN function returns the count of months between the two dates.
* ADD\_MONTHS function add 'n' number of months to an input date.
* NEXT\_DAY function returns the next day of the date specified.
* LAST\_DAY function returns last day of the month of the input date.
* ROUND and TRUNC functions are used to round and truncates the date value.

# Conversion Functions :

Type conversion can be either implicitly done by Oracle or explicitly done by the programmer.

Implicit Type Conversion – A VARCHAR2 or CHAR value can be implicitly converted to NUMBER or DATE type value by Oracle. Similarly, a NUMBER or DATA type value can be automatically converted to character data by Oracle server. Note that the impicit interconversion happens only when the character represents the a valid number or date type value respectively.

Explicit Type Conversion – SQL Conversion functions are single row functions which are capable of typecasting column value, literal or an expression . TO\_CHAR, TO\_NUMBER and TO\_DATE are the three functions which perform cross modification of data types.

* TO\_CHAR function is used to typecast a numeric or date input to character type with a format model (optional). Dates can be formatted in multiple formats after converting to character types using TO\_CHAR function. Example Formats for dates :

|  |  |
| --- | --- |
| **Format Model** | **Description** |
| YYYY | Full year in number |
| Year | Year spelled out |
| MM | Two digit value for month |
| Month | Full name of month |
| DD | Numeric day of month |
| DAY | Full name of day the week |
| TH | Produces ordinal number |
| SP | Spell out the number. |
| SPTH | Spell out the ordinal number. |

Example formats for Numbers:

|  |  |
| --- | --- |
| **Form at Mode l** | **Description** |
| , | It returns a comma in the specified position. You can specify multiple commas in a number format model. |
| . | Returns a decimal point, in the specified position. |

|  |  |
| --- | --- |
| $ | Returns value with a leading dollar sign |
| 0 | Returns leading zeros. |
| 9 | Returns value with the specified number of digits |
| L | It is used for local currency symbol |

* The TO\_NUMBER function converts a character value to a numeric datatype. If the string being converted contains nonnumeric characters, the function returns an error.
* The function takes character values as input and returns formatted date equivalent of the same. The TO\_DATE function allows users to enter a date in any format, and then it converts the entry into the default format used by Oracle server.

# General Functions :

General functions are used to handle NULL values in database. The objective of the general NULL handling functions is to replace the NULL values with an alternate value. We shall briefly see through these functions below.

* NVL – The NVL function substitutes an alternate value for a NULL value. NVL(arg, replace\_with)
* NVL2 – NVL2 function can be used to substitute an alternate value for NULL as well as non NULL value. NVL2(arg, value\_if\_not\_null,value\_if\_null);
* NULLIF – The NULLIF function compares two arguments expr1 and expr2. If expr1 and expr2 are equal, it returns NULL; else, it returns expr1. Unlike the other null handling function, first argument can't be NULL. NULLIF(exp1,exp2)
* COALESCE function, a more generic form of NVL, returns the first non-null expression in the argument list. It takes minimum two mandatory parameters but maximum arguments has no limits. COALESCE (exp1,exp2,……,expn)

Aim: Write a Query to demonstrate character manipulation function Query:

select concat('Just',' like that') as Moment from dual

select length('length') as len from dual

select inStr('instring','s') as occurenceofs from dual

select inStr('Hello','l',2) as occurcenceofinStr from dual

select substr('Hello Hyderabad',2,8) from dual

select lpad('3500',10,'$')

from dual

select rpad('3500',10,'$')

from dual Output:







1. Aim: Write a Query to demonstrate number function Query:

select round(99.9987654,6)

from dual

select round(99.9987654,3)

from dual

select round(99.9987654,1)

from dual

select round(99.9987654,-1)

from dual

select round(99.9987654,0)

from dual

select trunc(99.9987654,6)

from dual

select trunc(99.9987654,3)

from dual

select trunc(99.9987654,1)

from dual

select trunc(99.9987654,-1)

from dual

select trunc(99.9987654,0)

from dual

Output:



1. Aim: Write a Query to demonstrate character manipulation function Query:

select concat('Just',' like that') as Moment from dual

select length('length') as len from dual

select inStr('instring','s') as occurenceofs from dual

select inStr('Hello','l',2) as occurcenceofinStr from dual

select substr('Hello Hyderabad',2,8) from dual

select lpad('3500',10,'$')

from dual

select

rpad('3500',10,'$')

from dual Output:







1. Aim: Write a Query to demonstrate number function Query:

select round(99.9987654,6)

from dual

select round(99.9987654,3)

from dual

select round(99.9987654,1)

from dual

select round(99.9987654,-1)

from dual

select round(99.9987654,0)

from dual

select trunc(99.9987654,6)

from dual

select trunc(99.9987654,3)

from dual

select trunc(99.9987654,1)

from dual

select trunc(99.9987654,-1)

from dual

select trunc(99.9987654,0)

from dual

Output:









160121733034

# Experiment 3:

**AIM:** Write the queries for evaluating group functions.

# THEORY:

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

* COUNT (): This function returns the number of rows in the table that satisfies the condition specified in the WHERE condition. If the WHERE condition is not specified, then the query returns the total number of rows in the table.
* MAX(): This function is used to get the maximum value from a column.
* MIN(): This function is used to get the minimum value from a column.
* AVG(): This function is used to get the averagevalue of a numeric column.
* SUM(): This function is used to get the sum of a numeric column

Aim:

Write a query to find out the maximum of grades Query:

select min(all

grade) from customer;

Output:



Query-5:

Aim:

Write a query to find out the count of grades Query:

select count(all

grade) from customer;

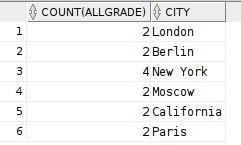
Output:



Query-6:

Aim:Write a query to demonstrate groupby class. Query:

select count(all grade),city from customer

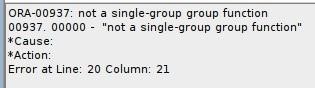
group by(city); Output:

Query-7:

Aim:Write a query to demonstrate group function without using groupby class Query:

select count(all grade),city from customer;

Output:



Query-8:

Aim:Write a query to demonstrate having class Query:

select count(all grade),city from customer

group by(city)

having city in 'New York';

Output:



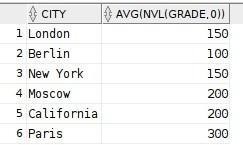
Query-9:

Aim:Write a query to demonstrate nesting

function. Query:

select city,avg(NVL(grade,0)) from customer

group by city;

Output:

# Query:

select add\_months('20-02-2022',5) from dual;

# Output:



**Query:**

select add\_months(sysdate,5) from dual;

# Output:



**Query:**

select months\_between('12-10-2022','01-10-2020') from dual;

# Output:

**Query:**

select next\_day(sysdate,'SUN') from dual;

# Output:



**Query:**

select last\_day(sysdate) from dual;

# Output:



**Query:**

select round(sysdate,'MON') from dual;

# Output:



**Query:**

select round(sysdate,'YEAR') from dual;

# Output:

**Query:**

select trunc(sysdate,'MON') from dual;

# Output:

**Query:**

select trunc(sysdate,'YEAR') from dual;

# Output:



**5. Conversion Functions Query:**

select to\_char(sysdate,'DD-MON-YYYY') from dual;

# Output:

**Query:**

select to\_char(5000,'$09,999') from dual;

# Output:



**Query:**

select to\_number('5999') from dual;

# Output:

**Query:**

select to\_number('$12,999','$99,999') from dual;

# Output:

**Query:**

select to\_date('20-10-2020') from dual;

# Output:

**Query:**

select to\_date('20,OCT-2020','DD,MON-YYYY') from dual;

# Output:

**Experiment No.** 4 Sub-Queries

**AIM:** Write the SQL queries using‘groupby’ function.

# THEORY:

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause. We know that WHERE clause is used to place conditions on columns but what if we want to place conditions on groupsthe HAVING clause comes into use. We can use HAVING clause to place conditions to decide which group will be the part of final result-set. Also we can not use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So we have to use HAVING clause if we want to use any of these functions in the conditions.

# SYNTAX:

SELECT column\_name(s) FROM table\_name WHERE condition

GROUP BY column\_name(s) ORDER BY column\_name(s);

# QUERIES:

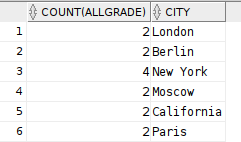
**Q1:**Write a query to demonstrate groupby class.

# Query:

select count(all grade),city from customer

group by(city);

# Output:

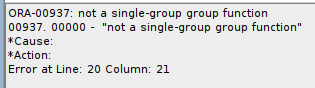


**Q2:**Write a query to demonstrate group function without using groupby class

# Query:

select count(all grade),city from customer;

# Output:



**Experiment No.** 5 Sub-Queries (constraints)

**Q3:**Write a query to demonstrate having class

# Query:

select count(all grade),city from customer

group by(city)

having city in 'New York';

# Output:



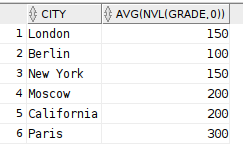
**Q4:**Write a query to demonstrate nesting function.

# Query:

select city,avg(NVL(grade,0)) from customer

group by city;

# Output:



**Experiment No.** 6 Joins ,Views

**AIM:** Write the queries to demonstrate various joins and views in SQL.

# THEORY:

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Here are the different types of the JOINs in SQL:

* CROSS JOIN:Returns all rows in first table joined to all rows in second table.
* (INNER) JOIN: Returns records that have matching values in both tables
* LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table
* RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table
* FULL (OUTER) JOIN: Return all records when there is a match in either left or right table
* EQUI JOIN: Returns records based on the equality condition specified.
* NATURAL JOIN: Automatically joins the matching columns in two tables and retuens matching records.

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following −

* Structure data in a way that users or classes of users find natural or intuitive.
* Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

Joins:

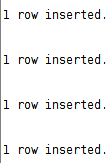
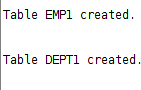
table for joins operations: Query:

CREATE table emp1( eid number(3),

ename varchar(20), did number(3));

CREATE table dept1( did number(3), dname varchar(20));

Output:



1. Equijoin:

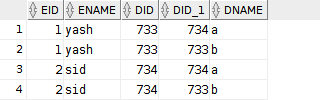
Query:

select \* from emp1,dept1 where emp1.did=dept1.did; Output:

Query:

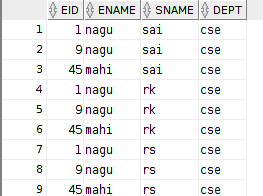
select \* from emp1,dept1; Output:

1. Crossjoin:



sid int,

Table for Joins: create table student(



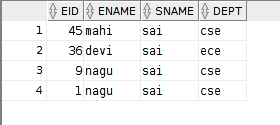
sname varchar(6), sdept varchar(6)); create table emp( eid int,

ename varchar(6), dept varchar(6));

insert into student values('1','sai','cse'); insert into student values('51','rk','cse'); insert into student values('72','rs','cse'); insert into emp values('45','mahi','cse'); insert into emp values('36','devi','ece'); insert into emp values('1','nagu','cse');

1. non equal join:

query1: select eid,ename,sname,dept from emp,student where sid between 1 and 10;

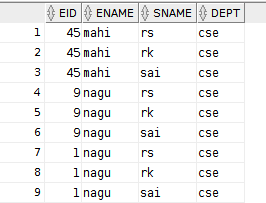
output1:

1. outerjoin:

query1: select

eid,ename,sname,dept from emp,student where emp.dept(+)=student.sdept;

