# INDEX

|  |  |  |
| --- | --- | --- |
| S.NO | PROGRAM | PAGE.NO |
| 1 | DDL statements and DML statements | 2-6 |
| 2 | Write SQL queries for student and employee database. | 7-17 |
| 3 | Single row functions | 18-30 |
| 4 | Group functions, Constraints, and Joins | 31-50 |
| 5 | Sub-Queries | 51-58 |
| 6 | Demonstrate Views | 59-61 |
| 7 | Use of PL/SQL | 62-66 |
| 8 | Cursors | 67-71 |
| 9 | Procedures and Functions | 72-78 |
| 10 | Exceptional Handling | 79-81 |

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 1 | |
| **Course Code : 20CSC17**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**Aim:** To execute DDL statements:

# 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;* **QUERIES:**

# To create a table salesman

create table salesman

( salesman\_id number(6), sname varchar2(20),

city varchar(30), commission float(5) );

# Table Created

1. **Altereing the table Query:**

alter table salesmans add salary number(5); select distinct \*

from salesmans;

# Output:

Table SALESMANS altered.

A close up of a sign  Description automatically generated

# Modifing a column Query:

alter table salesmans modify salary number(4);

# Output:

Table SALESMANS altered.

# Droping a table Query:

drop table salesmans; select distinct \*

from salesmans;

# Output:

Table SALESMANS dropped.

ORA-00942: table or view does not exist 00942. 00000 - "table or view does not exist"

# Renaming values to the new column

Query:

update salesman set salary=1200

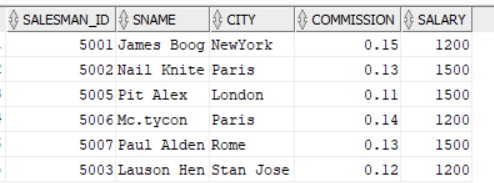
where salesman\_id in (5001,5006,5003);

update salesman set salary=1500

where salesman\_id in (5002,5007,5005);

select distinct \* from salesman;

# Output:



**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];*

# Queries: Creating a table:

create table student( roll\_no int,

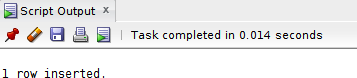
name varchar(35),

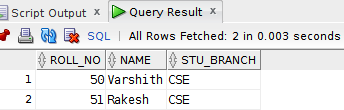
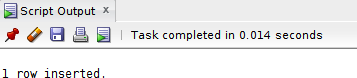
stu\_branch varchar(20)

);

# Inserting values :

insert into student values(50,'Varshith',’CSE’); insert into student values(51,'Rakesh',’CSE’); **Output :**



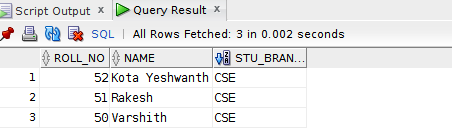


# Updating :

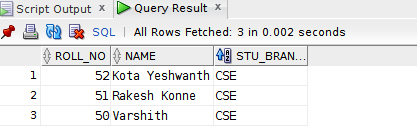
Update student

name = ‘Rakesh Konne; Where roll\_no= 51 Select \* from student; **Output :**

# Before update:



**After update:**

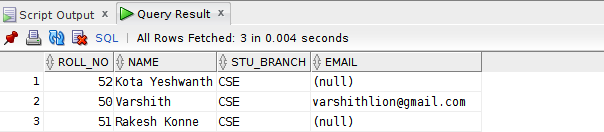
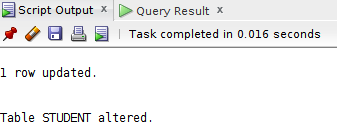


# Alter & Update:

Alter table student add email VARCHAR(35);

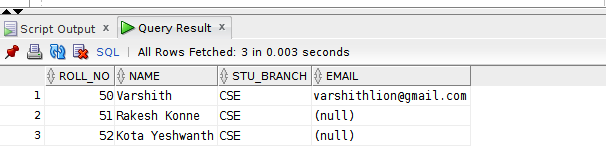
UPDATE student set [email=varshithlion@gmail.com](mailto:email%3Dvarshithlion@gmail.com) where( roll\_no=50)

# Output:



**Orderby:**

# Select \* from student orderby roll\_no Output:



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 2 | |
| **Course Code : 20CSC17**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

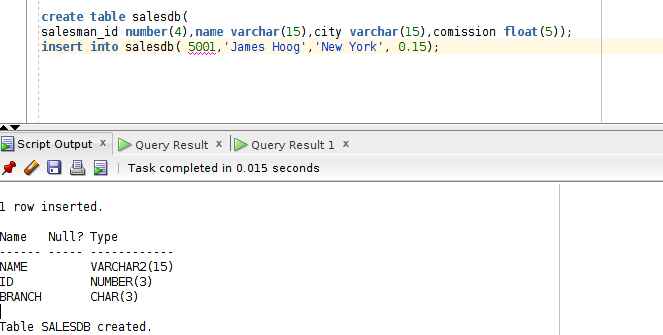
**AIM: To write queries using DDL and DML statements for employee,salesman**

# PROGRAMS:

**Question – 1:**

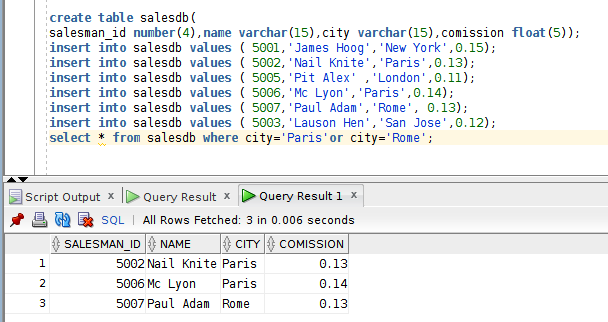
# Create a sales database with the given data :

**Code & Output:**



# Question – 2: write a query to display the salesman from city Paris or Rome from the given salesman database.

**Code & Output:**



# Graphical user interface, text Description automatically generatedQuestion – 3: write a query to display the sales person who are not from either Rome not Paris. Code & Output:

**Create a database for customer data Code:**

create table customerdb(

*customer\_id number(4), cust\_name varchar(15), city varchar(15), grade number(6),salesman\_id number(15));*

*insert into customerdb values ( 3002,'Nick Rimando','New York',100,5001); insert into customerdb values (3007,'Brad Davis','New York',200,5001);*

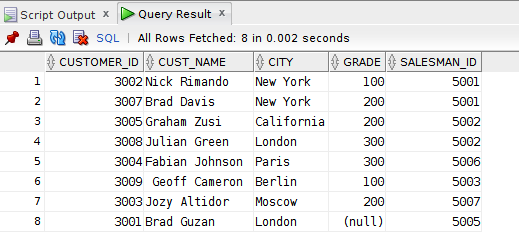
*insert into customerdb values ( 3005,'Graham Zusi','California',200,5002); insert into customerdb values (3008,'Julian Green','London ',300,5002);*

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

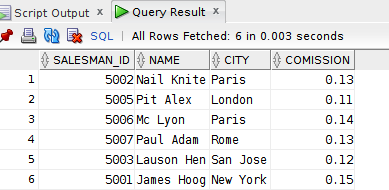
*insert into customerdb values (3009,' Geoff Cameron ','Berlin',100,5003);*

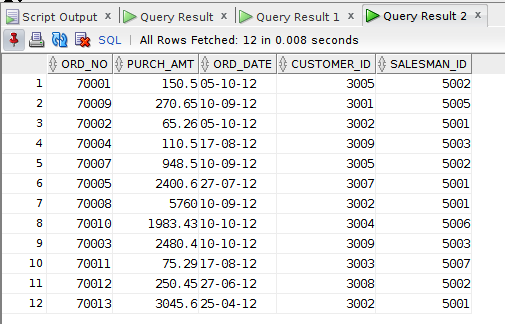
*insert into customerdb values ( 3003,'Jozy Altidor ','Moscow ',200,5007); insert into customerdb values (3001,'Brad Guzan ','London',200,5005); select \* from customerdb*

