1601-21-733-055 DBMS RECORD INDEX

|  |  |  |
| --- | --- | --- |
| **S.No** | **Program** | **Page No.** |
| 1 | Write the queries for Data Manipulation and Data  Definition Language. | 2 |
| 2 | Write SQL queries for salesman, Customers and  Orders database and Demonstrating Operators. | 7 |
| 3 | Write SQL query demonstrating Single row  functions and general functions. | 19 |
| 4 | Write SQL queries demonstrating group functions | 33 |
| 5 | Write SQL queries using group by clause | 36 |
| 6 | Write SQL queries using constraints. | 36 |
| 7 | Write SQL queries to demonstrate joins , Sets and  EXERCISES. | 42 |
| 8 | Write programs on sub queries | 67 |
| 9 | Write SQL Queries demonstrating views | 79 |
| 10 | Write programs using PL/SQL and Implicit cursors | 87 |
| 11 | Write programs on procedures and functions | 99 |
| 12 | Write programs on Explicit cursors | 106 |
| 13 | Write programs on exception handling and triggers | 110 |

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 1&2 | |
| **Course Code : 20CSC14**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**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;*

**THEORY:**

DML: A **data manipulation language** (**DML**) is a family of syntax elements similar to a computer programming language used for selecting, inserting, deleting and updating data in a database. Performing read-only queries of data is sometimes also considered a component of DML. Commands in DML are:

1. INSERT
2. UPDATE
3. DELETE
4. SELECT

## DML COMMANDS:

**SYNTAX:**

INSERT Statement:

*Single Row into a Table:*

*INSERT INTO table – name [column- identifier-comma-list)] VALUES (column-valuecomma-list);*

*Multiple Row into a Table:*

*insert into <table name> values (&col1, &col2, ….);*

UPDATE Statement: *UPDATE table-name SET update- column-list [WHERE search- condition];*

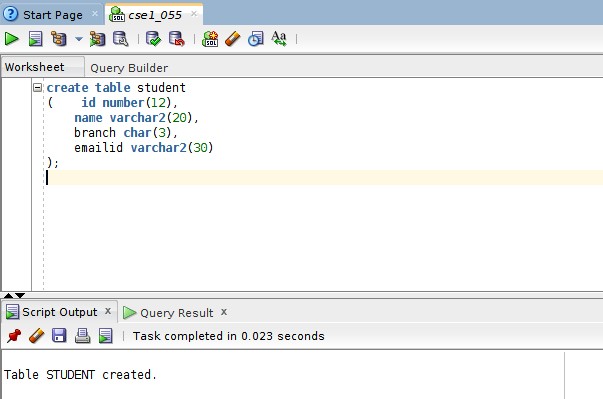
DELETE Statement: *DELETE FROM table-name [WHERE search- condition];*

NAME: MOHAMMED MOHIUDDIN ROLL NUMBER : 1601-21-733-055 CSE-1

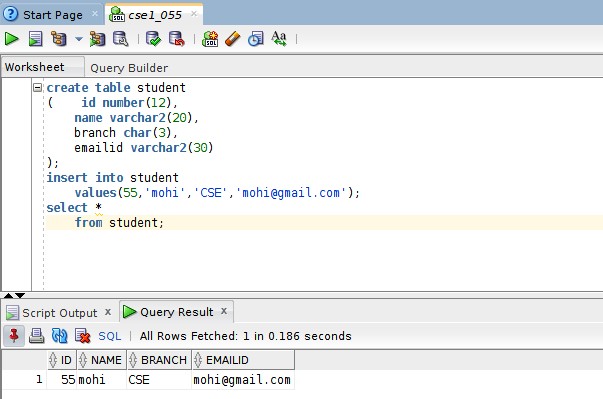
DBMS LAB WEEK-1

TASKS:-

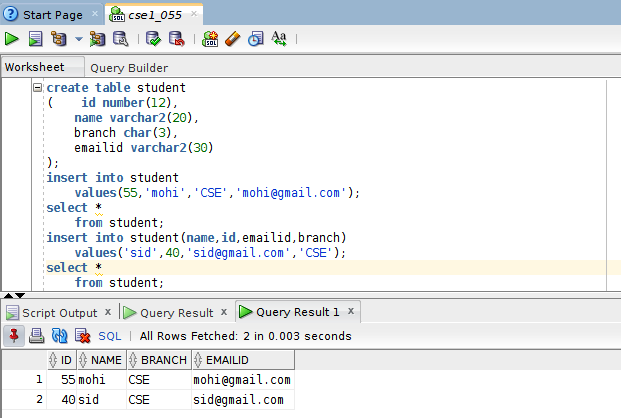
1. CREATING A TABLE NAMED “STUDENT”.



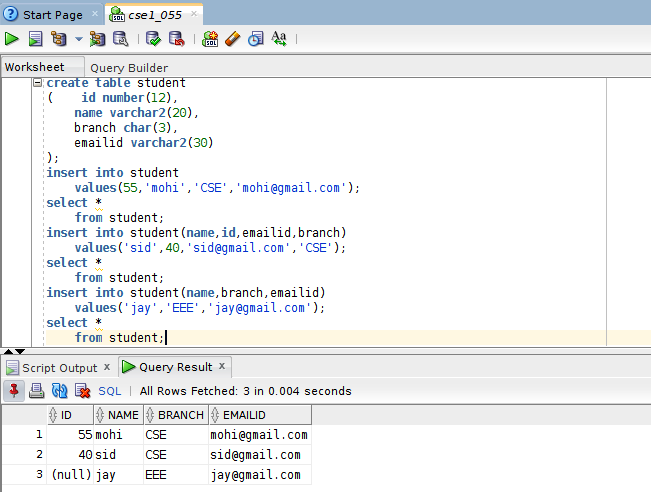
1. INSERTING A TUPLE INTO TABLE



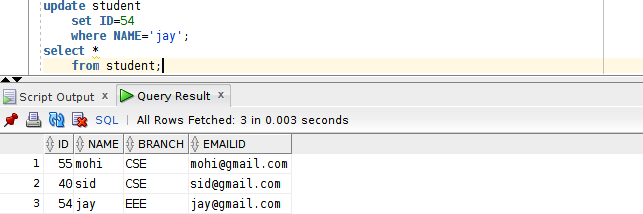
1. INSERTING NEW TUPLE BY GIVING ATTRIBUTES IN DIFFERENT ORDER



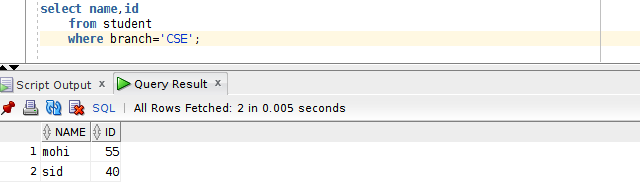
1. INSERTING NULL VALUE BY NOT SPECIFYING A PARTICULAR COLUMN



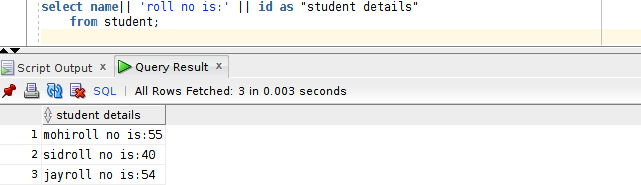
1. UPDATING A TUPLE ATTRIBUTE



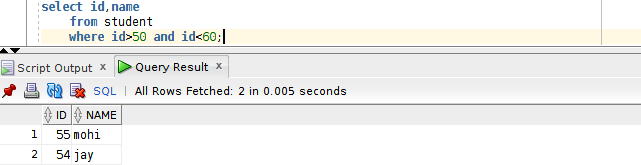
1. DISPLAYING TABLE WITH CONSTRAINT THAT BRANCH AS ONLY CSE



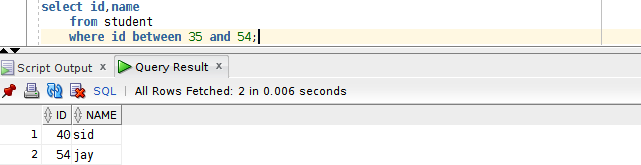
1. USING CONCATENATE OPERATOR AND ALIASING IT



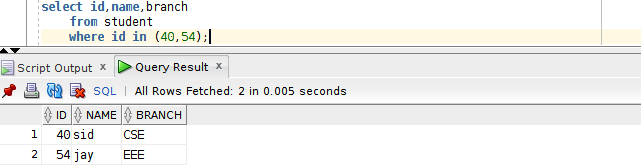
1. USING RELATONAL OPERATORS



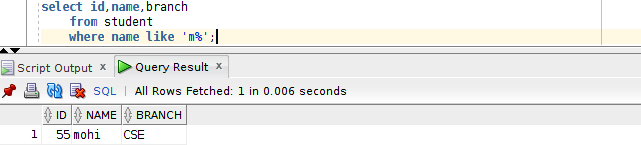
1. USING BETWEEN AND- RANGE OPERATOR



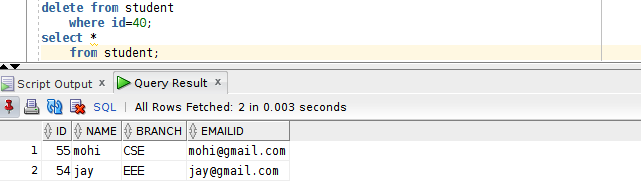
1. USING IN – RANGE OPERATOR



1. USING LIKE OPERATOR



1. DELETING TUPLE



WEEK-2

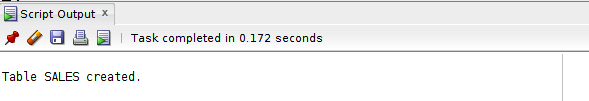
1. From the following table, write a SQL query to find the details of those salespeople who come from the 'Paris' City or 'Rome' City. Return salesman\_id, name, city, commission.

CODE:-

create table sales (

salesman\_id number(4), s\_name varchar2(20), city varchar2(20), commision number(3,2)

); OUTPUT:-



CODE:

insert into sales

values(5001,'James Hoog','New York',0.15); insert into sales

values(5002,'Nail Knite','Paris',0.13); insert into sales

values(5005,'Pit Alex','London',0.11); insert into sales

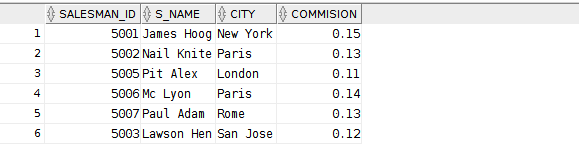
values(5006,'Mc Lyon','Paris',0.14);

insert into sales

values(5007,'Paul Adam','Rome',0.13); insert into sales

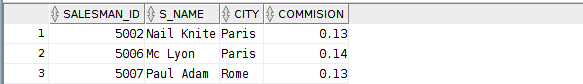
values(5003,'Lawson Hen','San Jose',0.12); select \*

from sales; OUTPUT:-



CODE:-

select \* from sales

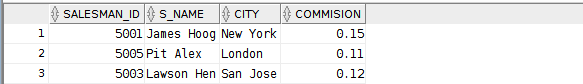
where city in ('Rome','Paris'); OUTPUT:-

2.

From the following table, write a SQL query to find the details of those salespeople who live in cities other than Paris and Rome. Return salesman\_id, name, city, commission.

CODE:-

select \* from sales

where city not in ('Rome','Paris'); OUTPUT:-

3. .

From the following table, write a SQL query to retrieve the details of all customers whose ID belongs to any of the values 3007, 3008 or 3009. Return customer\_id, cust\_name, city, grade, and salesman\_id.

CODE:-

create table customer (

customer\_id number(4), cust\_name varchar2(20), city varchar2(10),

grade number(3), salesman\_id number(4)

);

insert into customer

values(3002,'Nick Rimando','New York',100,5001); insert into customer

values(3007,'Brad Davis','New York',200,5001); insert into customer

values(3005,'Graham Zusi','California',200,5002); insert into customer

values(3008,'Julian Green','London',300,5002); insert into customer

values(3004,'Fabian Johnson','Paris',300,5006); insert into customer

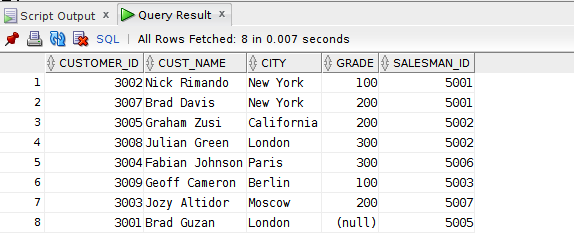
values(3009,'Geoff Cameron','Berlin',100,5003); insert into customer

values(3003,'Jozy Altidor','Moscow',200,5007);

insert into customer(CUSTOMER\_ID,cust\_name,city,salesman\_id) values(3001,'Brad Guzan','London',5005);

select \*

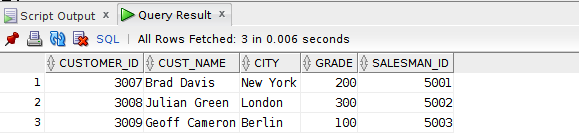
from customer; OUTPUT:-



CODE:-

select \*

from customer

where CUSTOMER\_ID in (3007,3008,3009); 4.