Output1:

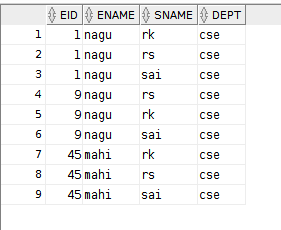
query2: select  eid,ename,sname,dept from emp,student where

emp.dept=student.sdept(+); Output2:



1. innerjoin:

query1: select eid,ename,sname,dept from emp,student where student.sdept=emp.dept order by eid;

Output1:

cross jojn:

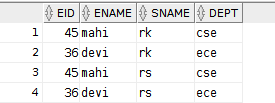
query1:-- insert into student values('51','rk','cse');

-- insert into student values('72','rs','cse');

-- insert into emp values('45','mahi','cse');

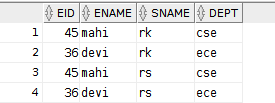
-- insert into emp values('36','devi','ece'); delete from emp where ename='nagu';

select eid,ename,sname,dept from emp cross join

student; Output1:

Natural Join

query1:select eid,ename,sname,dept from emp natural join student;

Output1:

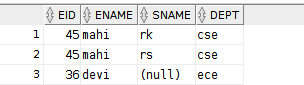
On class:

query1:select eid,ename,sname,dept from emp join student on(emp.dept=student.sdept);

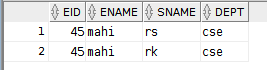
Output1:

leftouter join:

query1:select eid,ename,sname,dept from emp left outer join student on(emp.dept=student.sdept);

Output1:

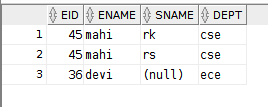
rightOuterjoin:

query1:select eid,ename,sname,dept from emp right outer join student on(emp.dept=student.sdept);

Output1:

FullOuterjoin:

query1:select eid,ename,sname,dept from emp full outer join student on(emp.dept=student.sdept);

Output1:

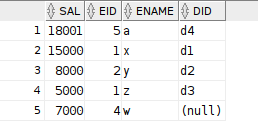
Views:

1. **Aim:** Create a view to display all employees.

# Code:

create or replace view emp\_d1 as

select \* from emp1



# Output:

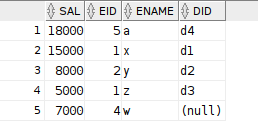


1. **Aim:**Inserting into created view

# Code:

insert into emp\_d1 values(18000,5,'a','d4');

# Output:



1. **Aim:**Updating the created view.

# Code:

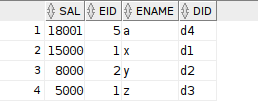
update emp\_d1 set sal=18001 where did='d4';

# Output:

1. **Aim:**Deleting the view.

# Code:

delete from emp\_d1 where did is null;



# Aim:

Inserting the data into view that contain groupfn.

# Code:

create or replace view emp\_grpfn as select min(sal) as Minimum\_S from emp1;

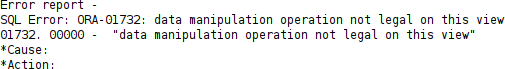
select \*

from emp\_grpfn;

insert into emp\_grpfn values(18999);

# Output:





**Q3.SUB-QUERIES:**

# AIM:

From the salesman and orders tables, find all the orders issued by

the salesman 'Paul Adam', Return ord\_no, purch\_amt, ord\_date, customer\_id and salesman\_id.

# CODE:

SELECT \*

FROM orders

WHERE salesman\_id = (SELECT salesman\_id FROM SALESPEOPLE

WHERE name='paul adam');

# OUTPUT:



1. **AIM:**

From the salesman and orders tables, find all the orders, which are generated by those salespeople, who live in the city of London.Return ord\_no, purch\_amt, ord\_date, customer\_id, salesman\_id.

# CODE:

select \* from orders

where SALESMAN\_ID=(select salesman\_id from SALESPEOPLE

where city='london');

# OUTPUT:



1. **AIM:**

From the orders table, find the orders generated by the salespeople who works for customers whose id is 3007. Return ord\_no, purch\_amt, ord\_date, customer\_id, salesman\_id. A customer can works only with a salespeople.

# CODE:

SELECT \*

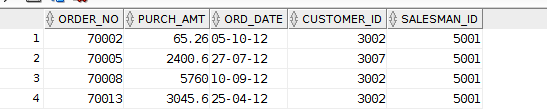
FROM orders

WHERE salesman\_id =

(SELECT DISTINCT salesman\_id FROM orders

WHERE customer\_id =3007);

# OUTPUT:-



1. **AIM:**

From the orders table, find the order values greater than the average order value of 10th October 2012. Return ord\_no, purch\_amt, ord\_date, customer\_id, salesman\_id.

# CODE:

SELECT \*

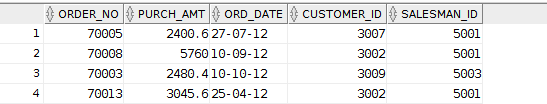
FROM orders WHERE purch\_amt >

(SELECT AVG(purch\_amt)

FROM orders

WHERE ord\_date ='10-Oct-2012');

# OUTPUT:



1. **AIM:**

From the salesman and orders tables, find all the orders generated in New York city. Return ord\_no, purch\_amt, ord\_date, customer\_id and salesman\_id.

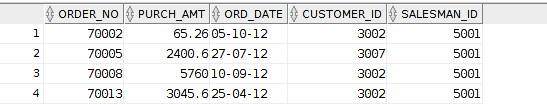
# CODE:

select \* from orders

where SALESMAN\_ID=(select salesman\_id from SALESPEOPLE

where city='new york');

# OUTPUT:



1. **AIM:**

From the customer and salesman tables, find the commission of the salespeople work in Paris City. Return commission.