***Output:***

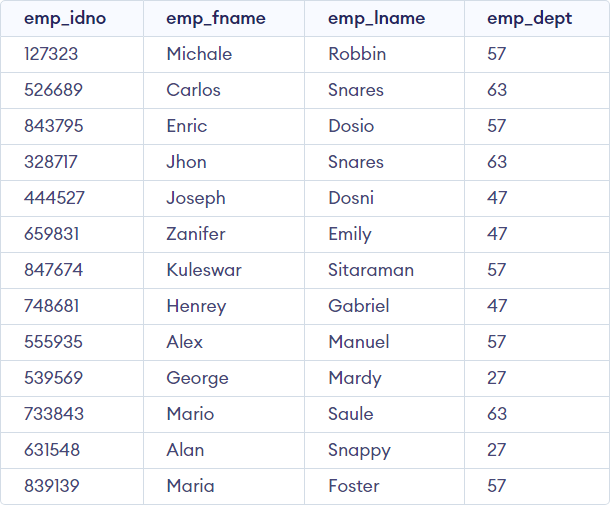


# sales table



***Orders table***

# Emp\_details database:



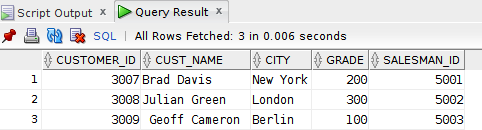
***4. From the following table, write a SQL query to retrieve the details of all***

***customers whose ID belongs to any of the values 3007, 3008 or 3009. Return customer\_id, cust\_name, city, grade, and salesman\_id.***

***Code:***

*select \* from customerdb where CUSTOMER\_ID between 3007 and 3009;*

# Output:

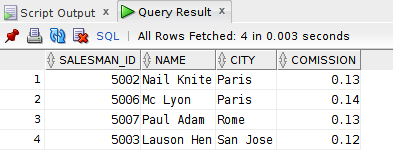


***5 From the following table, write a SQL query to find salespeople who receive commissions between 0.12 and 0.14 (begin and end values are included). Return salesman\_id, name, city, and commission***

***Code:***

*select \* from salesdb where comission between 0.12 and 0.14;*

***Output:***

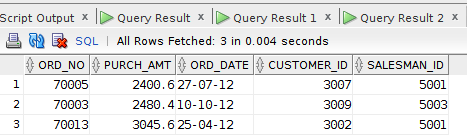


***6.***

***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:***

*select \*from orders where purch\_amt between 500 and 4000 and purch\_amt NOT In ( 948.50 ,1983.43);*

***Output:***

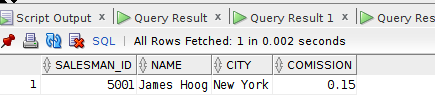
***7.***

***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 SALESDB where name between 'A' and 'L';*

***Output:***

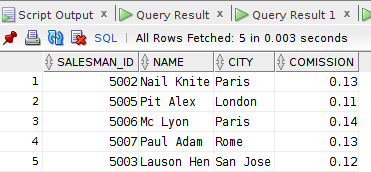


***8. From the following table, write a SQL query to find the details of all salespeople except those whose names begin with any letter between 'A' and 'L' (not inclusive). Return salesman\_id, name, city, commission.***

***Code:***

*select \* from SALESDB where name NOT BETWEEN 'A'AND 'L';*

***Output:***



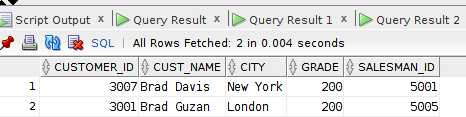
***9.***

***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 CUSTOMERDB where CUST\_NAME like 'B%';*

***Output:***



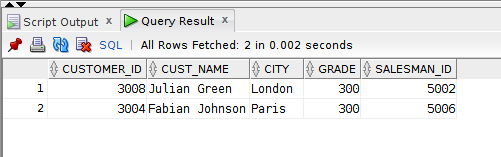
1. ***From the following table, write a SQL query to find the details of the***

***customers whose names end with the letter 'n'. Return customer\_id, cust\_name, city, grade, salesman\_id***

***Code:***

*select \* from CUSTOMERDB where CUST\_NAME like '%n';*

***Output:***

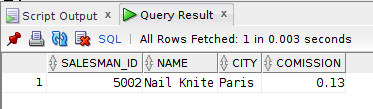


1. ***From the following table, write a SQL query to find the details of those salespeople whose names begin with ‘N’ and the fourth character is 'l'. Rests may be any character. Return salesman\_id, name, city, commission.***

***Code:***

*select \* from CUSTOMERDB where cust\_name like '%n';*

***Output:***

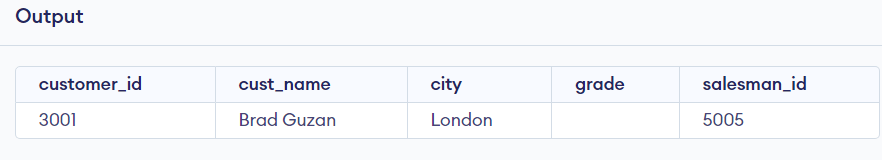


1. ***From the***

***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:***

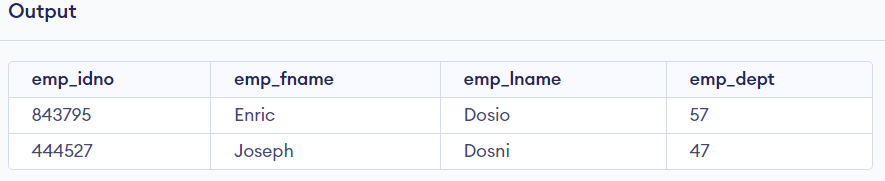
*select \* from customerdb where grade is NULL;*



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:***

*select \* from emp\_details where emp\_lname like "d%";*



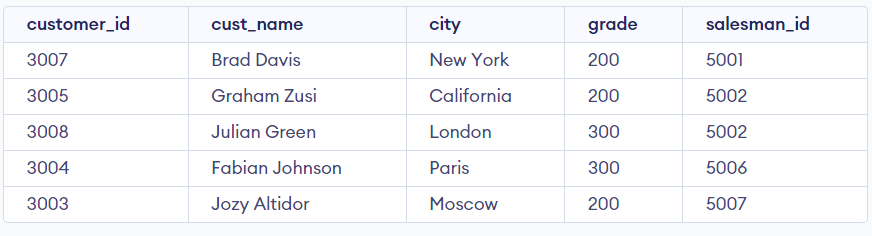
# Week-3(exercise-Operators)

* 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.**

# Code:

select \* from customerdb where grade>100;

# Output:

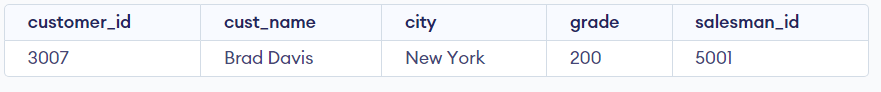


* 1. **From the following table, write a SQL query to find all the customers in ‘New York’ city who have a grade value above 100. Return customer\_id, cust\_name,city, grade, and salesman\_id.**

# Code:

select \* from customerdb where grade>100 and city="New York";

# Output:

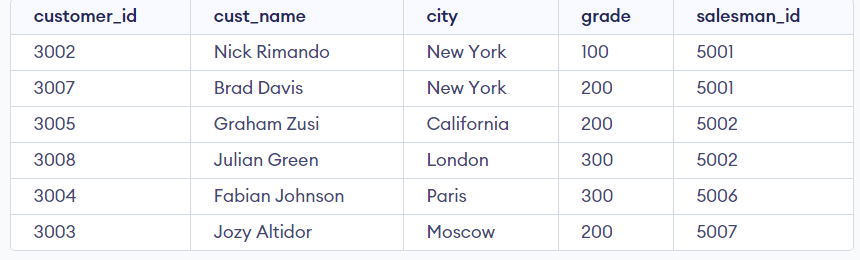


* 1. **From the following table, write a SQL query to find customers who are from the city of New York or have a grade of over 100. Return customer\_id, cust\_name,city, grade, and salesman\_id.**

# Code:

select \* from customerdb where grade>100 or city="New York";

# Output:

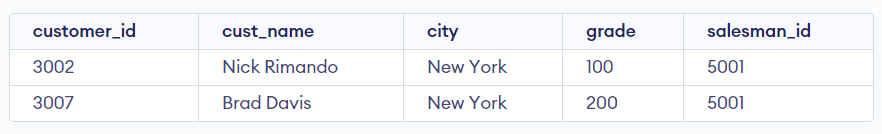


* 1. **From the following table, write a SQL query to find customers who are either from the city 'New York' or who do not have a grade greater than 100. Return customer\_id, cust\_name, city, grade, and salesman\_id**.