From the following table, write a SQL query to find salespeople who receive commissions between

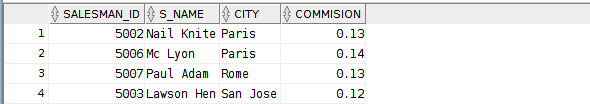
0.12 and 0.14 (begin and end values are included). Return salesman\_id, name, city, and commission

CODE:-

select \*

from SALES

where COMMISION between 0.12 and 0.14;

OUTPUT:- 5.

From the following table, write a SQL query to select orders between 500 and 4000 (begin and end values are included). Exclude orders amount 948.50 and 1983.43. Return ord\_no, purch\_amt, ord\_date, customer\_id, and salesman\_id.

CODE:-

create table orders (

order\_no number(6), purch\_amt number(8,3), ord\_date date, customer\_id number(5), salesman\_id number(5)

);

insert into orders

values(70001,150.5,'05-Oct-2012',3005,5002);

insert into orders

values(70009,270.65,'10-Sep-2012',3001,5005);

insert into orders

values(70002,65.26,'05-Oct-2012',3002,5001);

insert into orders

values(70004,110.5,'17-Aug-2012',3009,5003);

insert into orders

values(70007,948.5,'10-Sep-2012',3005,5002);

insert into orders

values(70005,2400.6,'27-Jul-2012',3007,5001);

insert into orders

values(70008,5760,'10-Sep-2012',3002,5001);

insert into orders

values(70010,1983.43,'10-Oct-2012',3004,5006);

insert into orders

values(70003,2480.4,'10-Oct-2012',3009,5003);

insert into orders

values(70012,250.45,'27-Jun-2012',3008,5002);

insert into orders

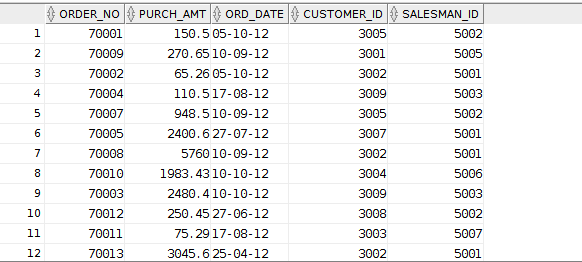
values(70011,75.29,'17-Aug-2012',3003,5007);

insert into orders

values(70013,3045.6,'25-Apr-2012',3002,5001);

select \* from orders;

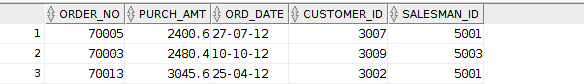
OUTPUT:-



CODE:-

select \* from orders

where PURCH\_AMT between 500 and 4000 and PURCH\_AMT not in (948.5,1983.43);

OUTPUT:- 6.

From the following table, write a SQL query to retrieve the details of the salespeople whose names begin with any letter between 'A' and 'L' (not inclusive). Return salesman\_id, name, city, commission.

CODE:-

select \* from sales

where S\_NAME between 'A' and 'J’;

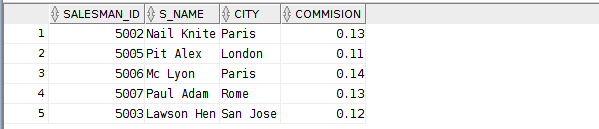
OUTPUT:- 7.

From the following table, write a SQL query to find the details of all salespeople except those whose names begin with any letter between 'A' and 'L' (not inclusive). Return salesman\_id, name, city, commission.

CODE:

select \* from sales

where S\_NAME not between 'A' and 'J'; OUTPUT:-

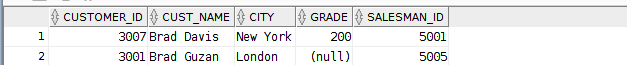
8. .

From the following table, write a SQL query to retrieve the details of the customers whose names begins with the letter 'B'. Return customer\_id, cust\_name, city, grade, salesman\_id..

CODE:

select \*

from CUSTOMER

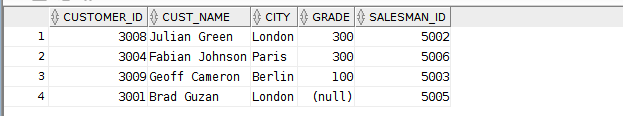
where CUST\_NAME like 'B%'; OUTPUT:-

1. From the following table, write a SQL query to find the details of the customers whose names end with the letter 'n'. Return customer\_id, cust\_name, city, grade, salesman\_id

CODE:

select \*

from CUSTOMER

where CUST\_NAME like '%n'; OUTPUT:-

1. From the following table, write a SQL query to find the details of those salespeople whose names begin with ‘N’ and the fourth character is 'l'. Rests may be any character. Return salesman\_id, name, city, commission.

CODE:

select \* from sales

where S\_NAME like 'N l%'; OUTPUT:-

1. From the following table, write a SQL query to find all those customers who does not have any grade. Return customer\_id, cust\_name, city, grade, salesman\_id.

CODE:

select \*

from CUSTOMER where GRADE is NULL;

OUTPUT:-



1. From the following table, write a SQL query to locate the employees whose last name begins with the letter 'D'. Return emp\_idno, emp\_fname, emp\_lname and emp\_dept.

CODE:

create table emp\_details (

emp\_idno number(6), emp\_fname varchar2(20), emp\_lname varchar2(20), emp\_dept number(2)

);

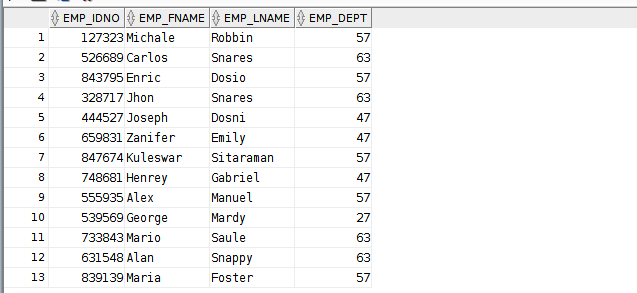
insert into EMP\_DETAILS values(127323,'Michale','Robbin',57); insert into EMP\_DETAILS values(526689,'Carlos','Snares',63); insert into EMP\_DETAILS values(843795,'Enric','Dosio',57); insert into EMP\_DETAILS values(328717,'Jhon','Snares',63); insert into EMP\_DETAILS values(444527,'Joseph','Dosni',47); insert into EMP\_DETAILS values(659831,'Zanifer','Emily',47); insert into EMP\_DETAILS

values(847674,'Kuleswar','Sitaraman',57); insert into EMP\_DETAILS values(748681,'Henrey','Gabriel',47); insert into EMP\_DETAILS values(555935,'Alex','Manuel',57);

insert into EMP\_DETAILS values(539569,'George','Mardy',27); insert into EMP\_DETAILS values(733843,'Mario','Saule',63); insert into EMP\_DETAILS values(631548,'Alan','Snappy',63); insert into EMP\_DETAILS values(839139,'Maria','Foster',57);

select \*

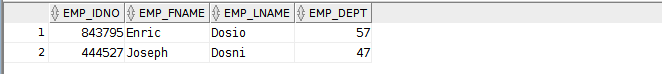
from EMP\_DETAILS; OUTPUT:-



CODE:-

select \*

from EMP\_DETAILS

where EMP\_LNAME like 'D%'; OUTPUT:-

OPERATORS:-

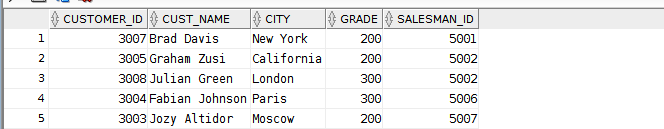
1. From the following table, write a SQL query to locate the details of customers with grade values above 100. Return customer\_id, cust\_name, city, grade, and salesman\_id. Sample table: customer

CODE:

select \*

from customer where GRADE>100;

OUTPUT:-



1. From the following table, write a SQL query to find all the customers in ‘New York’ city who have a grade value above 100. Return customer\_id, cust\_name, city, grade, and salesman\_id. Sample table: customer

CODE:

select \*

from customer

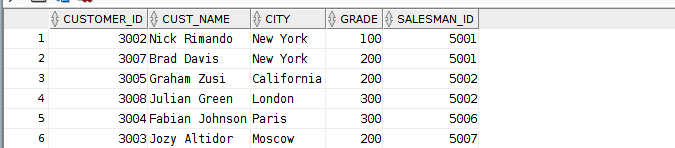
where GRADE>100 and city='New York'; OUTPUT:

1. From the following table, write a SQL query to find customers who are from the city of New York or have a grade of over 100. Return customer\_id, cust\_name, city, grade, and salesman\_id.

CODE:

select \*

from customer

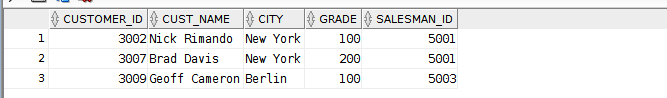
where GRADE>100 or city='New York'; OUTPUT:-

1. From the following table, write a SQL query to find customers who are either from the city 'New York' or who do not have a grade greater than 100. Return customer\_id, cust\_name, city, grade, and salesman\_id. Sample table: customer

CODE:

select \*

from customer

where GRADE=100 or city='New York'; OUTPUT:-

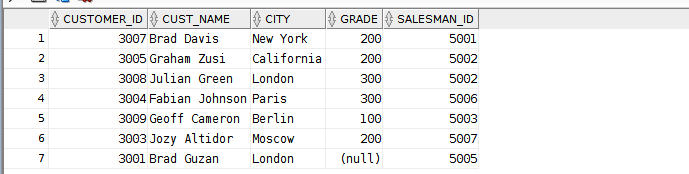
1. From the following table, write a SQL query to identify customers who do not belong to the city of 'New York' or have a grade value that exceeds 100. Return customer\_id, cust\_name, city, grade, and salesman\_id. Sample table: customer

CODE:-

select \*

from customer

where GRADE>100 or city<>'New York'; OUTPUT:-



1. From the following table, write a SQL query to find details of all orders excluding those with ord\_date equal to '2012-09-10' and salesman\_id higher than 5005 or purch\_amt greater than 1000.Return ord\_no, purch\_amt, ord\_date, customer\_id and salesman\_id. Sample table : orders

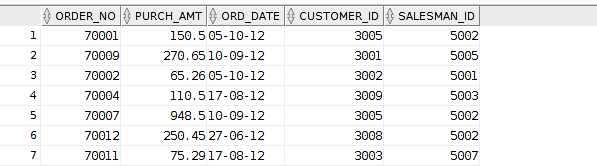
CODE: SELEC FROM WHER

|  |  |  |
| --- | --- | --- |
| T | \* |  |
| orders | | |
| E | NOT | |

((ord\_date ='2012-09-10'

AND salesman\_id>5005) OR purch\_amt>1000.00);

OUTPUT:-



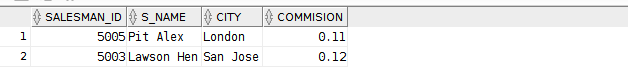
1. From the following table, write a SQL query to find the details of those salespeople whose commissions range from 0.10 to0.12. Return salesman\_id, name, city, and commission. Sample table : salesman

CODE:

select \*

from SALES

where COMMISION between 0.10 and 0.12;

OUTPUT:-

1. From the following table, write a SQL query to find details of all orders with a purchase amount less than 200 or exclude orders with an order date greater than

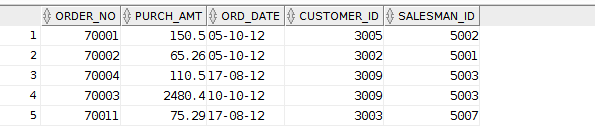
or equal to '2012-02-10' and a customer ID less than 3009. Return ord\_no, purch\_amt, ord\_date, customer\_id and salesman\_id.