# CODE:

select commision from SALESPEOPLE

where SALESMAN\_ID in (select salesman\_id from CUSTOMER

where city='Paris');

# OUTPUT:



1. **AIM:**

Write a query to display all the customers whose id is 2001 bellow the salesman ID of Mc Lyon.

# CODE:

SELECT \*

FROM customer WHERE customer\_id =

(SELECT salesman\_id -2001 FROM SALESPEOPLE

WHERE name = 'mc lyon');

# OUTPUT:



1. **AIM:**

From the customer table, count number of customers with grades above the average grades of New York City. Return grade and count.

# CODE:

SELECT grade, COUNT (\*) FROM customer

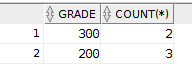
GROUP BY grade HAVING grade >

(SELECT AVG(grade)

FROM customer

WHERE city = 'New York');

# OUTPUT:



1. **AIM:**

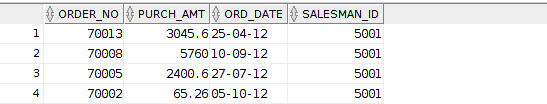
From the salesman and orders tables, find those salespeople who earned the maximum commission. Return ord\_no, purch\_amt, ord\_date, and salesman\_id.

# CODE:

SELECT order\_no, purch\_amt, ord\_date, salesman\_id FROM orders

WHERE salesman\_id IN( SELECT salesman\_id FROM salespeople WHERE commision = ( SELECT MAX(commision) FROM salespeople));

# OUTPUT:



1. **AIM:**

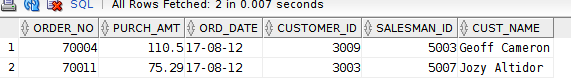
From the customer and orders tables, find the customers whose orders issued on 17th August, 2012. Return ord\_no, purch\_amt, ord\_date, customer\_id, salesman\_id and cust\_name.

# CODE:

SELECT b.\*, a.cust\_name FROM orders b, customer a

WHERE a.customer\_id=b.customer\_id AND b.ord\_date='17-Aug-2012';

# OUTPUT:

 11.

# AIM:

From the customer and salesman tables, find the salespeople who had more than one customer. Return salesman\_id and name.

# CODE:

SELECT salesman\_id,name FROM SALESPEOPLE a WHERE 1 <

(SELECT COUNT(\*)

FROM customer

WHERE salesman\_id=a.salesman\_id);

# OUTPUT:

 12.

# AIM:

From the orders table, find those orders, which are equal or higher than average

amount of the orders. Return ord\_no, purch\_amt, ord\_date, customer\_id and salesman\_id.

# CODE:

select \* from orders

where PURCH\_AMT>=( select avg(purch\_amt) from orders);

# OUTPUT:

13.

# AIM:

From the customer and salesman tables, find the salespeople who deal a single customer. Return salesman\_id, name, city and commission.

# CODE:

SELECT \*

FROM SALESPEOPLE a WHERE 1 =

(SELECT COUNT(\*)

FROM customer

WHERE salesman\_id=a.salesman\_id);

# OUTPUT:

14.

# AIM:

From the salesman and orders tables, find the salespeople who deal the customers with more than one order. Return salesman\_id, name, city and commission.

# CODE:

SELECT \*

FROM SALESPEOPLE a WHERE 1 <

(SELECT COUNT(\*) FROM ORDERS

WHERE salesman\_id=a.salesman\_id);

# OUTPUT:

15.

# AIM:

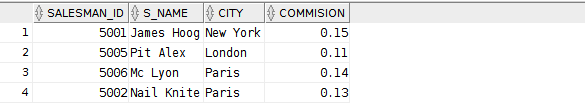
From the customer and salesman tables, find the salespeople who deals those customers who live in the same city. Return salesman\_id, name, city and commission.

# CODE:

select \* from sales

where city in (select city from customer);

# OUTPUT:

 16.

# AIM:

From the customer and salesman tables, find the

salespeople whose place of living (city) matches with any of the city where customers live. Return salesman\_id, name, city and commission.

# CODE:

select \* from sales

where city in (select city from customer);

# OUTPUT:

17.

# AIM:

From the orders table, find all those orders whose order amount greater than at least one of the orders of September 10th 2012. Return ord\_no, purch\_amt, ord\_date, customer\_id and salesman\_id.

# CODE:

SELECT \*

FROM orders

WHERE purch\_amt >ANY (SELECT purch\_amt FROM orders

WHERE ord\_date ='10-Sep-2012');

# OUTPUT:

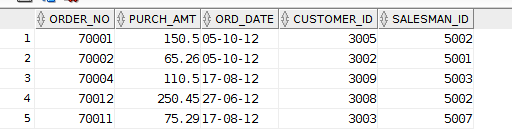
18.

# AIM:

From the customer and orders tables, find those orders where an order amount less than any order amount of a customer lives in London City. Return ord\_no, purch\_amt, ord\_date, customer\_id and salesman\_id.

# CODE:

select \*

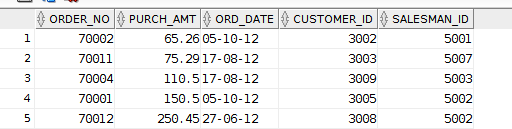


from orders

where PURCH\_AMT <ANY (SELECT purch\_amt FROM orders a, customer b

WHERE a.customer\_id=b.customer\_id AND b.city='London');

# OUTPUT:



1. **AIM:**

From the customer and orders tables, find those orders where every order amount less than the maximum order amount of a customer lives in London City. Return ord\_no, purch\_amt, ord\_date, customer\_id and salesman\_id.

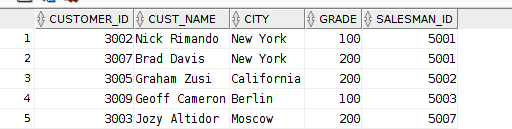
# CODE:

SELECT \*

FROM orders WHERE purch\_amt <

(SELECT MAX (purch\_amt) FROM orders a, customer b

WHERE a.customer\_id=b.customer\_id AND b.city='London');



# AIM:

From the customer table, find those customers whose grade are higher than customers living in New York City. Return customer\_id, cust\_name, city, grade and salesman\_id.