# Code:

select \* from customerdb where grade<100 or city="New York";

# Output:

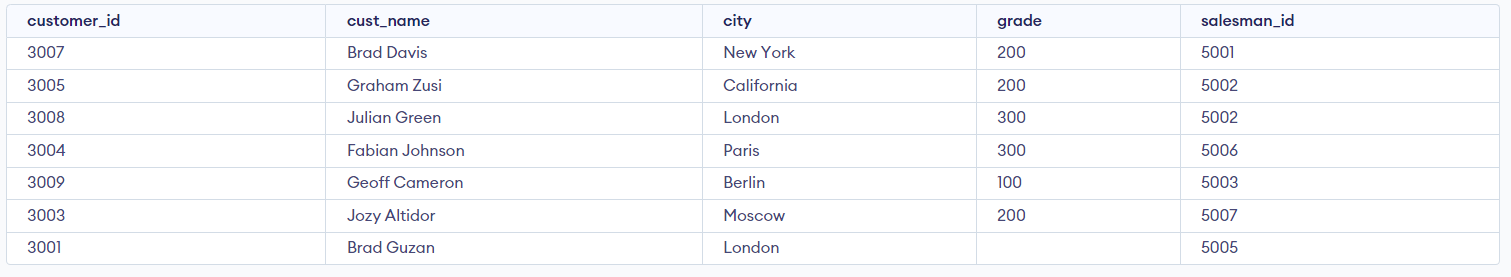


* 1. **From the following table, write a SQL query to identify customers who do not belong to the city of 'New York' or have a grade value that exceeds 100. Return customer\_id, cust\_name, city, grade, and salesman\_id.**

# Code:

select \* from customerdb where grade>100 or city<>"New York";

# Output:

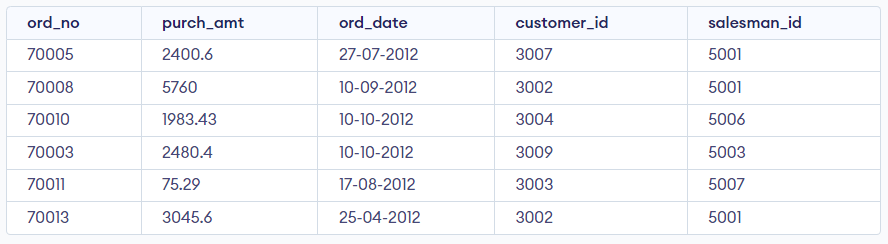


* 1. **From the following table, write a SQL query to find details of all orders excluding those with ord\_date equal to '2012-09-10' and salesman\_id higher than 5005 or purch\_amt greater than 1000.Return ord\_no, purch\_amt, ord\_date,customer\_id and salesman\_id.**

# Code:

select \* from orders where ord\_date<>'10-09-2012' and salesman\_id>5005 or purch\_amt>1000;

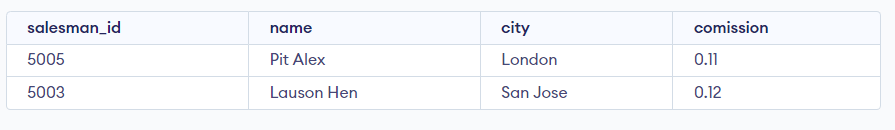
# Output:



* 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.**

# Code:

Select \* from salesdb where comission BETWEEN 0.10 AND 0.12; Output:

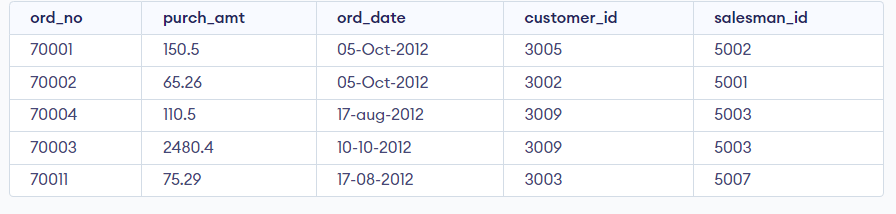


# From the following table, write a SQL query to find details of all orders with a purchase amount less than 200 or orders with an order date greater than or equal to '2012-02-10' and a customer ID less than 3009. Return ord\_no, purch\_amt, ord\_date, customer\_id and salesman\_id.

**Code:**

select \* from orders where purch\_amt<200 or (ord\_date < '2012-02-10' 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.**

# Code:

select \* from orders where not(ord\_date = '2012-08-17' OR (customer\_id > 3005 AND purch\_amt < 1000))

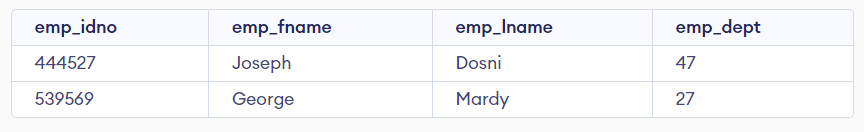
# Output:

* 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.**

# Code:

select \* from emp\_details where emp\_lname="Dosni" or emp\_lname="Mardy";

# Output:

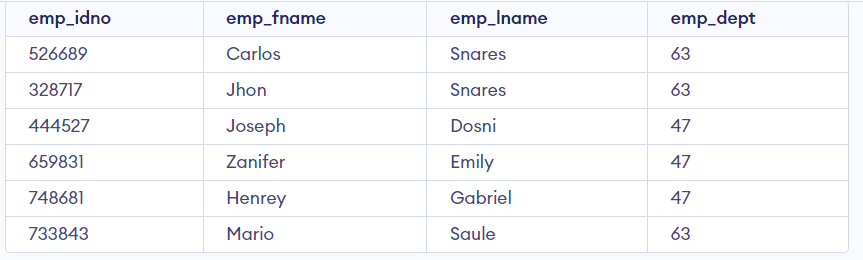


* 1. **From the following table, write a SQL query to find the employees who work at depart 47 or**

# 63. Return emp\_idno, emp\_fname, emp\_lname, and emp\_dept. Code:

select \* from emp\_details where emp\_dept='47' or emp\_dept='63';

# Output:



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 3 | |
| **Course Code : 20CSC17**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**AIM:** To write queries using 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)

# PROGRAMS:

**creating a table for single row functions**

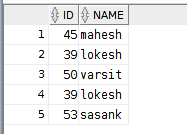
# query:

create table names(id number,name varchar(6)); insert into names values(45,'mahesh');

insert into names values(50,'varsit'); insert into names values(39,'lokesh');

insert into names values(53,'sasank'); select \* from names;

# Output:



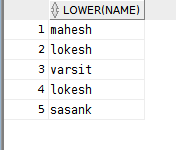
**Case manipulation functions:**

# lower function:

**Query1:**

select lower(name) from names;

# Output:



**Query2:**

select lower(name) from names where name='mahesh';

# Output:

1. **Upper function:**

# Query1:

select upper(name) from names;

# Output:

**Query2:**

select uppet(name) from names where name='mahesh';

# Output:

1. **InitCap:**

# query1: Output:

select INITCAP(name) from names ;

# Query2:

select INITCAP(name) from names where name='mahesh';

# Output:

**Character functions:**

# length

**Query1:**

select length(name) from names where name='mahesh';

# output:

**query1:** select length(name) from names;

# Output:

1. **concat**

**Query1:** select concat(id,name) from names;

# Output:

**Query2:** select concat(id,name) from names where id=45;

# Output:



1. **instr**

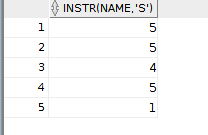
**Query1:** select instr(name,'s') from names where id=45;

# Output:

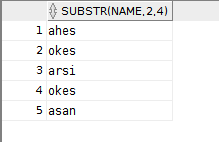


**Query2:** select instr(name,'s') from names;

# Output:



1. **Substr**

**Query1:** select SUBSTR(name,2,4) from names; Output:

**Query2:** select SUBSTR(name,2,4) from names where id=50;

# Output:

1. **lpad**

# rpad

1. **trin:**

**Query1:** select lpad(name,8,'@') from names where id=50;

# Output:

**Query2:** select lpad(name,8,'@') from names ;

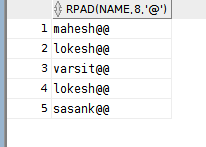
# Output:

**Query1:** select rpad(name,8,'@') from names where id=50;

# Output:

**Query2:** select rpad(name,8,'@') from names ;

# Ouput:



**Query1:** select trim('v' from name) from names where id=50;

# A screen shot of a computer Description automatically generated with low confidenceoutput:

**Query2:** select trim('v' from name) from names;

# A screenshot of a computer Description automatically generated with medium confidenceOutput:

**number functions:**

# mod function:

**query1:** select mod(-638,5) from dual;

# Output:

**round function:**

**query1:** select round(456.8974,3) from dual;

# A screenshot of a computer Description automatically generated with low confidenceOutput:

**trunc function:**

**query1:** select trunc(456.8974,2) from dual;

# A screenshot of a computer Description automatically generated with low confidenceOutput:

**query2:** select trunc(456.8974,-2) from dual;

# Output:

**Date fucntions:**

# add\_months function:

**query1:** select add\_months('25-01-2003',8) from dual

# Output:

**months\_between function:**

**query1:** select months\_between('25-05-2003','25-01-2003') from dual;

# A screenshot of a computer Description automatically generated with low confidenceOutput:

**next\_day function:**

**query1:**select next\_day('03-05-2003','tuesday') from dual;

# A close-up of a computer screen Description automatically generated with low confidenceoutput:

**last\_day function:**

**query1:** select last\_day('03-05-2003') from dual;

# Output:

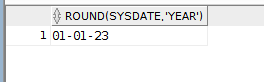
**round function:**

**query1:** select round(sysdate,'month') from dual;

# Output:

**query2:** select round(sysdate,'year') from dual;

# Output:



**Adding hrs,weeks to date:**

**Query1:** select (sysdate+ 45/24) from dual;

# A screenshot of a computer Description automatically generated with low confidenceOutput:

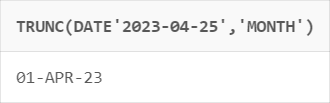
**Query2:** select (sysdate+ 45/7) from dual;

# A screenshot of a computer Description automatically generated with low confidenceOutput:

**trunc function:**

**Query1:** SELECT TRUNC(DATE '2023-04-25', 'MONTH') FROM DUAL;

# Output:



**Query2**: SELECT TRUNC(DATE '2023-04-25', 'year') FROM DUAL;

# A screenshot of a phone Description automatically generated with medium confidenceOutput:

**Query3:** SELECT TRUNC(DATE '2023-04-05', 'month') FROM DUAL;

# A picture containing text, font, screenshot, white Description automatically generatedOutput:

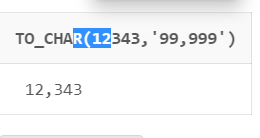
**Conversion functions:**

# To\_char( )

**a. syntax**: to\_char(number,format):

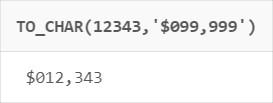
**query1:** select to\_char(12343,'99,999') from dual;

# Output:



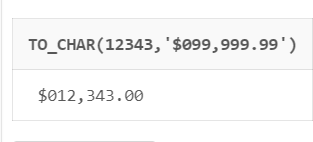
**Query2:** select to\_char(12343,'$099,999') from dual;

# Output:



**Query3:** select to\_char(12343,'$099,999.99') from dual;

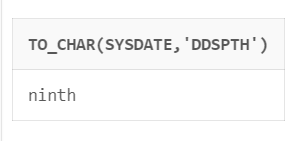
# Output:



**b.Syntax: to\_char(date,format)**

**query1:** select to\_char(sysdate,'ddspth') from dual;

# Output:



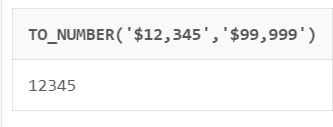
**Query2:** select to\_char(sysdate,'ddth " of the month" month') from dual;

# A picture containing text, font, screenshot, white Description automatically generatedOutput:

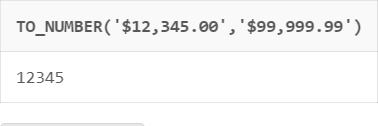
1. **To\_number()**

**Syntax:** to\_number(char,[format model])

# Query1: select to\_number('$12,345','$99,999')from dual; Output:



**Query2: select to\_number('$12,345.00','$99,999.99')from dual; Output:**



# To\_date( )

**Syntax: to\_number(char,[format model])**

# Query1: select to\_date('25/09.2003','dd/mm.yyyy') from dual; Output:

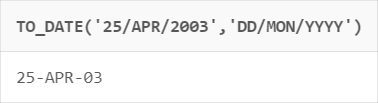
**Query2:** select to\_date('22 of the april 2003','dd "of the" month yyyy') from dual;

# Output:



**Query3:** select to\_date('25/apr/2003','dd/mon/yyyy') from dual;

# Output:



**General Functions( ):**

# A screenshot of a computer Description automatically generated with low confidenceSample table for implementation of general functions:

1. **NVL( )**

**Syntax**: NVL(EXP1,EXP2)

**Query1:** select nvl(salary,99999)from emp;

# Output:



1. **NVL2( )**

**Syntax:** NVL2(EXP1,EXP2,EXP3)

**Query1:** select nvl2(salary,99999,0)from emp;

# Output:

1. **NULLIF( )**

**Syntax:** NULLIF(EXP1,EXP2)

**Query1:** select nullif(salary,18000) from emp;

# Output:



**Query2:** select nullif(salary,99999) from emp;

# Output:



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No.** 4 | |
| **Course Code : 20CSC17**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

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

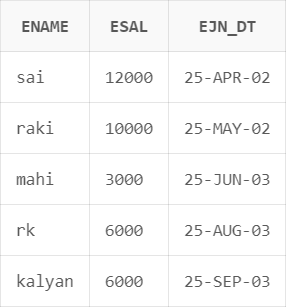
# THEORY:

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

# QUERIES:

**Database table:**



# Group functions:

**Avg():**

**Query1:** select avg(esal) from emp;

# A picture containing text, font, white, design Description automatically generatedOutput:

**Query2:** select avg(distinct esal) from emp;

# A screenshot of a computer Description automatically generated with low confidenceOutput2:

**Sum():**

**Query1:** select sum( distinct esal) from emp;

# A picture containing text, font, white, screenshot Description automatically generatedOutput1:

**Query2:** select sum(esal) from emp;

# A picture containing text, font, white, design Description automatically generatedOutput2:

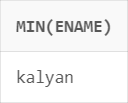
**Min():**

**Query1:** select min(esal) from emp;

# Output1:

**Query2:**select min(ename) from emp;

# Output2:



**Query3:** select min(ejn\_dt) from emp;

# Output3:



**Max( ):**

**Query1:** select max(ejn\_dt) from emp;

# A picture containing text, font, white, design Description automatically generatedOutput1:

**Query2:** select max( distinct esal) from emp;

# A picture containing text, font, white, design Description automatically generatedOutput2:

**Count( ):**

**Query1:** select count(distinct esal) from emp;

# A picture containing text, font, white, screenshot Description automatically generatedOutput1:

**Query2:** select count(esal) from emp;

# A picture containing text, font, white, design Description automatically generatedOutput2:

**GroupBy:**

**Query:** select ename,sum(esal) from emp group by ename;

# Output:

**CONSTRAINTS:**

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.

# QUERIES:

1. **notnull( ):**

**Query1:** *create table student(*

*sid int not null, sname varchar(6), contactno int*

*);*

*insert into student values(null,'rakesh','8929929222');*

***output:*** ORA-01400: cannot insert NULL into

("SQL\_NVVKNNFLQUFUIMXJXMHMDAEYJ"."STUDENT"."SID") ORA-06512: at "SYS.DBMS\_SQL", line 1721

**query2:** *create table student(*

*sid int,*

*sname varchar(6), contactno int*

*constraint studnt\_sid\_cno notnull(sid,contactno);*

*);*

**Output:** ORA-02253: constraint specification not allowed here Invalid statement

# Unique:

**Query1*:*** *create table student(*

*sid int,*

*sname varchar(6), contactno int,*

*constraint studnt\_sid\_cno unique(sid)*

*);*

*insert into student values('9','rakesh','8929929222'); insert into student values('9','raju','8929929222');*

**Output:** ORA-00001: unique constraint

(SQL\_NVVKNNFLQUFUIMXJXMHMDAEYJ.STUDNT\_SID\_CNO) violated ORA-06512: at "SYS.DBMS\_SQL", line 1721

**Query2:** *create table student( sid int unique,*

*sname varchar(6), contactno int*

*);*

*insert into student values('9','rakesh','8929929222'); insert into student values('9','raju','8929929222');*

***Output:*** ORA-00001: unique constraint

(SQL\_NVVKNNFLQUFUIMXJXMHMDAEYJ.SYS\_C00124800090) violated ORA-06512: at "SYS.DBMS\_SQL", line 1721

# primary key:

**Query1:** *create table student(*

*sid int primary key, sname varchar(6), contactno int*

*);*

*insert into student values('9','rakesh','8929929222'); insert into student values('9','raju','8929929222');*

***Output:*** ORA-00001: unique constraint

(SQL\_NVVKNNFLQUFUIMXJXMHMDAEYJ.SYS\_C00124800759) violated ORA-06512: at "SYS.DBMS\_SQL", line 1721