Sample table : orders CODE:-

SELECT \*

FROM orders WHERE(purch\_amt<200 OR NOT(ord\_date>='10-Feb-2012' AND customer\_id<3009));

OUTPUT:-



1. From the following table, write a SQL query to find all orders that meet the following conditions. Exclude combinations of order date equal to '2012-08-17' or customer ID greater than 3005 and purchase amount less than 1000.

Sample table : orders CODE:-

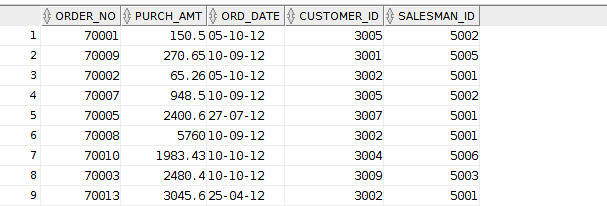
SELECT \*

FROM orders

WHERE NOT((ord\_date ='17-Aug-2012' OR customer\_id>3005)

AND purch\_amt<1000);

OUTPUT:-



1. From the following table, write a SQL query to find the details of all employees whose last name is ‘Dosni’ or ‘Mardy’. Return emp\_idno, emp\_fname,

emp\_lname, and emp\_dept. Sample table : emp\_details

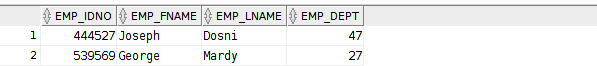
CODE:-

select \*

from EMP\_DETAILS

where EMP\_LNAME in ('Dosni','Mardy');

OUTPUT:-



1. From the following table, write a SQL query to find the employees who work at depart 47 or 63. Return emp\_idno, emp\_fname, emp\_lname, and

emp\_dept.

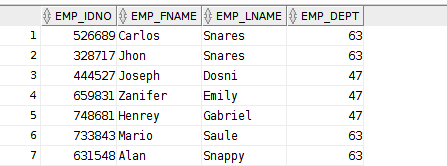
Sample table : emp\_details CODE:-

select \*

from EMP\_DETAILS

where EMP\_DEPT in (47,63);

OUTPUT:-



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 3 | |
| **Course Code : 20CSC14**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**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:

|  |  |
| --- | --- |
| **Format Model** | **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)

WEEK-3

SINGLE ROW FUNCTIONS

CASE MANIPULATION FUNCTION

1. Lower CODE:-

select lower('MOHI') from dual;

OUTPUT:-



1. Upper CODE:-

select upper('mohi') from dual;

OUTPUT:-



1. INITCAP CODE:-

select initcap('MOHI') from dual;

OUTPUT:-



CHARACTER MANIPULATION FUNCTIONS

1. Concat CODE:-

select concat(‘Mohi’,’uddin’) from dual;

OUTPUT:-

1. Length CODE:-

select length('Mohiuddin') from dual;

OUTPUT:-



1. instr CODE:-

select instr('Mohiuddin','h') from dual;

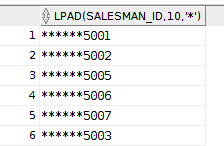
OUTPUT:-



1. lpad CODE:-

select lpad(salesman\_id,10,'\*') from sales;

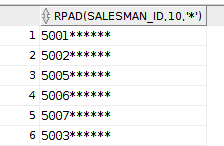
OUTPUT:-



1. Rpad CODE:-

select rpad(salesman\_id,10,'\*') from sales;

OUTPUT:-



1. substr CODE:

select SUBSTR('Mohiuddin',3,4) from dual;

OUTPUT:-



NUMBER FUNCTIONS

1. ROUND CODE:-

select round(45.923,2) from dual;

OUTPUT:-



CODE:-

select round(45.923,-1) from dual;

OUTPUT:-



1. TRUNC CODE:-

select trunc(2436.87,-2) from dual;

OUTPUT:-



1. mod CODE:-

select mod(55,12) from dual;

OUTPUT:-



DATE FUNCTIONS

1. add\_months CODE:-

select add\_months('25-Oct-2023',6) from dual;

OUTPUT:-



1. months\_between CODE:-

select months\_between('22-Apr-2004','03-05-2023') from dual;

OUTPUT:-



1. next\_day CODE:-

select next\_day('22-Apr-2004','Tuesday') from dual;

OUTPUT:-



1. last\_day CODE:-

select last\_day('22-Apr-2023') from dual;

OUTPUT:-



1. round CODE:-

select round(To\_date('22-Apr-2004'),'month')

from dual; OUTPUT:-

CODE:-

select round(To\_date('22-Apr-2004'),'year') from dual;

OUTPUT:-



1. trunc CODE:-

select trunc(To\_date('22-Apr-2004'),'month') from dual;

OUTPUT:-



CODE:-

select trunc(To\_date('22-Apr-2004'),'year') from dual;

OUTPUT:-



COVERSION FUNCTIONS

1. To\_char CODE:-

select To\_char(To\_date('22-Apr-2004'),'YYY') from dual;

OUTPUT:-



CODE:-

select To\_char(To\_date('22-Apr-2004'),'DY') FROM dual;

OUTPUT:-



CODE:-

select To\_char(To\_date('22-Apr-2004'),'RM') FROM dual;

OUTPUT:-



CODE:-

select To\_char(12345,'99999.99') from dual;

OUTPUT:-



CODE:-

select To\_char(12345,'$099,999.99') from dual;

OUTPUT:-



CODE:-

select To\_char(To\_date('22-Apr-2004'),'DDSPTH') from dual;

OUTPUT:-



CODE:-

select To\_char(To\_date('22-Apr-2004'),'DDSP "of the" MONTH,YYYY') from dual;

OUTPUT:-



1. To\_number CODE:-

select To\_number('$1,234','$9,999') from dual;

OUTPUT:-



1. To\_date CODE:-

select To\_date('22-Apr-2004') from dual;

OUTPUT:-

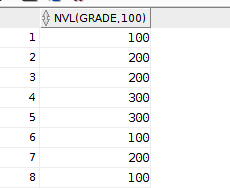


GENERAL FUNCTIONS

1. NUL CODE:-

select NVL(grade,100) from CUSTOMER;

OUTPUT:-



## GENERAL FUNCTIONS

**Queries to demonstrate all the general functions.**

**TABLE:-**

select \* from cbit



* 1. NVL

select nvl(sid,129) from cbit



* 1. NVL2

select nvl2(sid,name,'ting') from cbit



* 1. NULLIF

select NULLIF('jon','jon') from dual



select NULLIF('jon','snow') from dual



* 1. COALESCE

select coalesce(null,null,'hh') from dual



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 4,5&6 | |
| **Course Code : 20CSC14**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**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 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);*

**AIM:** Write the queries to demonstrate various constraints in SQL.

**THEORY:**

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Following are some of the most commonly used constraints available in SQL.

* + - [NOT NULL Constraint](https://www.tutorialspoint.com/sql/sql-not-null.htm) − Ensures that a column cannot have NULL value.
    - [UNIQUE Constraint](https://www.tutorialspoint.com/sql/sql-unique.htm) − Ensures that all values in a column are different.
    - [PRIMARY Key](https://www.tutorialspoint.com/sql/sql-primary-key.htm) − Uniquely identifies each row/record in a database table.
    - [FOREIGN Key](https://www.tutorialspoint.com/sql/sql-foreign-key.htm) − Uniquely identifies a row/record in any of the given database table.
    - [CHECK Constraint](https://www.tutorialspoint.com/sql/sql-check.htm) − The CHECK constraint ensures that all the values in a column satisfies

certain conditions.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

160121733055 DBMS RECORD

WEEK-4

## GROUP FUNCTIONS

1. AVG CODE

select avg(distinct commision) from sales;

OUTPUT



CODE

select avg(all commision) from sales;

OUTPUT



1. SUM,MIN,MAX CODE

select avg(commision),sum(commision),max(commision),min(commision) from sales;

OUTPUT



CODE

select max(city),min(city) from sales;

OUTPUT



1. COUNT CODE

select count(\*) from sales;

OUTPUT



CODE

select count(distinct commision) from sales;

OUTPUT



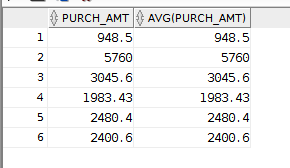
## GROUP BY CLAUSE

CODE

select purch\_amt,avg(purch\_amt) from ORDERS

group by(purch\_amt)

having avg(purch\_amt)>500; OUTPUT



## CONSTRAINTS

1. NOT NULL AND UNIQUE

CODE

create table stu (

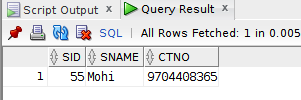
sid number(15) not null, sname varchar2(20) unique, ctno number(20)

);

insert into stu

values(55,'Mohi',9704408365);

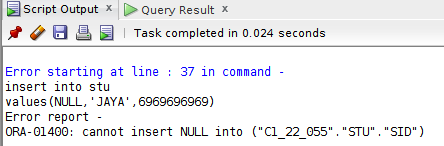
select \* from stu;



CODE

insert into stu values(NULL,'JAYA',6969696969);

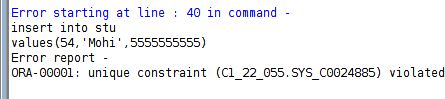
ERROR



CODE

insert into stu values(54,'Mohi',5555555555);

ERROR



1. PRIMARY KEY CODE

create table dept (

did number(10) primary key, dname varchar2(20),

HOD char(20)

);

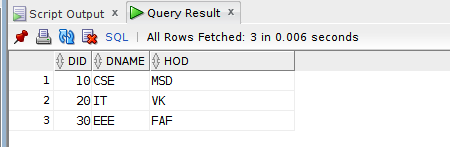
insert into dept values(10,'CSE','MSD');

insert into dept values(20,'IT','VK');

insert into dept values(30,'EEE','FAF');

select \* from dept;

TABLE



FOREIGN KEY CODE

create table emp (

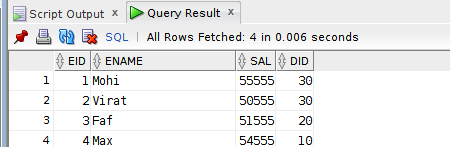
eid number(10) primary key, ename char(20),

sal number(9), did number(10),

constraint emp\_did\_fk foreign key(did) References dept(did)

);

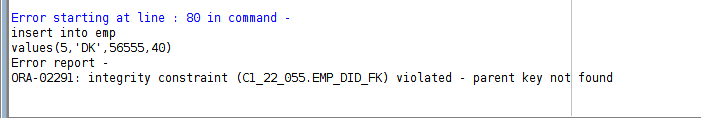
insert into emp values(1,'Mohi',55555,30); insert into emp values(2,'Virat',50555,30); insert into emp values(3,'Faf',51555,20); insert into emp values(4,'Max',54555,10); select \*

from emp; TABLE

CODE

insert into emp values(5,'DK',56555,40);

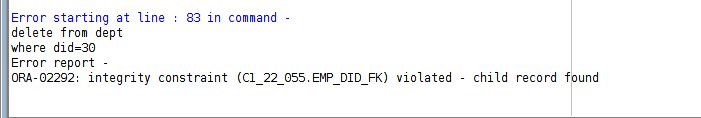
ERROR



ON DELETE CASCADE CODE

delete from dept where did=30;

ERROR

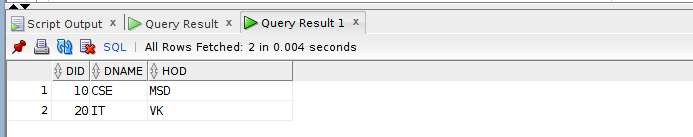


CODE

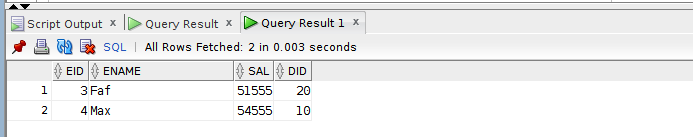
delete from dept where did=30;

select \* from emp; select \* from DEPT;

DEPT TABLE



EMP TABLE

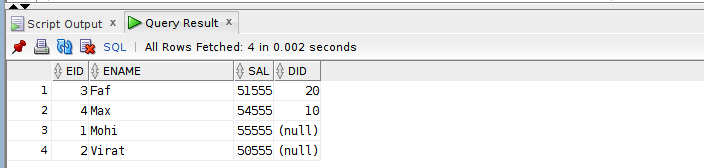


ON DELETE SET NULL CODE

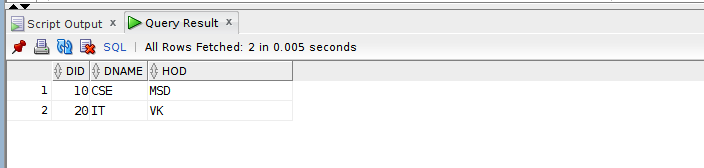
delete from dept where did=30;

select \* from emp; select \* from DEPT;

EMP TABLE



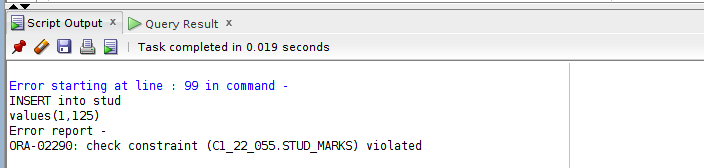
DEPT TABLE



CHECK

INSERT into stud values(1,125);

ERROR



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** *7* | |
| **Course Code : 20CSC14**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**AIM:** Write the queries to demonstrate various joins 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.