# CODE:

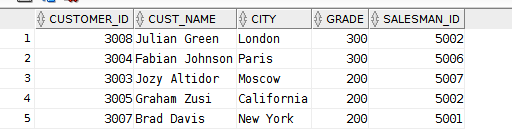
select \*

from CUSTOMER

where grade>ANY ( select grade from customer

where city='New York');

# OUTPUT:



1. **AIM:**

From the customer table, find those customers whose grade doesn't same of those customers live in London City. Return customer\_id, cust\_name, city, grade and salesman\_id.

# CODE:

select \*

from CUSTOMER

where grade<>ANY ( select grade from customer

where city='London');

# AIM:

From the customer table, find those customers whose grade are not same of those customers living in Paris. Return customer\_id, cust\_name, city, grade and salesman\_id.

# CODE:

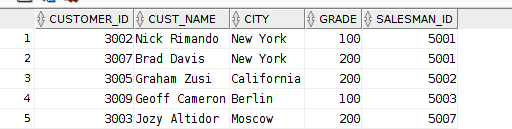
select \*

from CUSTOMER

where grade<>ANY ( select grade from customer

where city='Paris');

# OUTPUT:



1. **AIM:**

From the employee department and employee details tables, find those employees who work for the department where the department allotment amount is more than Rs. 50000. Return emp\_fname and emp\_Iname.

# CODE:

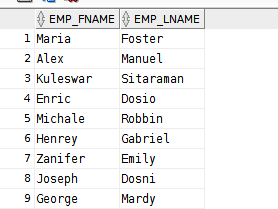
SELECT emp\_fname, emp\_lname FROM emp\_details

WHERE emp\_dept IN (SELECT dpt\_code

FROM emp\_dept

WHERE dpt\_allotment > 50000);

# OUTPUT:



1. **AIM:**

From the employee department table, find the departments where the sanction amount is higher than the average sanction amount of all the departments. Return dpt\_code, dpt\_name and dpt\_allotment.

# CODE:

SELECT \*

FROM emp\_dept WHERE dpt\_allotment > (

SELECT AVG(dpt\_allotment) FROM emp\_dept

);

# OUTPUT:



1. **AIM:**

From the employee department and employee details tables, find the departments where more than two employees work. Return dpt\_name.

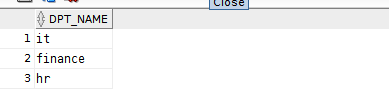
# CODE:

select dpt\_name from emp\_dept

where dpt\_code in ( select emp\_dept from EMP\_DETAILS

group by emp\_dept having count(\*)>2);

# OUTPUT:



**Experiment:7 Pl/sql**

**AIM:** write the queries for cursors and in pl/sql

# THEORY:

A place where we can have active set of commands.It allows you to name a work area and access its stored information.There are two types of cursors: -

o Implicit cursors

**o** Explicit cursors

# Implicit cursors: -

**Explicit cursors: -**

* Defined by the oracle server.
* They don’t have any name.
* It releases the memory allocated when the SQL statements execution is completed.
* In this we can’t use open, fetch and close commands.

Declaring the cursor:

* It is defined by the user.
* It is used by the user to process multiple rows returned by a select statement.
* Steps for defining an explicit cursor:
  + Declare the cursor.
  + Open the cursor.
  + Fetch data from the cursor.
  + Close the cursor.

SynTax: Cursor <cursor name> is SQL statement.

Example: cursor c1 is

Select \* from emp where sid between 100 and 110;

* + - Opening the cursor includes: Allocating memory

Parse the SQL statement for execution.

* + - The rows which are selected and stored in an Active Set and this active set are stored in the memory. The active set always points to the first row in the active set.
    - Fetch: To retrieve data from the active set and is stored in the variables.

E.g.: fetch ename into a;

* + - After the fetch instruction the active set is checked. If it is empty it enables the close else it moves the cursor to the next row in the active set and continues until the last row is reached. Once the last row is reaches it enables the close.

# Queries:

* So, close will be enabled in two cases: o When active set is empty

o When active set is pointing to the last row in the active set.

SynTax: close <cursor name>;

**Q1:**Write a query to demonstrate Implicit Cursor.

# Code:

set SERVEROUTPUT ON; DECLARE

v\_num\_rows NUMBER; BEGIN

-- Update salaries of employees UPDATE employees

SET salary = salary \* 1.1 WHERE department\_id = 10;

-- Get the number of updated rows v\_num\_rows := SQL%ROWCOUNT;

-- Print the number of updated rows DBMS\_OUTPUT.PUT\_LINE('Number of updated rows: ' || v\_num\_rows);

END;

# Output:

**Q2.**Write a query to show the use of %notfound attribute. i.Declare

X integer (10);

A char (10);

Begin

Name varchar2 (20);

For I in 140.. 149

Loop

End;

Select first\_name into name from employees where employee\_id=i; End loop;

X: =sql%rowcount;

DBMS\_OUTPUT.PUT\_LINE ('no. of rows selected: '||x);

# Output:

No. of rows selected: 1

Statement processed.

ii. declare

vname employees.ename %type; vsal employees.salary %type;

begin

end;

select ename,salary into vname,vsal from employees where emp\_id=100; dbms\_output.put\_line('No of rows effected '||to\_char(sql%rowcount));

# Output:

No of rows effected 1

PL/SQL procedure successfully completed.