1. ***foreign key():***

***Query:*** *create table student( sid int,*

*did int primary key, sname varchar(6), contactno int*

*);*

*insert into student values('9','1','rakesh','8929929222'); insert into student values('12','2','raju','8929456455'); create table emp(*

*eid int primary key, did int,*

*ename varchar(6), ecntno int,*

*constraint ct\_fkey foreign key(did) references student(did) on delete cascade*

***Check( ):***

*);*

*insert into emp values('2','3','srinu','9568292143');*

***Output:*** ORA-02291: integrity constraint

(SQL\_NVVKNNFLQUFUIMXJXMHMDAEYJ.CT\_FKEY) violated - parent key not found ORA- 06512: at "SYS.DBMS\_SQL", line 1721

***Query1:*** *create table student(*

*sid int check between 1 and 20, sname varchar(6),*

*sno int,*

*);*

*insert into student values('72','rakesh','9832563248');*

***output1:*** ORA-02290: check constraint (SQL\_NVVKNNFLQUFUIMXJXMHMDAEYJ.CHK\_ID) violated ORA-06512: at "SYS.DBMS\_SQL", line 1721

***Query2:*** *create table student( sid int,*

*sname varchar(6), sno int,*

*constraint chk\_id check (sid in (1,20))*

*);*

*insert into student values('72','rakesh','9832563248');*

***output2:*** ORA-02290: check constraint (SQL\_NVVKNNFLQUFUIMXJXMHMDAEYJ.CHK\_ID) violated ORA-06512: at "SYS.DBMS\_SQL", line 1721

# JOINS:

**THEORY:**

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

Here are the different types of the JOINs in SQL:

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

A view is nothing more than a SQL statement that is stored in the database with an associated name.

A view is actually a composition of a table in the form of a predefined SQL query.

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

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

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

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

**Table for Joins:** *create table student( sid int,*

*sname varchar(6), sdept varchar(6)*

*);*

*create table emp( eid int,*

*ename varchar(6), dept varchar(6)*

*);*

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

*insert into emp values('1','nagu','cse');*

# equal join:

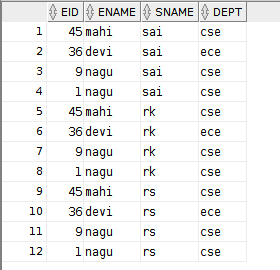
**query1:***select eid,ename,sname,dept from emp,student where emp.eid = student.sid;*

***Output1:***



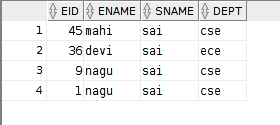
1. ***Cartesian product Join:***

***Query1:*** *select eid,ename,sname,dept from emp,student*

***Output1:***

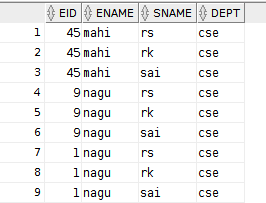
1. ***non equal join:***

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

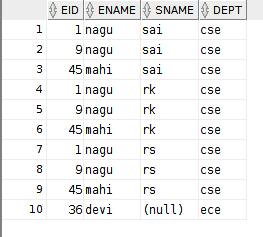
***output1:***

1. ***outerjoin:***

***query1:*** *select eid,ename,sname,dept from emp,student where emp.dept(+)=student.sdept;*

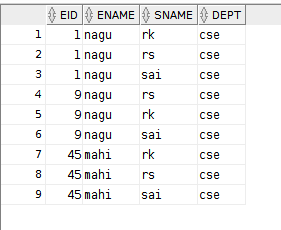
***Output1:***

***query2:*** *select eid,ename,sname,dept from emp,student where emp.dept=student.sdept(+);*

***Output2:***

1. ***innerjoin:***

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

***Output1:***

***cross jojn:***

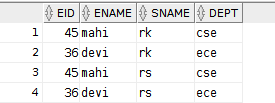
***query1:*** *-- insert into student values('51','rk','cse');*

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

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

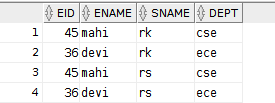
*-- insert into emp values('36','devi','ece');*

*select eid,ename,sname,dept from emp cross join student;*

***Output1:***

***Natural Join***

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

***Output1:***

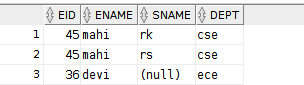
***On class:***

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

***Output1:***

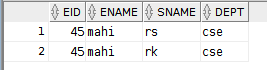
# leftouter join:

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

***Output1:***

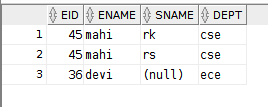
***rightOuterjoin:***

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

***Output1:***

***FullOuterjoin:***

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

***Output1:***

# Set Operations:

**Sample tables creation for set operations:**

***create table cse( sname varchar(6), srollno number(12)***

***);***

***insert into cse values('rakesh','160121733051'); insert into cse values('mahesh','160121733045'); insert into cse values('sai','160121733050');***

***create table it( sname varchar(6),***

***srollno number(12)***

***);***

***insert into it values('ramu',160121733143); insert into it values('sai',160121733050);***

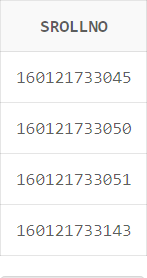
# Union:

**Query1:** *select srollno from cse*

*union*

*select srollno from it;*

# Output:



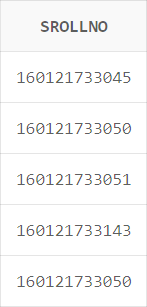
**Union All:**

# Code:

**Output:**

*select srollno from cse UNION ALL*

*select srollno from it;*



# Intersect:

**Query:** *select srollno from cse intersect*

*select srollno from it;*

# Output:



**Minus:**

**Query:** *select srollno from cse minus*

*select srollno from it;*

# Output:



**Exercise-1 Query1:**

*create table departments(*

*department\_id number(4) NOT NULL,*

*department\_name VARCHAR2(30) NOT NULL, manager\_id number(6),*

*location\_id number(4)); desc departments;*

*insert into departments values(10,'Adminstration',200,1700); insert into departments values(20,'Marketing',201,1800);*

*insert into departments values(50,'Shipping',124,1500); insert into departments values(60,'IT',103,1400);*

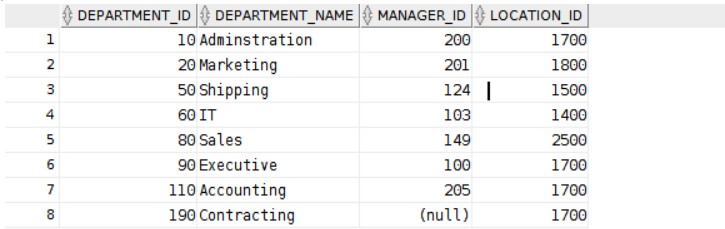
*insert into departments values(80,'Sales',149,2500);*

*insert into departments values(90,'Executive',100,1700);*

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

*select \* from departments;*

# Output:



**Query2:**

*create table Employees(*

*employee\_id number(6) not null, first\_name varchar(20),*

*last\_name varchar(25) not null, email varchar(25) not null, phone\_number varchar(20),*

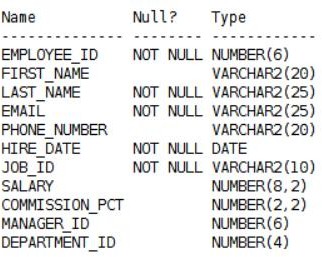
*Hire\_date date not null, job\_id varchar(10) not null, salary number(8,2), commission\_pct number(2,2), manager\_id number(6),*

*department\_id number(4)*

*);*

*describe Employees;*

# Output:



**Query3:**

*create table Employees(*

*employee\_id number(6) not nul first\_name varchar(20),*

*last\_name varchar(25) not null,*

*email varchar(25) ,*

*phone\_number varchar(20), Hire\_d ate date not null, job\_idvarchar(10) not null, salary number(8,2), commission\_pct number(2,2), manager\_id number(6),*

*department\_id number(4)*

*);*

*describe Employees;*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(100,'king','AD\_PRES','17-jun-1987');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(101,'kochhar','AD\_VP','21-sep-1989');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(102,'De Haan','AD\_VP','13-jan-1993');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(103,'Hunold','IT\_PROG','03-jan-1990');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(104,'Emst','IT\_PROG','21-MAY-1991');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(107,'Lorentz','IT\_PROG','07-FEB-1999');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(124,'Mourgos','ST\_MAN','16-NOV-1999');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(141,'Rajas','ST\_CLERK','17-OCT-1995');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(142,'Davies','ST\_CLERK','29-jan-1997');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(143,'Matos','ST\_CLERK','15-mar-1998');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(144,'Vargas','ST\_CLERK','09-jul-1998');*