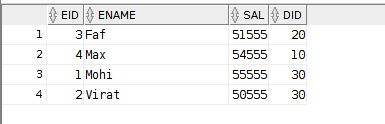
Here are the different types of the JOINs in SQL:

* 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.

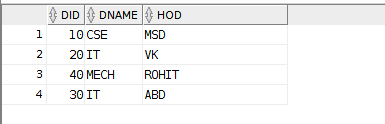
1601-21-733-055 DBMS RECORD WEEK-5

JOINS

1. CROSS JOIN EMP TABLE



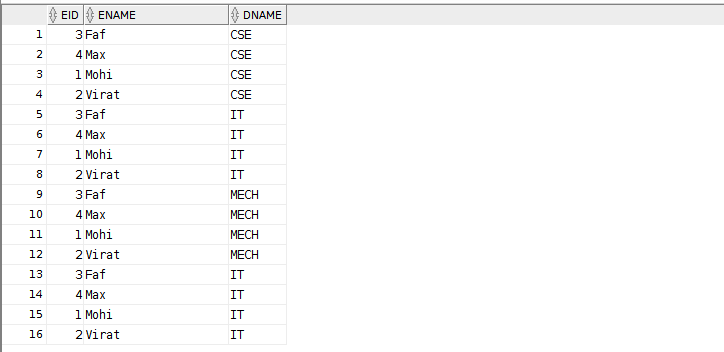
DEPT TABLE



CODE

OLDER VERSION

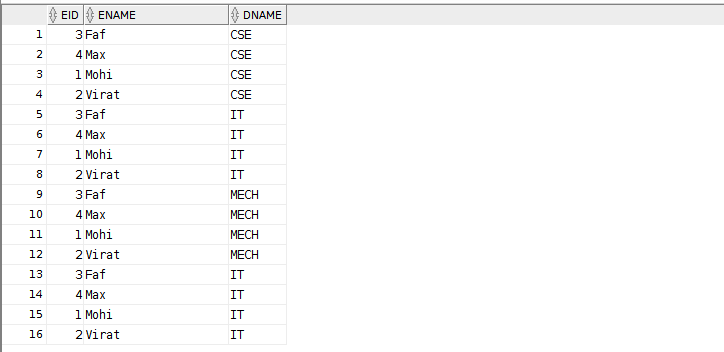
select eid,ename,dname from emp,dept; OUTPUT



NEW VERSION

select eid,ename,dname from emp cross join dept;

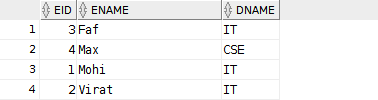
OUTPUT



1. Natural join / Equi join OLDER VERSION CODE

select eid,ename,dname from emp,dept

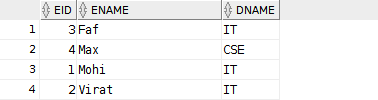
where emp.DID = dept.DID; OUTPUT



NEW VERSION CODE

select eid,ename,dname

from emp natural join dept; OUTPUT

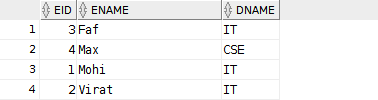


1. ON CLAUSE

NEW VERSION CODE

select eid,ename,dname from emp join dept

on emp.DID = dept.DID; OUTPUT



GRADES TABLE

create table grades (

grade char(1), lowest number(6), highest number(6)

);

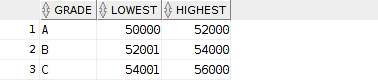
insert into grades values('A',50000,52000);

insert into grades values('B',52001,54000);

insert into grades values('C',54001,56000);

select \* from grades;

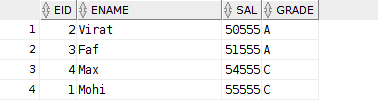
OUTPUT



OLDER VERSION NON-EQUI JOIN

select eid,ename,sal,grade from emp,grades

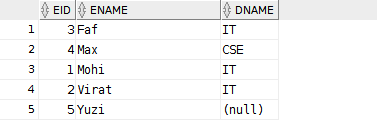
where sal between lowest and highest; OUTPUT



1. OUTER JOIN LEFT OUTER JOIN OLDER VERSION CODE

select eid,ename,dname from emp,dept

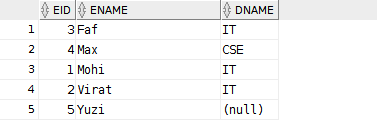
where emp.DID = dept.DID(+); OUTPUT



NEW VERSION CODE

select eid,ename,dname from emp left outer join dept on emp.DID = dept.DID;

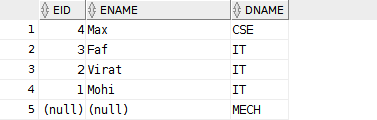
OUTPUT



RIGHT OUTER JOIN OLDER VERSION CODE

select eid,ename,dname from emp,dept

where emp.DID(+) = dept.DID; OUTPUT

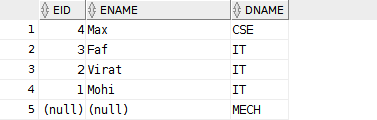


NEW VERSION CODE

select eid,ename,dname

from emp right outer join dept on emp.DID = dept.DID;

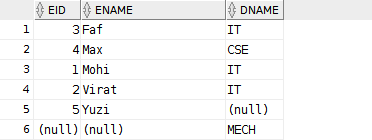
OUTPUT



FULL OUTER JOIN CODE

select eid,ename,dname from emp full outer join dept on emp.DID = dept.DID;

OUTPUT



SET OPERATORS

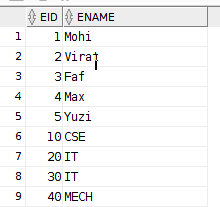
1. UNION| CODE

select eid,ename from emp

union

select did,dname from dept;

OUTPUT



union all includes duplicates

1. INTERSECT CODE

select eid,ename from emp intersect

select did,dname from dept;

OUTPUT



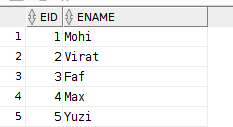
1. MINUS CODE

select eid,ename from emp

minus

select did,dname from dept;

OUTPUT

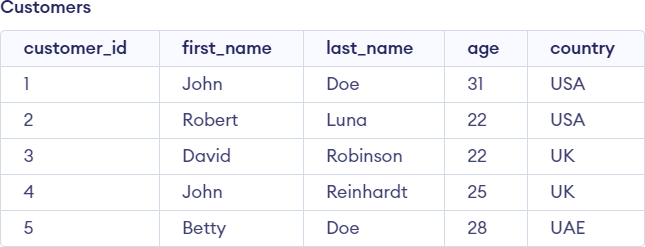


# 1601-21-733-055

**INCOMPLETE WORK**

## ORDER BY CASES:-

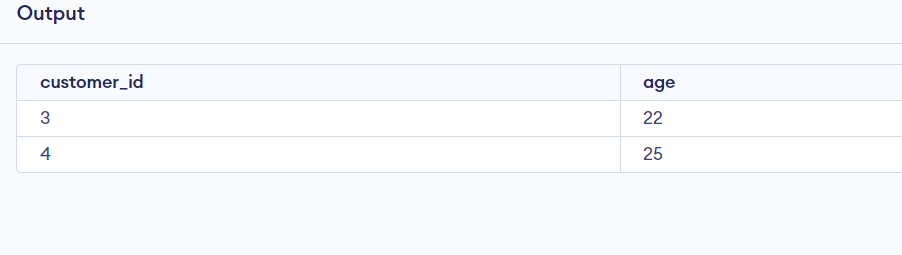
Customers Table:



* 1. Column present in select clause CODE:

select customer\_id,age from customers

where country='UK' order by age;

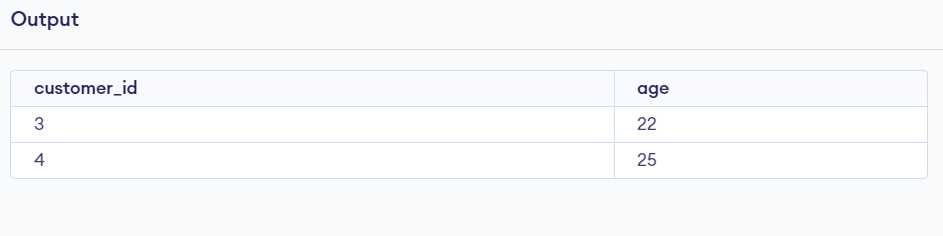


* 1. Using column number

CODE:

select customer\_id,age from customers

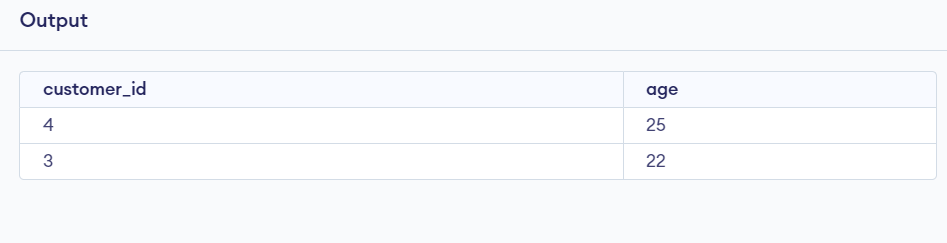
where country='UK' order by 2;



* 1. Descending order CODE:

select customer\_id,age from customers

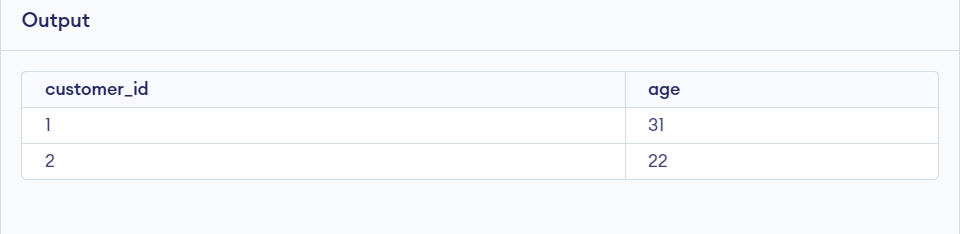
where country='UK' order by 2 DESC;



* 1. Column not part of select clause CODE:

select customer\_id,age from customers

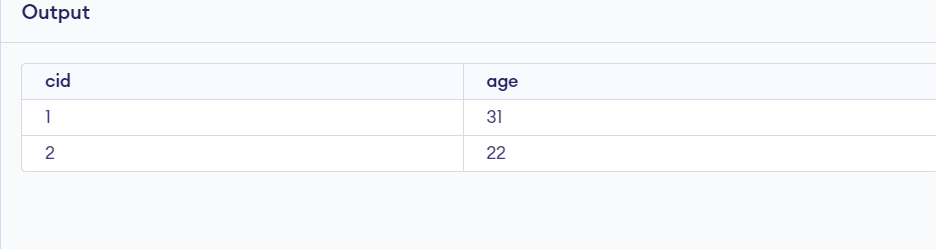
where country='USA' order by first\_name;