# Q3.Explicit cursor [pl/sql block to retrieve 10 employees’ details using cursor].

Declare

Vname employees.ename %type; Vjid employees.job\_id %type;

Cursor emp\_det is

select ename,job\_id into vname,vjid from employees where emp\_id between 100 and 175;

begin

open emp\_det ; for i in 1..10 loop

end;

fetch emp\_det into vname,vjid ;

dbms\_output.put\_line (i||'.Name is : '||vname||',Job id is : '||vjid); end loop;

close emp\_det;

# Output:

1.Name is : king, Job id is : ad\_pres 2.Name is : kochhar, Job id is : ad\_vp 3.Name is : de haan, Job id is : ad\_vp 4.Name is : hunold, Job id is : it\_prog 5.Name is : emst, Job id is : it\_prog 6.Name is : mourgos, Job id is : st\_man 7.Name is : rajs, Job id is : st\_clk 8.Name is : davies, Job id is : st\_clk 9.Name is : zlotkey, Job id is : sa\_man 10.Name is : able, Job id is : sa\_rep

PL/SQL procedure successfully completed.

# Q4.Write a pl/sql block using %rowcount attribute in explicit cursor:

Declare

vname employees.ename %type; vjid employees.job\_id %type;

cursor emp\_det is

select ename,job\_id into vname,vjid from employees where emp\_id=174 ;

begin

end;

open emp\_det ;

fetch emp\_det into vname,vjid; dbms\_output.put\_line(to\_char(emp\_det%rowcount)); close emp\_det;

# Output:

1

PL/SQL procedure successfully completed.

# Q5.Write a pl/sql block using % not found attribute in explicit cursor:

declare

vname employees.ename %type; vjid employees.job\_id %type;

cursor emp\_det is

select ename,job\_id into vname,vjid from employees where emp\_id=174 ;

begin

open emp\_det ;

fetch emp\_det into vname,vjid; if emp\_det%notfound then

dbms\_output.put\_line('NOT found is true');

else

dbms\_output.put\_line('NOT found is false');

end if;

close emp\_det;

end;

# Output:

NOT found is false

PL/SQL procedure successfully completed.

# Q6.Write a pl/sql block using %found attribute in explicit cursor

declare

vname employees.ename %type; vjid employees.job\_id %type;

cursor emp\_det is

select ename,job\_id into vname,vjid from employees where emp\_id=174 ;

begin

open emp\_det ;

fetch emp\_det into vname,vjid; if emp\_det%found then

dbms\_output.put\_line('found is true');

else

dbms\_output.put\_line(' found is false');

end;

end if;

close emp\_det;

# Output:

found is true

PL/SQL procedure successfully completed.

# Experiment8: Procedures and functions

**AIM:** write the queries for procedures and functions in pl/sql

# THEORY:

**PROCEDURES:**

A procedure is a program that performs a specific action.A procedure has two parts: specification and body.Procedure specification begins with the keyword ‘PROCEDURE’ and ends with the procedure ‘name’ or a ’parameter list’.

The procedure body begins with the keyword ‘IS’ and ends with the keyword ‘END’. The procedure body has three parts a declarative part, an executable part and an optional exception handling part.

**Syntax for creating a procedure:-** Procedure name [(parameter list)] [Local declaration]

Begin

Executable statements Exception

Exception handlers End [<procedure name>] **Calling a procedure:-**

Procedure\_name (parameter list);

# FUNCTIONS:

A function is a sub program that computes a value.They have a return clause.A function has two parts: specification and body.Function specification begins with the keyword ‘FUNCTION’ and ends with the ’return clause’, which specifies the data type of the return value.The function body begins with the keyword ‘IS’ and ends with the keyword ‘END’.The function body has three parts a declarative part, an executable part and an optional exception handling part.The return statement immediately ends the execution of a sub program and returns control to the caller.Execution continues with the statement following the sub program call.

# Syntax for creating a function:-

Function <function name> [arguments..]

Return data type is [(Local declaration)] Begin

Executable statements Exception

Exception handlers End [<function name>]

# QUERIES:

**Q1.**Write a query to demonstrate Procedures.

# Code:

set SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE print\_employee\_salary (employee\_id IN employees.employee\_id%TYPE) AS

v\_employee\_salary employees.salary%TYPE; BEGIN

-- Retrieve the salary of the employee SELECT salary INTO v\_employee\_salary FROM employees

WHERE employee\_id = employee\_id;

-- Print the employee ID and salary

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || employee\_id || ', Salary: ' || v\_employee\_salary);

END;

/

# Output:



**Q2:**Write a query to demonstrate Functions.

# Code:

set SERVEROUTPUT ON;

CREATE OR REPLACE FUNCTION calculate\_bonus (salary IN NUMBER) RETURN NUMBER AS

v\_bonus NUMBER; BEGIN

-- Calculate bonus based on salary IF salary > 5000 THEN

v\_bonus := salary \* 0.1; -- 10% bonus ELSE

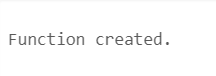
v\_bonus := salary \* 0.05; -- 5% bonus END IF;

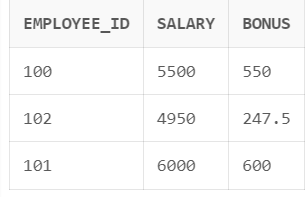
RETURN v\_bonus; END;

/

SELECT employee\_id, salary, calculate\_bonus(salary) AS bonus FROM employees;

# Output:





**Experiment No. 9 Cursors in pl/sql**

**AIM:** write the queries for cursors and in pl/sql

# THEORY:

A place where we can have active set of commands.It allows you to name a work area and access its stored information.There are two types of cursors: -

o Implicit cursors

**o** Explicit cursors

# Implicit cursors: -

* Defined by the oracle server.

# Explicit cursors: -

* They don’t have any name.
* It releases the memory allocated when the SQL statements execution is completed.
* In this we can’t use open, fetch and close commands.

Declaring the cursor:

* It is defined by the user.
* It is used by the user to process multiple rows returned by a select statement.
* Steps for defining an explicit cursor:
  + Declare the cursor.
  + Open the cursor.
  + Fetch data from the cursor.
  + Close the cursor.