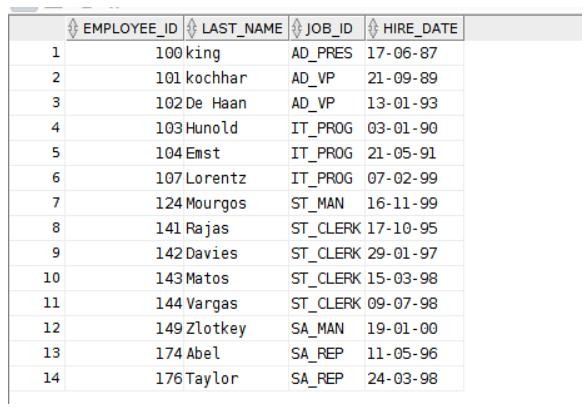
*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(149,'Zlotkey','SA\_MAN','19-jan-2000');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(174,'Abel','SA\_REP','11-may-1996');*

*insert into Employees(EMPLOYEE\_ID,last\_name,job\_id,hire\_date) values(176,'Taylor','SA\_REP','24-mar-1998');*

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

# Output:



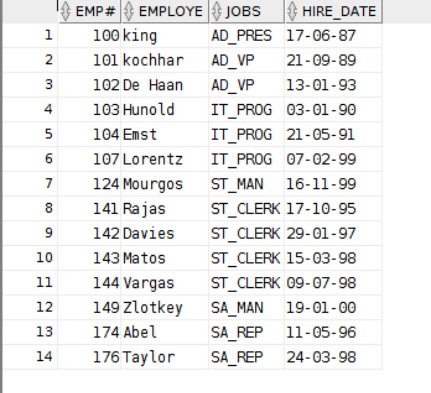
1. **Create a query to display unique job codes from the EMPLOYEES table. Query1:***select distinct job\_id from employees;*

# Output:



1. **Name the column headings Emp #, Employee, Job, and Hire Date, respectively Query2:** *select employee\_id as emp#,last\_name as employe,job\_id as jobs,hire\_date as hire\_date from employees;*

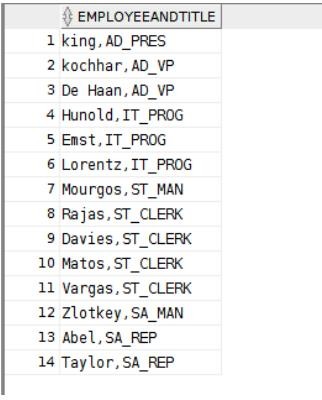
# Output:



**3.**

**Query3:** *select last\_name||','||job\_id as Employeeandtitle from employees;*

# Output:



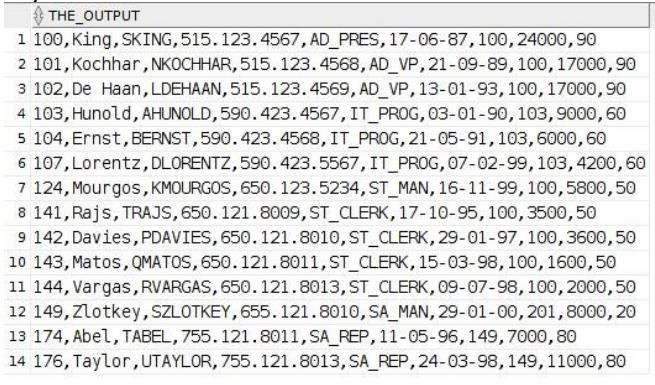
1. **Query:**

*select*

*employee\_id||','||last\_name||','||email||','||phone\_number||','||job\_id||','||hire\_date||','||mana g*

*er\_ id||','||salary||','||department\_id as THE\_OUTPUT from employees;*

# Output:



1. **Query:**

*select last\_name,salary from employees where salary>12000;*

# Output:



1. **Query:**

*select last\_name,department\_id from employees where employee\_id=176;*

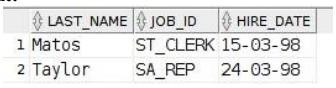
# A close up of a sign Description automatically generatedOutput:

1. **Query:**

*select last\_name,job\_id,hire\_date from*

*employees where(hire\_date between '20-Feb98' and '01-May-98') order by hire\_date ASC;*

# Output:



1. **Query :**

*select last\_name,department\_id from employees where department\_id in (20,50) order by*

*last\_name ASC;*

# Output:



1. **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:**



# 10Query:

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

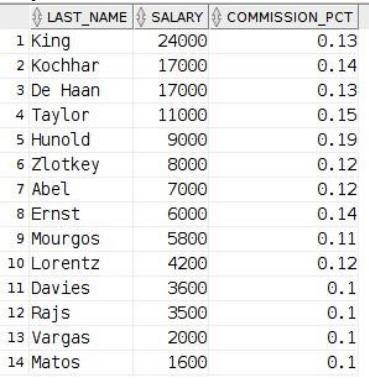
# Output:

A close up of a sign  Description automatically generated

1. **Query:**

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

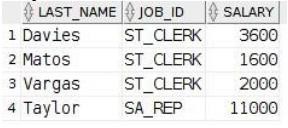
***Output:***



1. **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



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No. 5** | |
| **Course Code : 20CSC17**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**AIM:** Write the queries for evaluating subqueries.

Theory:

A selected clause under or embedded in select clause or nested query.

Syntax:

select cloumns from table\_name where( col\_name (operator) (select statement));

# SUBQUERIES

1. select ord\_no,purch\_ant,ord\_date,customer\_id,salesman\_id from orders

where salesman\_id=( select salesman\_id from salesman

where sname='Paul Adam' );

# OUTPUT:

A screenshot of a computer  Description automatically generated;

1. select ord\_no,purch\_ant,ord\_date,customer\_id,salesman\_id from orders

where salesman\_id in (select salesman\_id from salesman

where city='london');

# OUTPUT:

A screenshot of a computer  Description automatically generatedl

1. select ord\_no,purch\_ant,ord\_date,customer\_id,salesman\_id from orders

where customer\_id=3007

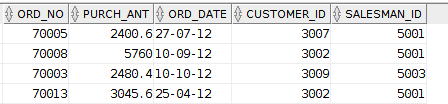
A screenshot of a computer  Description automatically generatedl

1. select ord\_no,purch\_ant,ord\_date,customer\_id,salesman\_id from orders

where purch\_ant >(select avg(purch\_ant) from orders

where ord\_date='10-Oct-12');

# OUTPUT:

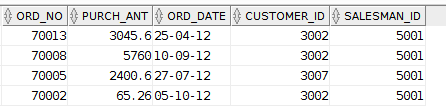
l

1. select ord\_no,purch\_ant,ord\_date,customer\_id,salesman\_id from orders

where salesman\_id in (select salesman\_id from salesman

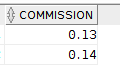
where city='NewYork');

# OUTPUT:



1. select commission from salesman

where city='Paris';

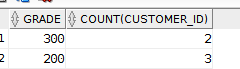


8. select grade,count(customer\_id) from customer

where grade > (select distinct avg(grade) from customer

where city='Newyork') group by (grade);

# Output:

l

7. select \* from customer

where customer\_id in (select salesman\_id-2001 from salesman

where sname='Mc Lyon');

# OUTPUT:

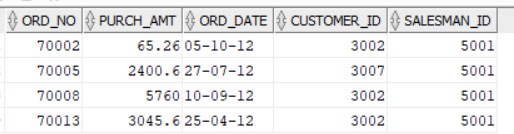
A close up of a logo  Description automatically generated

1. select \* from orders

where salesman\_id in (select salesman\_id from salesman

where commission=(select max(commission) from salesman ));

# OUTPUT:

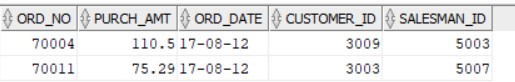


1. select \*

from orders

where ord\_date='17-aug-2012';

# OUTPUT:

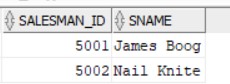


1. select salesman\_id,sname from salesman

where salesman\_id in (select salesman\_id from customer

having count(salesman\_id)>1 group by salesman\_id);

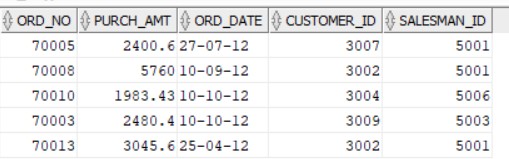
# OUTPUT:



1. select \* from orders

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

# OUTPUT:

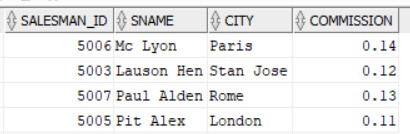


1. select \* from salesman

where salesman\_id in (select salesman\_id from customer

having count(salesman\_id)=1 group by salesman\_id );

# OUTPUT:

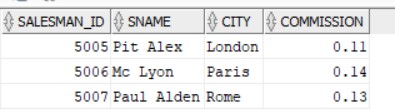


1. select \* from salesman

where salesman\_id in (select salesman\_id from orders

having count(salesman\_id)=1 group by salesman\_id);

# OUTPUT:

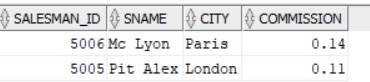


16. select \* from salesman

where salesman\_id in (select salesman\_id from customer

where salesman.city=customer.city);

# OUTPUT:

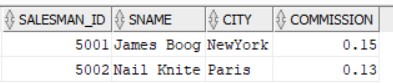


15. select \* from salesman

where salesman\_id in (select salesman\_id from customer

having count(city)>1 group by salesman\_id);

# OUTPUT:

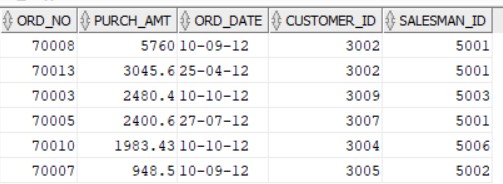


1. select \* from orders

where purch\_amt > ANY (select purch\_amt from orders

where ord\_date='10-sep-2012');

# OUTPUT:

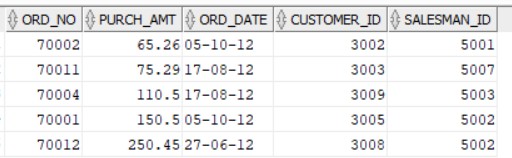


1. select \* from orders

where purch\_amt < any (select purch\_amt from orders a, customer b

where a. customer\_id=b.customer\_id and b.city='London');

# OUTPUT:

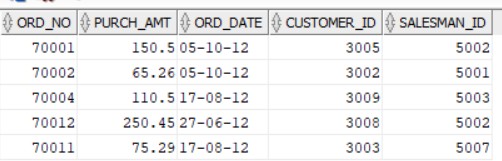


1. select \* from orders

where purch\_amt<(select max(purch\_amt) from orders a,customer b

where a.customer\_id=b.customer\_id and b.city='London');

# OUTPUT:

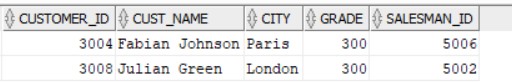


1. select \* from customer

where grade > all(select grade from customer

where city='New York');

# OUTPUT:

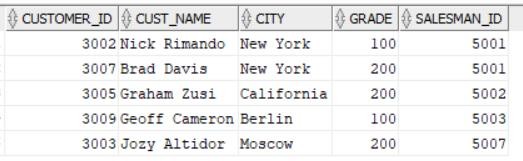


1. select \* from customer

where grade <> any(select grade from customer

where city = 'London');

# OUTPUT:



1. select \* from customer

where grade <> any(select grade from customer

where city = 'Paris');

# OUTPUT:

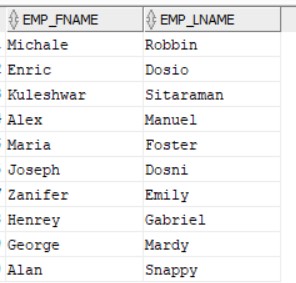


1. select emp\_fname,emp\_lname from emp\_details

where emp\_dept in (select dept\_code from emp\_department

where dpt\_allotment>50000);

# OUTPUT:



1. select dept\_code,dpt\_name,dpt\_allotment from emp\_department

where dpt\_allotment>all(select avg(dpt\_allotment) from emp\_department);

# OUTPUT:

A close up of a box  Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No. 6** | |
| **Course Code : 20CSC17**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

AIM: To write queries for evaluating views

THEORY:

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.

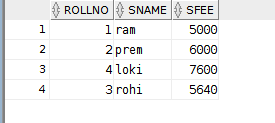
A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

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.

# Sample Table :



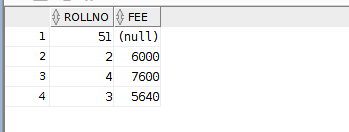
***create view using group functions***

***Query:***

***Output:***

*create view grpviw as*

*select rollno,min(sfee) as fee from student group by rollno; select \* from grpviw;*



# creating a view:

***Query:***

***Output:***

*create view cse\_studt*

*as select rollno,sname,sfee from student where sfee=5000;*

*select \* from cse\_studt;*

A close-up of a white rectangular object  Description automatically generated

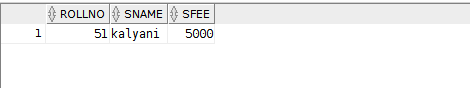
***Insert into view :***

***Query:***

*insert into cse\_studt values(51,'kalyani',5000);*

*select \* from cse\_studt;*

***Output:***



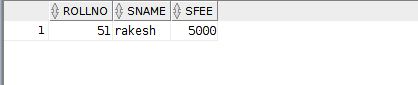
***Update:***

***Query:***

update cse\_studt set sname='rakesh' where rollno=51;

select \* from cse\_studt;

# Output:



***Delete:***

*delete cse\_studt where rollno=51; select \* from cse\_studt;*

***Output:***



|  |  |  |  |
| --- | --- | --- | --- |
| **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY** | | | |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No. 7** | |
| **Course Code : 20CSC17**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

Aim: To implement variable declarations and execute control statements. 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

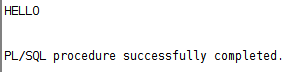
Exception (optional)

-Action performed

End;

***Hello program:***

Set SERVEROUTPUT ON

begin DBMS\_output.put\_line('HELLO'); end;

# Output:

**insert into student table by pl/sql:**

# code:

*declare*

*vid student.rollno%type:='36' ; vsname student.sname%type:='ramu'; vsfee student.sfee%type:=4600;*

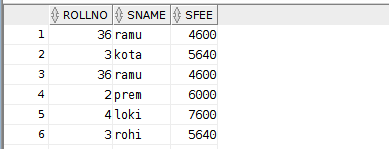
*begin*

*insert into student(rollno,sname,sfee) values(vid,vsname,vsfee); end;*

***Output:***



# verification in student table:



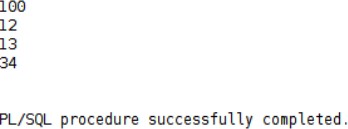
**4.Different types of DECLARATIONS declare**

*x number(10):=100;*

*y number(10) default 12;*

*z number(10) not null:=13; s constant number(10):=34;* **begin** *DBMS\_output.put\_line(x); DBMS\_output.put\_line(y); DBMS\_output.put\_line(z); DBMS\_output.put\_line(s);* **end;**

# Output:



**Condition Structures:**

# if else:

*declare*

*x number(5):=51; begin*

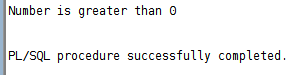
*if(x>0) then*

*DBMS\_output.put\_line('Number is greater than 0'); else*

*DBMS\_output.put\_line('Number is less than 0'); end if;*

*end;*

***Output:***



***if elseif:***

*declare*

*x number(10):=-27; begin*

*if(x>0) then*

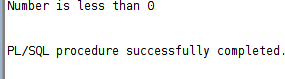
*DBMS\_output.put\_line('Number is greater than 0'); elsif(x<0) then*

*DBMS\_output.put\_line('Number is less than 0'); else*

*DBMS\_output.put\_line('Number is equal than 0'); end if;*

*end;*

***Output:***



***Case:***

*declare*

*v\_grade varchar2(10):='A'; begin*

*case (v\_grade)*

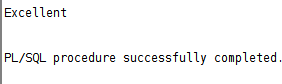
*when 'S' then DBMS\_output.put\_line('Outstanding'); when 'A' then DBMS\_output.put\_line('Excellent'); when 'B' then DBMS\_output.put\_line('Good');*