* 1. Using alias names CODE:

select customer\_id cid,age from customers

where country='USA' order by cid;

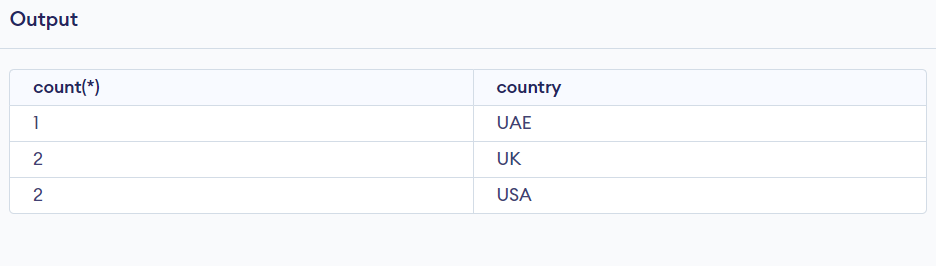


## GROUP BY CASES:-

1. Colum present in select clause CODE:

select count(\*),country from customers

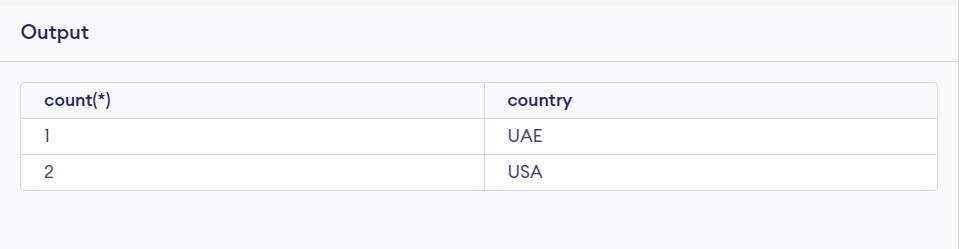
group by country;



1. Having clause CODE:

select count(\*),country from customers

group by country Having age>22;

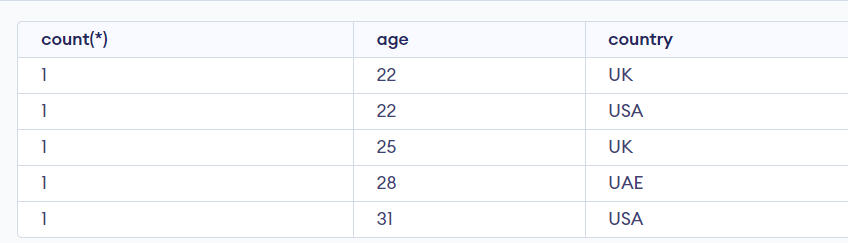


1. Grouping multiple columns CODE:

select count(\*),age,country from customers

group by age,country;

OUTPUT:

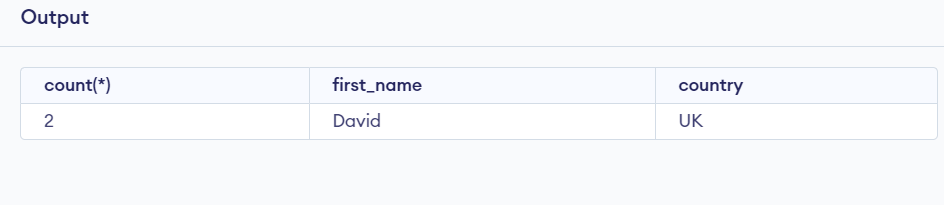


1. Complete SQL Query CODE:

select count(\*),first\_name,country from customers

where last\_name like 'R%' group by country

having age>20 order by age DESC;



## 1601-21733-055

**WEEK-6 EXERCISE-1**

## Department Table

QUERY:

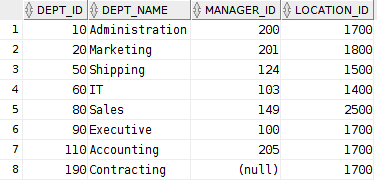
create table dept

(dept\_id number(4) Not Null, dept\_name varchar2(30) Not Null, manager\_id number(6), location\_id number(4));

insert into dept values(10,'Administration',200,1700); insert into dept values(20,'Marketing',201,1800); insert into dept values(50,'Shipping',124,1500); insert into dept values(60,'IT',103,1400);

insert into dept values(80,'Sales',149,2500); insert into dept values(90,'Executive',100,1700); insert into dept

values(110,'Accounting',205,1700); insert into dept values(190,'Contracting',null,1700);

select \* from dept; OUTPUT:

## Employees Table

QUERY:

create table employees (employee\_id number(6), first\_name varchar2(20), last\_name varchar2(25),

email varchar2(25), phone\_number varchar2(10), hire\_date date,

job\_id varchar2(10), salary number(8,2),

commission\_pct number(2,2), manager\_id number(6), department\_id number(4)

);

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(100,'King','AD\_PRES','17-jun-87');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(101,'kochhar','AD\_VP','21-sep-9');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(102,'De Haan','AD\_VP','13-jan-93');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(103,'Hunold','IT\_PROG','03-jan-90');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(104,'Emst','IT\_PROG','21-may-91');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(107,'Lorentz','IT\_PROG','07-feb-99');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(124,'Mourgos','ST\_MAN','17-oct-95');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(141,'Rajs','ST\_CLERK','17-oct-95');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(142,'Davis','ST\_CLERK','29-jan-97');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(143,'Matos','ST\_CLERK','15-mar-98');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(144,'Vertas','ST\_CLERK','09-jul-98');

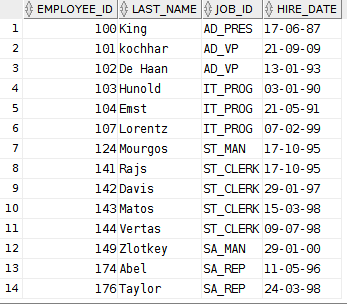
insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(149,'Zlotkey','SA\_MAN','29-jan-00');

insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(174,'Abel','SA\_REP','11-may-96');

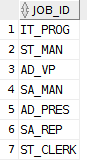
insert into employees(employee\_id, last\_name, job\_id, hire\_date) values(176,'Taylor','SA\_REP','24-mar-98');

select employee\_id, last\_name, job\_id, hire\_date from employees;

OUTPUT:

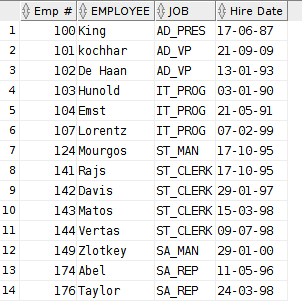


1. AIM: To display the unique jobs of employees. QUERY:

select distinct job\_id from employees; OUTPUT:

1. AIM: To display the column names with alias names. QUERY:

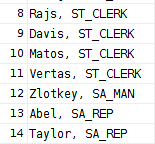
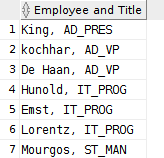
select employee\_id as "Emp #",last\_name as Employee,job\_id as Job,hire\_date as "Hire Date"

from employees; OUTPUT:

1. AIM:To concatenate the columns.

QUERY:

select last\_name||','||' '||job\_id as "Employee and Title" from employees;

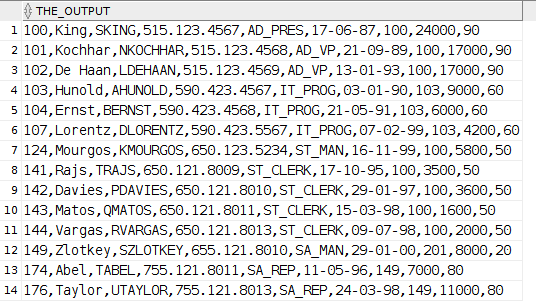
OUTPUT:

1. AIM:To combine all the columns and then display. QUERY:

selectemployee\_id||','||last\_name||','||email||','||phone\_number||','||job\_id||','||hire\_date||','||manage r\_id||','||salary||','||department\_id as THE\_OUTPUT

from employees;

OUTPUT:



1. AIM: To display last name and salary of employees earning salary more than 12,000/- QUERY:

select last\_name,salary from employees

where salary>12000; OUTPUT:

1. AIM: To display the employee last name and department number for employee number 176.

QUERY:

select last\_name,department\_id from employees

where employee\_id=176;

OUTPUT:



1. AIM: To display the employee last name, job ID, and start date of employees hired between February 20, 1998, and May 1, 1998 and order them.

QUERY:

select last\_name,job\_id,hire\_date from employees where(hire\_date between '20-Feb-98' and '01-May-98') order by hire\_date ASC;

OUTPUT:

1. AIM: To list the last name and salary of employees who earn between $5,000 and $12,000, and are in department 20 or 50.

QUERY:

select last\_name as Employee,salary as Monthly\_Salary from employees where((department\_id in (20,50)) and (salary between 5000 and 12000)); OUTPUT:

1. AIM:To display the last name and hire date of every employee who was hired in 1994. QUERY:

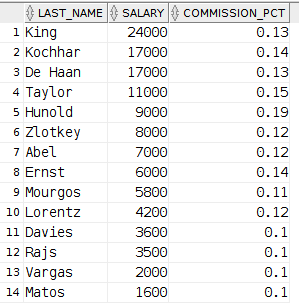
select last\_name,hire\_date from employees where(hire\_date like '%94');

OUTPUT: 

1. AIM:To Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

QUERY:

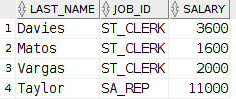
select last\_name,salary,commission\_pct from employees order by salary DESC,commission\_pct DESC;

OUTPUT:

1. AIM: To display the last name, job, and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to $2,500, $3,500, or $7,000. QUERY:

select last\_name,job\_id,salary from employees

where((job\_id in ('ST\_CLERK','SA\_REP')) and salary not in (2500,3500,7000)); OUTPUT:



EXERCISE:-

160121733055 DBMS LAB RECORD WEEK 6

Exercise 1. Answer each of the following questions briefly. The questions are based on the following relational schema:

Emp(eid: integer, ename: string, age: integer, salary: real) Works(eid: integer, did: integer, pcttime: integer)

Dept(did: integer, dname: string, budget: real, managerid: integer)

1. Give an example of a foreign key constraint that involves the Dept relation. What are the options for enforcing this constraint when a user attempts to delete a Dept tuple?
2. Write the SQL statements required to create the preceding relations, including appropriate versions of all primary and foreign key integrity constraints.
3. Define the Dept relation in SQL so that every department is guaranteed to have a manager.
4. Write an SQL statement to add John Doe as an employee with eid = 101, age = 32 and salary = 15, 000.
5. Write an SQL statement to give every employee a 10 percent raise.
6. Write an SQL statement to delete the Toy department. Given the referential integrity constraints you chose for this schema, explain what happens when this statement is executed.

## CODE

Creating tables:-

create table emplo (

eid number(4) primary key, ename char(20),

age number(2), salary number(10,2)

);

create table depart (

did number(4) primary key, dname char(5),

budget number(8),

managerid number(4) not null,

CONSTRAINT dept\_mid\_fk foreign key(managerid) references emplo(eid) on delete set null

);

create table works (

eid number(4), did number(4),

pcttime number(4),

constraint works\_eid\_fk foreign key(eid) references emplo(eid) on delete set null, constraint works\_did\_fk foreign key(did) references depart(did) on delete set null

);

Inserting JOHN DOE:-

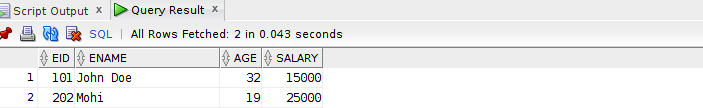
insert into emplo values(101,'John Doe',32,15000); insert into emplo values(202,'Mohi',19,25000);

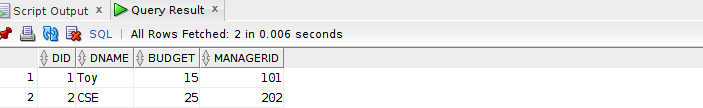
insert into depart values(1,'Toy',15,101); insert into depart values(2,'CSE',25,202);

TABLES:-

select \* from emplo;

select \* from depart;



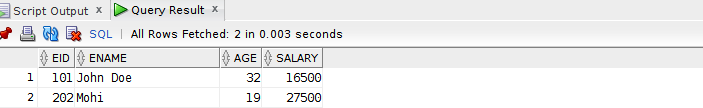


UPDATING SALARY BY 10%

update emplo

set SALARY=(salary)+(salary\*(0.1));

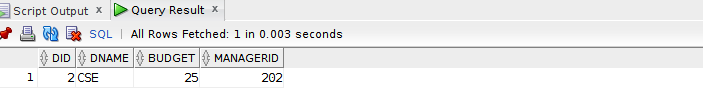
select \* from emplo;



DELETING TOY DEPARTMENT

delete from depart where dname='Toy';

select \* from depart;



Exercise 2. Consider the following relations:

Student(snum: integer, sname: string, major: string, level: string, age: integer) Class(name: string, meets at: string, room: string, fid: integer) Enrolled(snum: integer, cname: string)

Faculty(fid: integer, fname: string, deptid: integer)

The meaning of these relations is straightforward; for example, Enrolled has one record per studentclass pair such that the student is enrolled in the class. Write the following queries in SQL. No duplicates should be printed in any of the answers.