SynTax: Cursor <cursor name> is SQL statement.

Example: cursor c1 is

Select \* from emp where sid between 100 and 110;

* + - Opening the cursor includes: Allocating memory

Parse the SQL statement for execution.

* + - The rows which are selected and stored in an Active Set and this active set are stored in the memory. The active set always points to the first row in the active set.
    - Fetch: To retrieve data from the active set and is stored in the variables.

E.g.: fetch ename into a;

* + - After the fetch instruction the active set is checked. If it is empty it enables the close else it moves the cursor to the next row in the active set and continues until the last row is reached. Once the last row is reaches it enables the close.

# Queries:

* So, close will be enabled in two cases: o When active set is empty

o When active set is pointing to the last row in the active set.

SynTax: close <cursor name>;

**Q1:**Write a query to demonstrate Implicit Cursor.

# Code:

set SERVEROUTPUT ON;

DECLARE

v\_num\_rows NUMBER; BEGIN

-- Update salaries of employees UPDATE employees

SET salary = salary \* 1.1 WHERE department\_id = 10;

-- Get the number of updated rows v\_num\_rows := SQL%ROWCOUNT;

-- Print the number of updated rows DBMS\_OUTPUT.PUT\_LINE('Number of updated rows: ' || v\_num\_rows);

END;

# Output:

**Q2.**Write a query to show the use of %notfound attribute. i.Declare

X integer (10);

A char (10);

Name varchar2 (20);

Begin

For I in 140.. 149

Loop

End;

Select first\_name into name from employees where employee\_id=i; End loop;

X: =sql%rowcount;

DBMS\_OUTPUT.PUT\_LINE ('no. of rows selected: '||x);

# Output:

No. of rows selected: 1

Statement processed.

ii. declare

vname employees.ename %type; vsal employees.salary %type;

begin

end;

select ename,salary into vname,vsal from employees where emp\_id=100; dbms\_output.put\_line('No of rows effected '||to\_char(sql%rowcount));

# Output:

No of rows effected 1

PL/SQL procedure successfully completed.

# Q3.Explicit cursor [pl/sql block to retrieve 10 employees’ details using cursor].

Declare

Vname employees.ename %type; Vjid employees.job\_id %type;

Cursor emp\_det is

select ename,job\_id into vname,vjid from employees where emp\_id between 100 and 175;

begin

open emp\_det ; for i in 1..10 loop

end;

fetch emp\_det into vname,vjid ;

dbms\_output.put\_line (i||'.Name is : '||vname||',Job id is : '||vjid); end loop;

close emp\_det;

# Output:

1. Name is : king, Job id is : ad\_pres 2.Name is : kochhar, Job id is : ad\_vp 3.Name is : de haan, Job id is : ad\_vp 4.Name is : hunold, Job id is : it\_prog 5.Name is : emst, Job id is : it\_prog 6.Name is : mourgos, Job id is : st\_man 7.Name is : rajs, Job id is : st\_clk 8.Name is : davies, Job id is : st\_clk 9.Name is : zlotkey, Job id is : sa\_man 10.Name is : able, Job id is : sa\_rep

PL/SQL procedure successfully completed.

# Q4.Write a pl/sql block using %rowcount attribute in explicit cursor:

Declare

vname employees.ename %type; vjid employees.job\_id %type;

cursor emp\_det is

select ename,job\_id into vname,vjid from employees where emp\_id=174 ;

begin

end;

open emp\_det ;

fetch emp\_det into vname,vjid; dbms\_output.put\_line(to\_char(emp\_det%rowcount)); close emp\_det;

# Output:

1

PL/SQL procedure successfully completed.

# Q5.Write a pl/sql block using % not found attribute in explicit cursor:

declare

vname employees.ename %type; vjid employees.job\_id %type;

cursor emp\_det is

select ename,job\_id into vname,vjid from employees where emp\_id=174 ;

begin

open emp\_det ;

fetch emp\_det into vname,vjid; if emp\_det%notfound then

dbms\_output.put\_line('NOT found is true');

else

dbms\_output.put\_line('NOT found is false');

end;

end if;

close emp\_det;

# Output:

NOT found is false

PL/SQL procedure successfully completed.

# Q6.Write a pl/sql block using %found attribute in explicit cursor

declare

vname employees.ename %type; vjid employees.job\_id %type;

cursor emp\_det is

select ename,job\_id into vname,vjid from employees where emp\_id=174 ;

begin

open emp\_det ;

fetch emp\_det into vname,vjid; if emp\_det%found then

dbms\_output.put\_line('found is true');

else

dbms\_output.put\_line(' found is false');

end;

end if;

close emp\_det;

# Output:

found is true

PL/SQL procedure successfully completed.

# Experiment:10 exceptional handling

**AIM:** write the queries for exception handling and in pl/sql

# THEORY:

**Exception handling:**

* An exception is an identifier which is raised when an oracle error occurs in pl/sql block.
* Warning or error condition is called an exception
* When an error occurs an exception is raised i.e.; the normal execution stops and the control transfers to the exception handling block of the PL/SQL block.
* The ‘throw’ and ‘try’ keywords which were normally used in exceptional handling have their functionalities in DBMS as ‘raise’ and ‘exception’.
* Exceptional block is not present in the declare .
* Exceptional block has exceptional handling statements.
* Exceptions can be three types:
  + Predefined exceptions
  + Non predefined exceptions
  + User defined exceptions

# Predefined exceptions:

* One of the approximately 20 errors that occurs in oracle PL/SQL block/code
* No need to declare the exception or raise the exception explicitly.
* E.g.: ora\_010001 invalid cursor Ora\_01403 no data found

# Non-predefined exception:

* Any other standard errors other than the predefined 20 come under non predefined exceptions.
* Needs to declare the exception explicitly in the declarative section.
* Allow the oracle server to raise that exception implicitly.
* Syntax:

Exception\_name exception;

PRAGMA\_exception\_INIT (Exception\_name, error number);

# User defined exceptions:

* In this, the user has to explicitly declare the exception, raise an exception as well as handle the exception.
* Steps for creating the user defined exception
  + declare the exception
  + raise the exception
  + Use reference name to handle the exception.