TABLES

create table stude (

snum char(10) primary key, sname char(10),

major char(15), levell char(2), age number(2)

);

CREATE TABLE class (

name char(5) primary key, meets char(10),

room char(5),

fid number(5),

constraint cls\_fid\_fk foreign key(fid) references faculty(fid)

);

create table faculty (

fid number(5) primary key, fname char(10),

did number(5)

);

create table enrolled (

snum char(10), cname char(5),

constraint enr\_snum\_fk foreign key(snum) references stude(snum), constraint enr\_cname\_fk foreign key(cname) references class(name)

);

insert into stude values(1,'a','History','JR',18); insert into stude values(2,'b','History','SR',19); insert into stude values(3,'c','DBMS','JR',18);

select \* from stude;

insert into faculty values(11,'aa',101); insert into faculty values(22,'bb',202);

select \*

from faculty;

insert into class values('abc','2','R128',11); insert into class values('def','3','R129',22); insert into class values('ghi','4','R130',11);

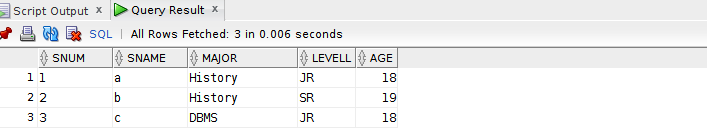
select \* from class;

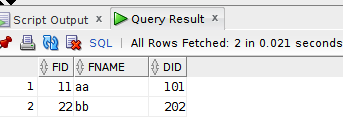
insert into enrolled values(1,'abc'); insert into enrolled values(2,'def');

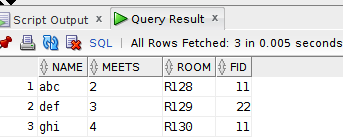
insert into enrolled values(3,'ghi');

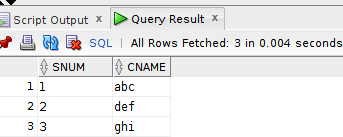
select \*

from enrolled;









1. Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach.

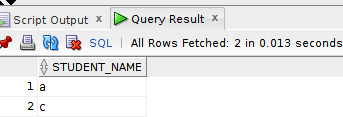
CODE

SELECT S.sname as Student\_Name

FROM Stude S, Class C, Enrolled E, Faculty F

WHERE S.snum = E.snum AND E.cname = C.name AND C.fid = F.fid AND F.fname = 'aa' AND S.levell = 'JR';

OUTPUT:-



1. Find the age of the oldest student who is either a History major or enrolled in a course taught by
2. Teach.

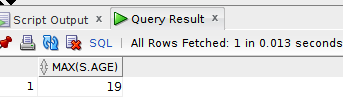
CODE

SELECT MAX(S.age)

FROM Stude S

WHERE (S.major = 'History') OR S.snum IN (SELECT E.snum

FROM Class C, Enrolled E, Faculty F,Stude S WHERE E.cname = C.name AND C.fid = F.fid AND F.fname = 'aa' );



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 8 | |
| **Course Code : 20CSC14**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**AIM:** Write the queries to demonstrate sub queries in SQL.

**DESCRIPTION**

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within clauses, most commonly in the WHERE clause. It is used to return data from a table, and this data will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow −

* Subqueries must be enclosed within parentheses.
* A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
* An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
* Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
* The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
* A subquery cannot be immediately enclosed in a set function.
* The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

# Subqueries with the SELECT Statement

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows −

SELECT column\_name [, column\_name ] FROM table1 [, table2 ]

WHERE column\_name OPERATOR (SELECT column\_name [, column\_name ] FROM table1 [, table2 ]

[WHERE])

160121733055

WEEK-7 SUB QUERIES

1. CODE:

SELECT \*

FROM orders

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

WHERE name='paul adam'); OUTPUT:



2. CODE:

select \* from orders

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

where city='london'); OUTPUT:



3. CODE:

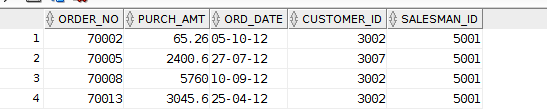
SELECT \*

FROM orders

WHERE salesman\_id =

(SELECT DISTINCT salesman\_id FROM orders

WHERE customer\_id =3007); OUTPUT:-



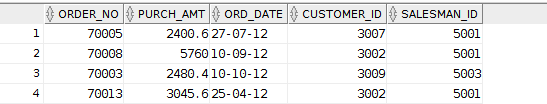
4. CODE:

SELECT \*

FROM orders WHERE purch\_amt >

(SELECT AVG(purch\_amt) FROM orders

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

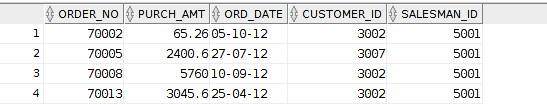


5. CODE:

select \* from orders

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

where city='new york'); OUTPUT:



6. CODE:

select commision from SALESPEOPLE

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

where city='Paris'); OUTPUT:



7. CODE:

SELECT \*

FROM customer WHERE customer\_id =

(SELECT salesman\_id -2001 FROM SALESPEOPLE

WHERE name = 'mc lyon'); OUTPUT:



8. CODE:

SELECT grade, COUNT (\*) FROM customer

GROUP BY grade HAVING grade >

(SELECT AVG(grade)

FROM customer

WHERE city = 'New York'); OUTPUT:

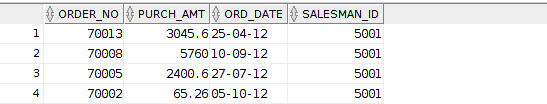


9. CODE:

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:

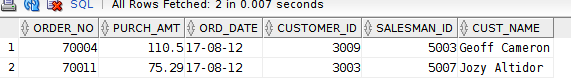


10. CODE:

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. CODE:

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

(SELECT COUNT(\*)

FROM customer

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

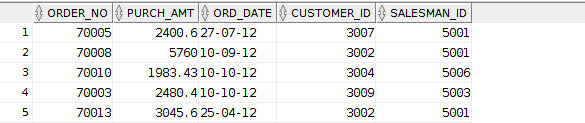


12. CODE:

select \* from orders

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

OUTPUT:



13. CODE:

SELECT \*

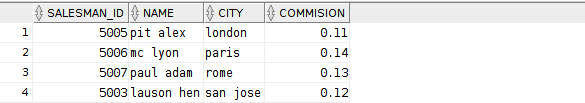
FROM SALESPEOPLE a WHERE 1 =

(SELECT COUNT(\*)

FROM customer

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

OUTPUT:



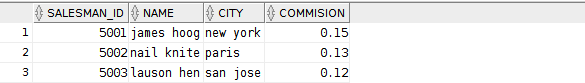
14. CODE:

SELECT \*

FROM SALESPEOPLE a WHERE 1 <

(SELECT COUNT(\*) FROM ORDERS

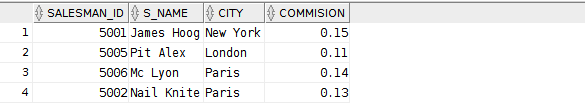
WHERE salesman\_id=a.salesman\_id); OUTPUT:



15. CODE:

select \* from sales

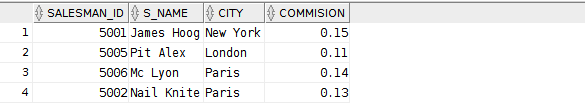
where city in (select city from customer); OUTPUT:



16. CODE:

select \* from sales

where city in (select city from customer); OUTPUT:



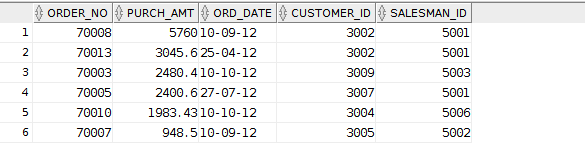
17. CODE:

SELECT \*

FROM orders

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

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



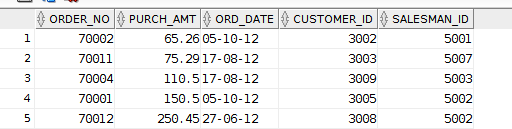
18. CODE:

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:



19. CODE:

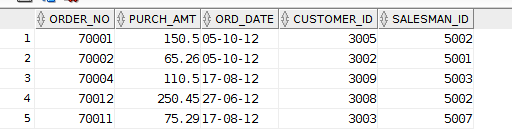
SELECT \*

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');

OUTPUT:

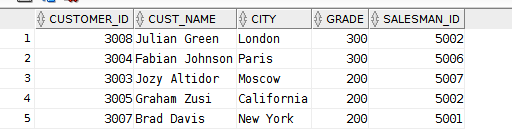


20. CODE:

select \*

from CUSTOMER

where grade>ANY ( select grade from customer

where city='New York'); OUTPUT:

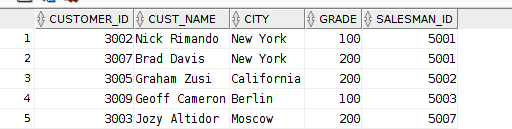
21. CODE:

select \*

from CUSTOMER

where grade<>ANY ( select grade from customer

where city='London'); OUTPUT:



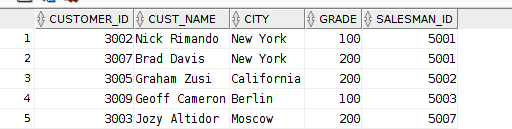
22. CODE:

select \*

from CUSTOMER

where grade<>ANY ( select grade from customer

where city='Paris'); OUTPUT:



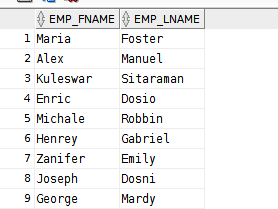
23. CODE:

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

WHERE emp\_dept IN (SELECT dpt\_code

FROM emp\_dept

WHERE dpt\_allotment > 50000); OUTPUT:



24. CODE:

SELECT \*

FROM emp\_dept WHERE dpt\_allotment > (

SELECT AVG(dpt\_allotment) FROM emp\_dept

); OUTPUT:



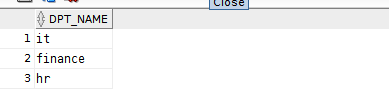
25. CODE:

select dpt\_name from emp\_dept

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

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

OUTPUT:



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 9 | |
| **Course Code : 20CSC14**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

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

**DESCRIPTION**

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.

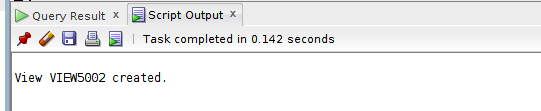
160121733055 WEEK-9 VIEWS

* 1. Creating view CODE:

create view view5002 as

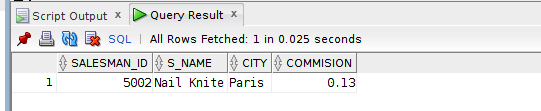
select \* from sales

where SALESMAN\_ID=5002; OUTPUT:



select \*

from view5002;



ALIASING CODE:

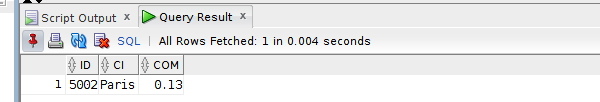
create view view1 as

select salesman\_id as id,city as ci,commision as com from sales

where SALESMAN\_ID=5002;

select \* from view1;

OUTPUT:



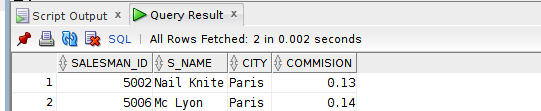
* 1. Inserting views CODE:

create view vp as

select salesman\_id,s\_name,city,commision from sales

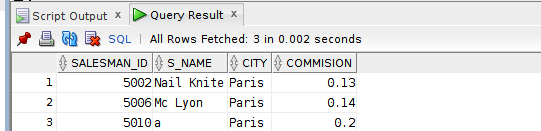
where city='Paris';

select \* from vp;

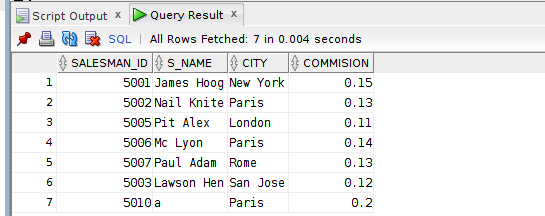


insert into vp values(5010,'a','Paris',0.20);

select \* from vp;



BASE TABLE:



* 1. Update CODE:

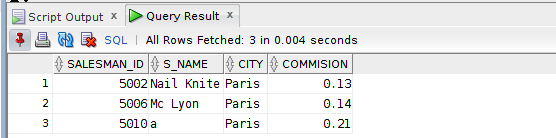
update vp

set COMMISION=0.21

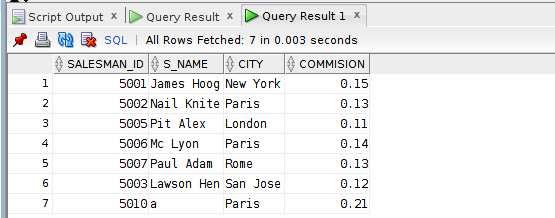
where SALESMAN\_ID=5010;

select \* from vp;

OUTPUT:



BASE TABLE:



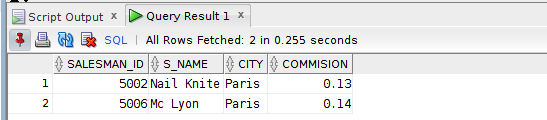
* 1. DELETE CODE:

delete from vp

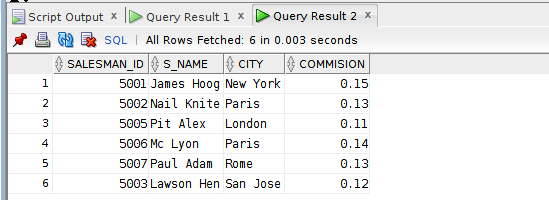
where SALESMAN\_ID=5010;

select \* from vp;

OUTPUT:

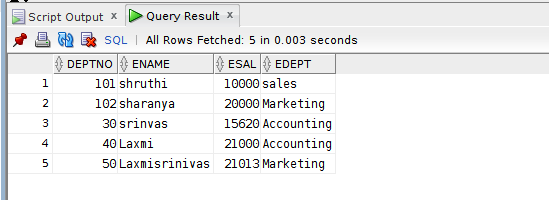


BASE TABLE:



* 1. INSERT AND DELETE CANNOT PERFORM CASES: CODE:

BASE TABLE:



create view vd as

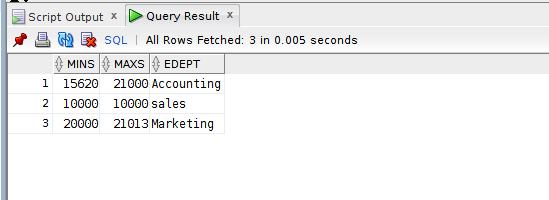
select min(esal) mins,max(esal) maxs,edept from emp

group by edept;

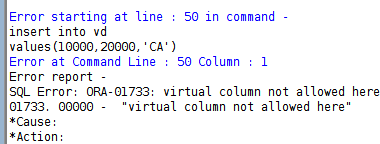
select \* from vd;

VIEW:

1. CONTAINING GROUP BY FUNCTION AND CLAUSE

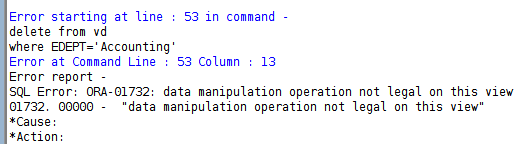


insert into vd values(10000,20000,'CA');



delete from vd

where EDEPT='Accounting';



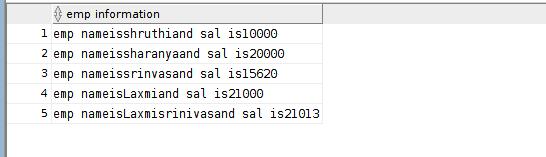
1. HAVING EXPRESIONS IN SELECT

CODE:

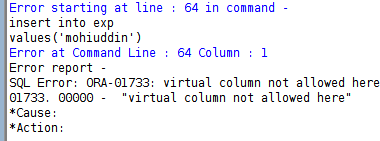
create view exp as

select 'emp name'||'is'||ename||'and sal is'||esal as "emp information" from emp;

select \* from exp;



insert into exp values('mohiuddin');



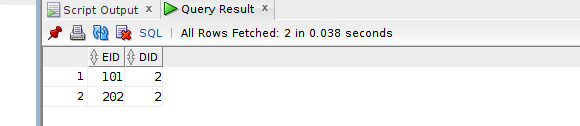
1. MULTIPLE TABLES create view ed

as

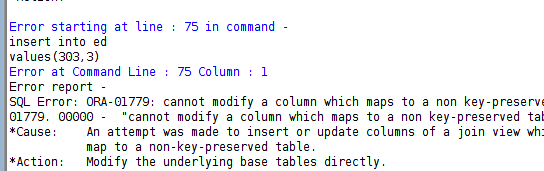
select eid,did

from emplo,depart;

select \* from ed;



insert into ed values(303,3);



* 1. WITH CHECK OPTION CODE:

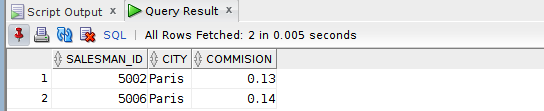
create view vpar as

select salesman\_id,city,commision from sales

where city='Paris' with check option;

select \* from vpar;

OUTPUT:



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 10 | |
| **Course Code : 20CSC14**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**AIM:** Implementation of PL/SQL queries and Implicit cursors.

**THEORY:**

PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL. PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java.

* Applications written in PL/SQL are fully portable.
* PL/SQL provides high security level.
* PL/SQL provides access to predefined SQL packages.
* PL/SQL provides support for Object-Oriented Programming.
* PL/SQL provides support for developing Web Applications and Server Pages.

Syntax:

Declare

-variables, cursors, user defined exceptions

Begin

-SQL statements

-PL/SQL statements

Exception (optional)

-Action performed

End;

*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: -*

* *Implicit cursors*
* *Explicit cursors*

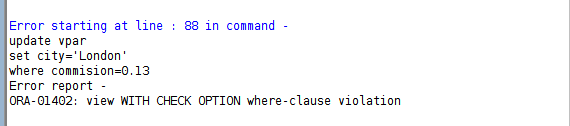
***Implicit cursors: -***

* *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.*

update vpar

set city='London'

where commision=0.13; OUTPUT:



* 1. WITH READ OPTION CODE:

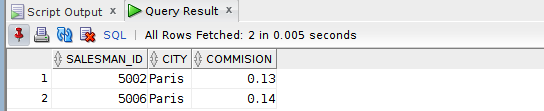
create view vpar2 as

select salesman\_id,city,commision from sales

where city='Paris' with read only;

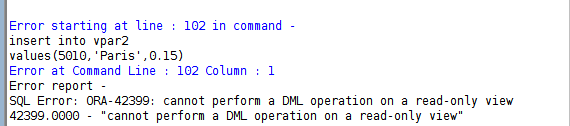
select \* from vpar2;

VIEW:



insert into vpar2 values(5010,'Paris',0.15);

OUTPUT:



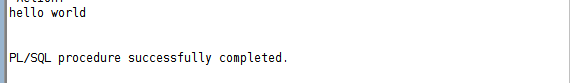
PL/SQL

1. printing hello world CODE:

set serveroutput on; begin

dbms\_output.put\_line('hello world'); end;

OUTPUT:



1. DATA FROM DB SERVER CODE:

set serveroutput on; declare

vn sales.s\_name%type; vc sales.city%type;

begin

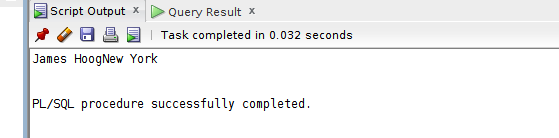
select s\_name,city into vn,vc

from sales

where salesman\_id=5001; dbms\_output.put\_line(vn||vc);

end;

OUTPUT:



MULTIPLE TUPLE ERROR:

set serveroutput on; declare

vn sales.s\_name%type; vc sales.city%type;

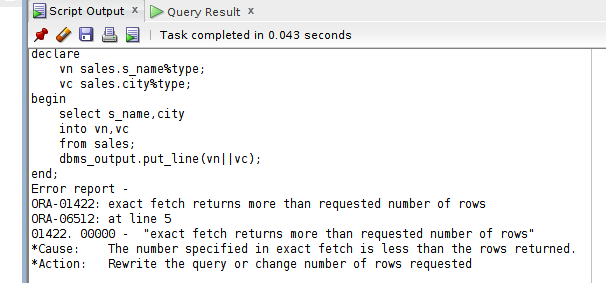
begin

select s\_name,city into vn,vc

from sales; dbms\_output.put\_line(vn||vc);

end;

ERROR:



INSERT PL/SQL TO SQL: CODE:

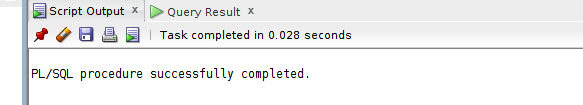
set serveroutput on; begin

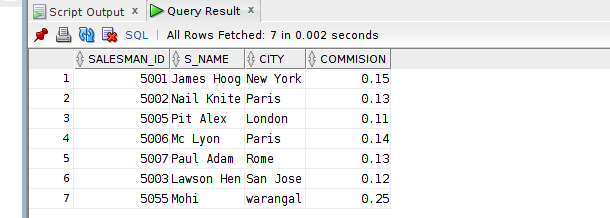
insert into sales values(5055,'Mohi','warangal',0.25);

end;

select \* from sales;

OUTPUTS:





UPDATE CODE:

set serveroutput on; begin

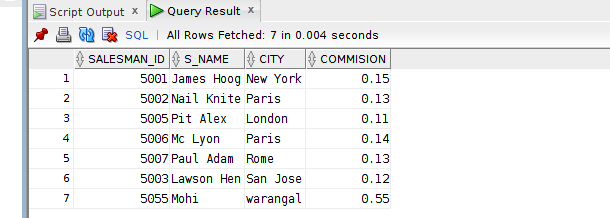
update sales

set commision=0.55 where salesman\_id=5055;

end;

select \* from sales;

OUTPUT:



DELETE CODE:

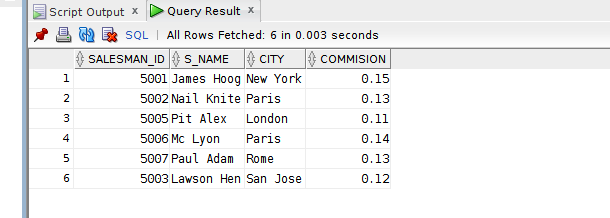
set serveroutput on; begin

delete from sales

where salesman\_id=5055; end;

select \* from sales;

OUTPUT:



IMPLICIT CURSORS:

%rowcount CODE:

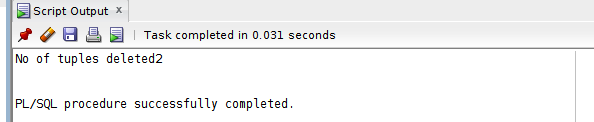
set serveroutput on; begin

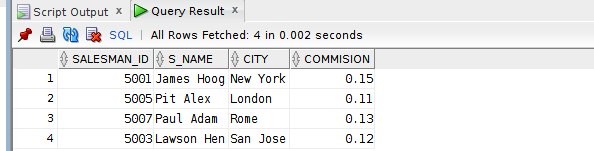
delete from sales where city='Paris';

dbms\_output.put\_line('No of tuples deleted'||SQL%rowcount); end;

select \* from sales;

OUTPUTS:





CONTROL STRUCTURES

1. IF ETHEN ELSIF ENDIF CODE:

set serveroutput on;

DECLARE

var number(3) := 50; BEGIN

IF (var = 10) THEN

dbms\_output.put\_line('Value of var is 10');

ELSIF (var = 20) THEN

dbms\_output.put\_line('Value of var is 20');

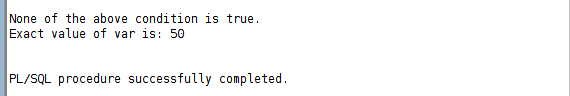
ELSIF (var = 30) THEN

dbms\_output.put\_line('Value of var is 30'); ELSE

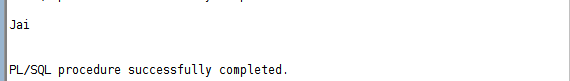
dbms\_output.put\_line('None of the above condition is true.'); END IF;

dbms\_output.put\_line('Exact value of var is: '|| var); END;

OUTPUT:



1. SWITCH CODE:



1. EXIT LOOP CODE:

set serveroutput on;

DECLARE

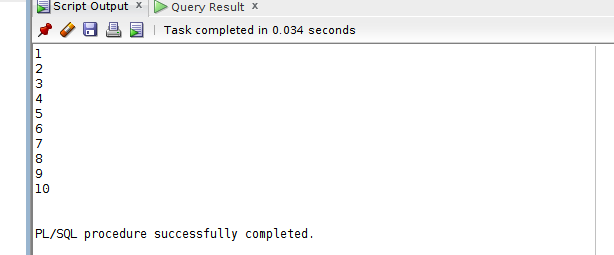
num NUMBER := 1; BEGIN

LOOP DBMS\_OUTPUT.PUT\_LINE(num); IF num = 10 THEN

EXIT; END IF;

num := num+1; END LOOP; END;

OUTPUT:



1. WHILE LOOP CODE:

set serveroutput on;

DECLARE

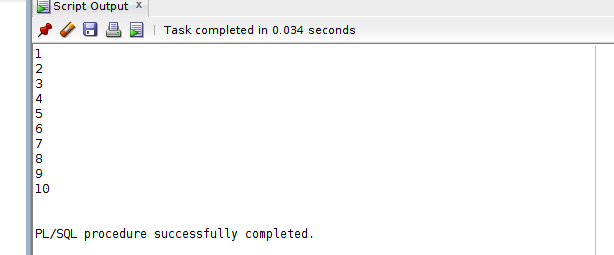
num NUMBER := 1; BEGIN

WHILE num <= 10 LOOP

DBMS\_OUTPUT.PUT\_LINE(num);

num := num+1; END LOOP;

END; OUTPUT:



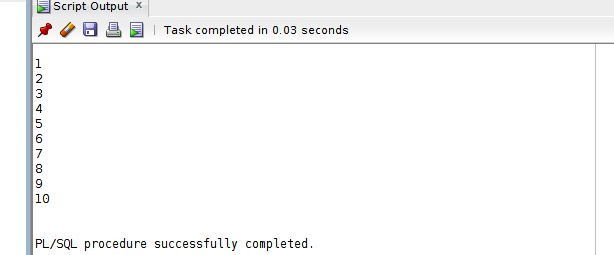
1. FOR LOOP CODE:

set serveroutput on; DECLARE BEGIN

FOR var IN 1..10 LOOP

DBMS\_OUTPUT.PUT\_LINE(var); END LOOP;

END; OUTPUT:



COMPOSITE DATA TYPE

1. %rowtype CODE:

set serveroutput on; declare

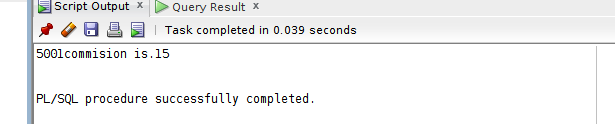
sr sales%rowtype; begin

select \* into sr from sales

where salesman\_id=5001; dbms\_output.put\_line(sr.salesman\_id||'commision is'||sr.commision);

end;

OUTPUT:



## SWITCH CASE CODE:

set serveroutput on;

declare

grade char(1) := 'J';

BEGIN

CASE grade

when 'A' then dbms\_output.put\_line('Afnan'); when 'B' then dbms\_output.put\_line('Billa');

when 'C' then dbms\_output.put\_line('Cheeku'); when 'M' then dbms\_output.put\_line('Mohi'); when 'J' then dbms\_output.put\_line('Jai');

else dbms\_output.put\_line('No such name'); END CASE;

End;

## IMPLICIT CURSORS

**%Found and %NotFound CODE:-**

DECLARE

total\_rows number(2); BEGIN

UPDATE sales

SET commision = commision + 0.5;

IF sql%notfound THEN

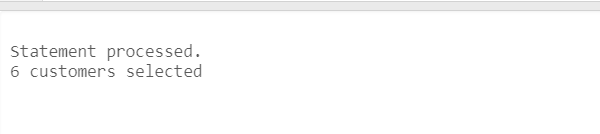
dbms\_output.put\_line('no customers selected');

ELSIF sql%found THEN total\_rows := sql%rowcount;

dbms\_output.put\_line( total\_rows || ' customers selected '); END IF;

END;

## OUTPUT:-



**COMPOSITE DATA TYPE**

## Record:

CODE:-

declare

type sa is record (

id sales.salesman\_id%type, name sales.s\_name%type

);

sale sa; begin

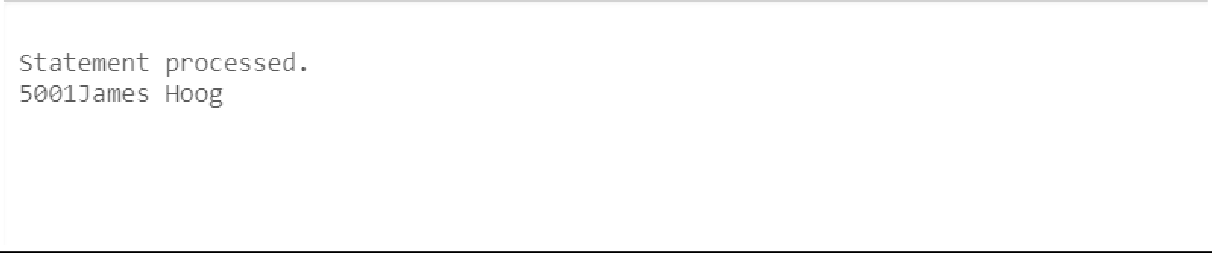
select salesman\_id,s\_name into sale

from sales

where salesman\_id=5001;

dbms\_output.put\_line(sale.id||sale.name); end;

OUTPUT:-



## PROCEDURES:-

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 11 | |
| **Course Code : 20CSC14**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**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>]*

## SIMPLE PROCEDURE

CODE:-

CREATE PROCEDURE hello AS

BEGIN

dbms\_output.put\_line('Hello World!'); END;

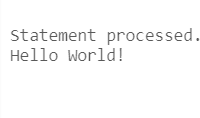


CALLING ENVIRONMENT:

begin

hello;

end;



## USING IN, OUT MODES

CODE:-

create PROCEDURE Min(x IN number, y IN number, z OUT number) as

BEGIN

IF x < y THEN

z:= x; ELSE

z:= y;

END IF;

END;

CALLING ENVIRONMENT:

DECLARE

a number; b number; c number;

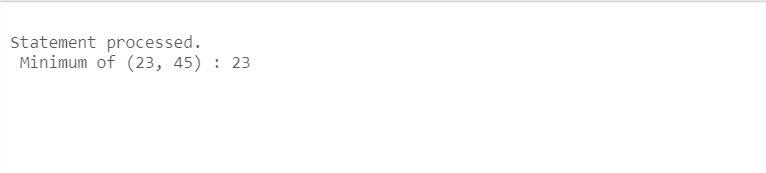
BEGIN a:= 23;

b:= 45;

Min(a, b, c);

dbms\_output.put\_line(' Minimum of (23, 45) : ' || c); END;

OUTPUT:-



## INOUT MODE

CODE:-

create PROCEDURE squareNum(x IN OUT number) AS BEGIN

x := x \* x; END;

CALLING ENVIRONMENT:- DECLARE

a number;

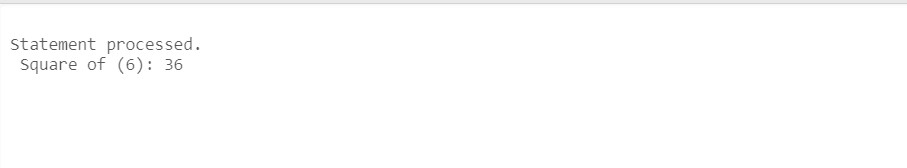
BEGIN

a:= 6;

squareNum(a);

dbms\_output.put\_line(' Square of (6): ' || a);

END; OUTPUT:-



## FUNCTIONS:- CODE:-

create FUNCTION FindMax(x IN number, y IN number)

RETURN number as

z number; BEGIN

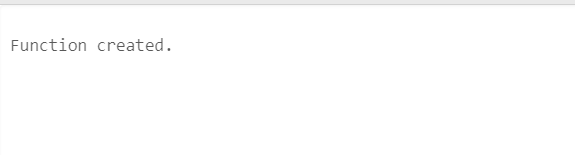
IF x > y THEN

z:= x; ELSE

Z:= y;

END IF;

RETURN z; END;



CALLING ENVIRONMENT: DECLARE

a number; b number; c number;

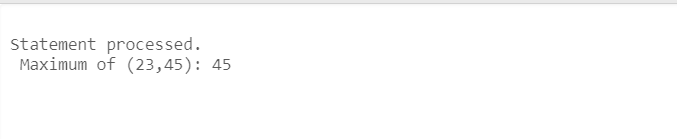
BEGIN a:= 23;

b:= 45;

c := findMax(a, b);

dbms\_output.put\_line(' Maximum of (23,45): ' || c); END;

## OUTPUT:-



**RECURSIVE FUNCTION CODE:-**

create FUNCTION fact(x number)

RETURN number as

f number; BEGIN

IF x=0 THEN f := 1;

ELSE

f := x \* fact(x-1); END IF;

RETURN f; END;

CALLING ENVIRONMENT: DECLARE

num number;

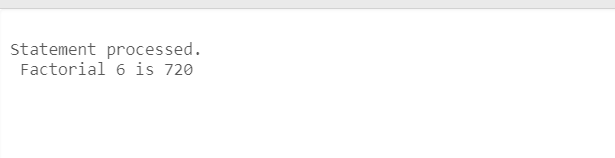
factorial number; BEGIN

num:= 6;

factorial := fact(num);

dbms\_output.put\_line(' Factorial '|| num || ' is ' || factorial); END;

## OUTPUT:-



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 12 | |
| **Course Code : 20CSC14**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**AIM:** write the queries for explicit cursors 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: -*

* *Implicit cursors*
* *Explicit cursors*

***Explicit cursors: -***

*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.*
    - *So, close will be enabled in two cases:*
      * *When active set is empty*
      * *When active set is pointing to the last row in the active set.*

*SynTax: close <cursor name>;*

**QUERIES:-**

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

Declare

begin

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;

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.

**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; **Output:** 1

open emp\_det ;

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

PL/SQL procedure successfully completed.

**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

end if;

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

end;

**Output:**

close emp\_det;

NOT found is false

PL/SQL procedure successfully completed.

**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

end if;

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

end;

**Output:**

close emp\_det;

found is true

PL/SQL procedure successfully completed.

**5.Write a pl/sql block using parameterized cursor**

declare

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

cursor emp\_det (veid employees. Emp\_id %type) is

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

begin

open emp\_det(174) ;

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

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

dbms\_output.put\_line ('1.Name is : '||vname||',Job id is : '||vjid);

else

end if;

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

end;

**Output:**

close emp\_det;

found is true

1. Name is : able,Job id is : sa\_rep

PL/SQL procedure successfully completed.

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 13 | |
| **Course Code : 20CSC14**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**AIM:** write the queries for exception handling 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:**

***Exception handling:***

**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

begin

id number(10); cursor c1 is

select employee\_id from employees;

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

begin

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

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

begin

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

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

Begin

Var1 number (3):=0;

Var2 number (4):=100;

vsal employees.salary%type;

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

begin

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

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

begin

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

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.

1. **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

1. **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;

**Output:**

open c1;

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

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

end if;

declare

\*

ERROR at line 1:

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