# Raise\_application\_error procedure:

* It is used to raise the user defined exceptions.
* It can be used inside PL/SQL block and an exception block

# Raise an exception in oracle:

There are two ways to raise an exception.

* 1. When an oracle error occurs, the associated error is automatically raised by the oracle server.

E.g. when no roes are selected by the select statement then oracle server raises ‘no data found’ exception.

* 1. User can explicitly raise an exception by using ‘raise’ statement or ‘Raise\_application\_error’ statement.

# Handling the exception:

* + 1. Trapping the exception: if an exception is raised, control is passed to the exception section of block. The exception will be trapped if it is handled in that exception block itself.
    2. Propagating an exception: if the exception is not handled in the outer block, it is propagated / forwarded to the enclosed block.

# Queries:

**1.write a predefined exception for cursor\_already\_open**:-When you open a cursor that is already open(ORA-06511)

Declare

name employee.ename%type; sal employee.salary%type;

cursor c1 is

select ename, salary from employee where eid between 101 and 105;

begin

open c1; loop

fetch c1 into name,sal; Dbms\_output.put\_line (name||sal); exit when c1%notfound;

end loop; open c1;

exception

when cursor\_already\_open then Dbms\_output.put\_line ('cursor already open');

end;

# OUTPUT:

Mohan rao 34000

Raja ram 3500

Meena Roy 6500

David 10000

Janaki 20000

Janaki 20000 Cursor already open

Statement processed.

**2..write a predefined exception for invalid cursor:**-When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened (ORA-01001).

Declare

id number(10); cursor c1 is

select employee\_id from employees;

begin

open c1; close c1;

fetch c1 into id;

Exception

when invalid\_cursor then

Raise\_application\_error(-20003,'invalid cursor to fetch');

end;

ORA-20003: invalid cursor to fetch

**3..write a predefined exception for NO\_DATA\_FOUND**:-When a SELECT...INTO clause does not return any row from a table (ORA-01403).

set serveroutput on declare

vname employ2.lastname%type; vsal employ2.salary%type;

begin

select lastname,salary into vname,vsal from employ2 where eid=810;

exception

when no\_data\_found then DBMS\_output.put\_line('no rows selected');

end;

no rows selected.

**4..write a predefined exception for TOO\_MANY\_ROWS:-**When you SELECT or fetch more than one row into a record or variable (ORA-01422).

set serveroutput on declare

vname employ2.lastname%type; vsal employ2.salary%type;

begin

select lastname,salary into vname,vsal from employ2 where salary between 10000 and 40000;

exception

when too\_many\_rows then DBMS\_output.put\_line('too many rows selected');

end;

too many rows selected

**5.write a predefined exception for ZERO\_DIVIDE**:-When you attempt to divide a number by zero (ORA-01476)

Declare

Var1 number (3):=0;

Var2 number (4):=100;

vsal employees.salary%type;

Begin

Update employees

Set salary= salary+ (var2/var1);

Exception

When ZERO\_DIVIDE then

Dbms\_output.put\_line ('divide by zero exception');

end; Output:

divide by zero exception Statement processed.

**6..write a predefined exception for case\_not\_found:**-this exception is raised when the else block isn’t use.

declare

name varchar2(10); sal number(10,2); did number(5);

begin

select ename,salary,dept\_id into name,sal,did

from employee where eid=106; case

end;

when did=204 then dbms\_output.put\_line('Meeting at 2 p.m.'); when did=205 then dbms\_output.put\_line('meeting at 3 p.m.');

end case; DBMS\_OUTPUT.PUT\_LINE(name||sal||did);

OUTPUT:

ORA-06592: CASE not found while executing CASE statement

**User defined exceptions:**

**1.write a userdefined exception for restricting employees based on experience**

declare

name varchar2(20); experience number(10); no\_promotion exception;

begin

select first\_name||last\_name,round(months\_between(sysdate,hire\_date)/12,0) into name,experience

from employees where employee\_id=107; case

when experience>15 then dbms\_output.put\_line(name||'elligible for promotion category');

when experience=15 then dbms\_output.put\_line(name||'get prepared for promotion test');

when experience<15 then raise no\_promotion; end case;

Exception

when no\_promotion

then

end;

dbms\_output.put\_line(name||'-experience is='||experience

||'yrs-'||'CANT BE PROMOTED---EXPERIENCE < 15yrs');

# OUTPUT:

DianaLorentz-experience is=13yrs-CANT BE PROMOTED---EXPERIENCE < 15yrs

**RAISE\_APPLICATION\_ERROR:**

This is used to convey user define message to the end user.

# RAISE\_APPLICATION\_ERROR inside the EXCEPTION BLOCK

set serveroutput on declare

vname employ2.lastname%type;

vsal employ2.salary%type;

begin

select lastname,salary into vname,vsal from employ2 where eid=810;

exception

when no\_data\_found then raise\_application\_error(-20101,'no rows selected');

end;

# output:

declare

\*

ERROR at line 1:

ORA-20111: no rows selected ORA-06512: at line 12

# RAISE\_APPLICATION \_ERROR using cursors

declare

vname employ2.lastname%type; vsal employ2.salary%type; cursor c1 is

select lastname,salary into vname,vsal from employ2 where eid=220;

begin

end;

open c1;

fetch c1 into vname,vsal; if c1%notfound then

raise\_application\_error(-20111,'no rows selected'); close c1;

end if;

# Output:

declare

\*

ERROR at line 1:

ORA-20111: no rows selected ORA-06512: at line 12.