*when 'C' then DBMS\_output.put\_line('Better'); else*

*DBMS\_output.put\_line('Need to work hard'); end case;*

*end;*

***Output:***



# BASIC LOOP:

**declare**

*x number(10):=5;*

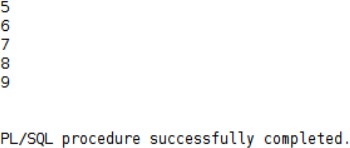
***begin***

*loop*

*DBMS\_output.put\_line(x); exit when(x=9);*

*x:=x+1;*

*end loop;* ***end;* OUTPUT:**



# WHILE:

***declare***

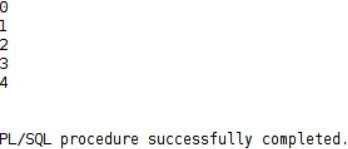
*x number(10):=0;*

***begin***

*while(x<>5) loop*

*DBMS\_output.put\_line(x); x:=x+1;*

*end loop;*

***end; Output:***

# FOR LOOP:

**Query:**

***declare***

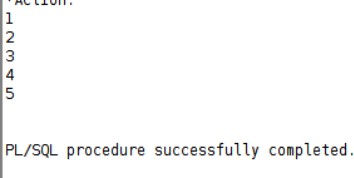
*x number(10):=0; begin*

*for x in 1..5 loop*

*DBMS\_output.put\_line(x); end loop;*

***end;***

# Output:



|  |  |  |  |
| --- | --- | --- | --- |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No. 8** | |
| **Course Code : 20CSC17**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

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

# THEORY:

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

* Implicit cursors
* Explicit cursors

# Implicit cursors: -

**Explicit cursors: -**

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

Declaring the cursor:

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

SynTax: Cursor <cursor name> is SQL statement.

Example: cursor c1 is

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

* + - Opening the cursor includes:

Allocating memory

Parse the SQL statement for execution.

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

E.g.: fetch ename into a;

* + - After the fetch instruction the active set is checked. If it is empty it

enables the close else it moves the cursor to the next row in the active set and continues until the last row is reached. Once the last row is

reaches it enables the close.

* + - 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>;

IMPLICT CURSORS:

FOUND AND ROWCOUNT:

Query:

declare

v\_id salesman.salesman\_id %type:=101; v\_name salesman.sname %type:='sanjana'; v\_city salesman.city %type:='hyderabad'; v\_com salesman.commission %type:=0.8; begin

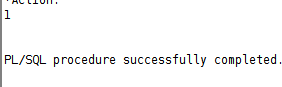
insert into salesman(salesman\_id,sname,city,commission) values(v\_id,v\_name,v\_city,v\_com);

if(sql%found) then

dbms\_output.put\_line(sql%rowcount); else

dbms\_output.put\_line('No rows are affected'); end if;

end;



ISOPEN:

Query:

declare

v\_name salesman.sname %type:='Sanjana'; begin

update salesman

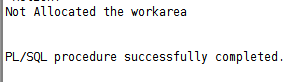
set sname=v\_name

where salesman\_id=101; if(sql%isopen) then

dbms\_output.put\_line('Allocated the workarea'); else

dbms\_output.put\_line('Not Allocated the workarea'); end if;

end;



NOT FOUND:

Query:

declare

v\_name salesman.sname %type:='Sanjana'; begin

update salesman set sname=v\_name

where salesman\_id=101; if(sql%notfound) then

dbms\_output.put\_line('Allocated the workarea and rows are affected'||sql%rowcount); else

dbms\_output.put\_line('Not Allocated the workarea'); end if;

end; Output:



EXPLICT CURSORS:

declare

v\_name place.sname % type; v\_city place.city %type;

v\_id place.sale\_id % type;

CURSOR v\_place is

select sale\_id,sname,city from place where sale\_id=5005;

begin

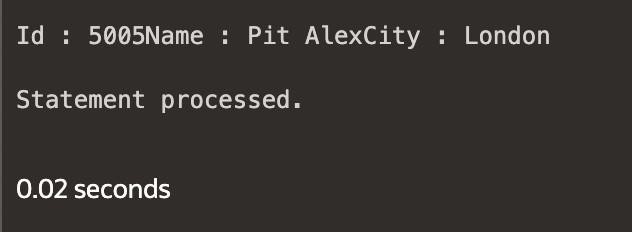
OPEN v\_place;

FETCH v\_place into v\_id,v\_name,v\_city;

dbms\_output.put\_line('Id : '||v\_id||'Name : '||v\_name||'City : '||v\_city); CLOSE v\_place;

end;

Output:



Syntax:

declare

v\_name place.sname % type; v\_city place.city %type;

v\_id place.sale\_id% type;

CURSOR v\_place is

select sale\_id,sname,city from place; begin

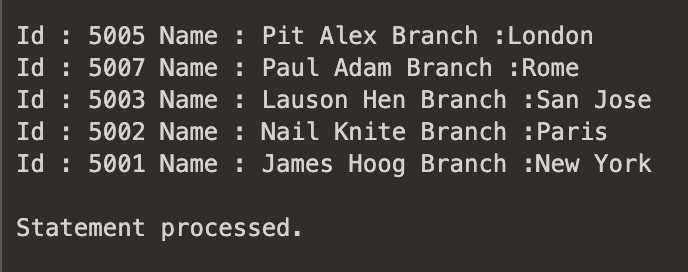
OPEN v\_place; loop

FETCH v\_place into v\_id,v\_name,v\_city; exit when v\_place%notfound;

dbms\_output.put\_line('Id : '||v\_id||' Name : '||v\_name||' Branch :'||v\_city); end loop;

CLOSE v\_place; end;

Output:



Composite variables:

Theory: We have 2 types: %rowtype and record.

%rowtype: refer to one row (depending on the database, can contain n columns). It's inbuiltand by default considers all columns.

record: Used when we want to have our own no.of columns.

Queries:

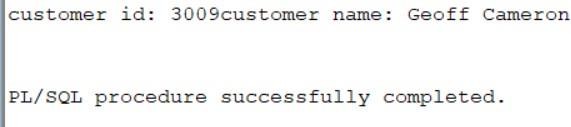
1. Using %rowtype:

Syntax:

declare customer\_rec customer%rowtype;

begin select \* into customer\_rec from customer where customer\_id=3009; DBMS\_output.put\_line('customer id: '||customer\_rec.customer\_id||'customer name: '||customer\_rec.cust\_name);

end; Output:



1. Using record:

Syntax:

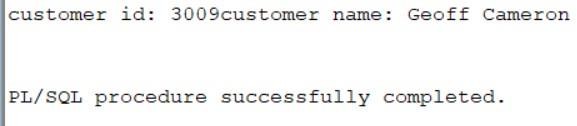
set serveroutput on declare

type cust\_rec is record (customer\_id customer.customer\_id%type,cust\_name customer.cust\_name%type);

customer\_rec cust\_rec; begin

select customer\_id,cust\_name into customer\_rec from customer where customer\_id=3009; DBMS\_output.put\_line('customer id: '||customer\_rec.customer\_id||'customer name: '||customer\_rec.cust\_name);

end; Output:



|  |  |  |  |
| --- | --- | --- | --- |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No. 9** | |
| **Course Code : 20CSC17**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch:** CSE | **Semester:** IV |

**AIM:** write the queries for procedures and 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>]*

# PROCEDURES:

**ADDITION OF 2 NUMBERS:**

*create or replace procedure addition*

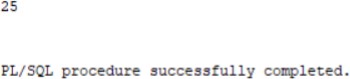
*( x in number,y in number,z out number) IS*

*begin z:=x+y; end;* ***declare***

*a number(5):=&a; b number(5):=&b; c number(5);* ***begin***

*addition(a,b,c); dbms\_output.put\_line(c);* ***end;***

***output:***



# MINIMUM OF THE NUMBERS:

*create or replace procedure min*

*( x in number,y in number,z out number) IS*

***begin***

*if x>y then z:=y;*

*else z:=x; end if;* ***end;***

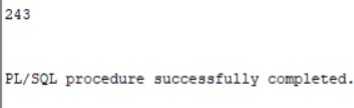
# declare

a number(5):=&a; b number(5):=&b; c number(5); **begin**

min(a,b,c);

dbms\_output.put\_line(c);

# end; OUTPUT:



**SQUARE OF A NUMBER:**

*create or replace procedure square(x in out number) is*

***begin*** *x:=x\*x;* ***end; declare***

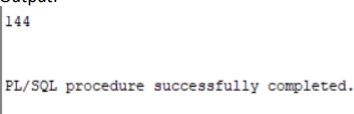
*a number(5):=&a;*

***begin***

*square(a);*

*dbms\_output.put\_line(a);*

***end; Output:***



# FUNCTIONS

***To print the prime numbers in the given range:*** *create or replace function prime(num number) return boolean*

*is*

***begin***

*for i in 2..num-1 loop*

*if(mod(num,i)=0) then return false;*

*end if;*

*end loop; return true;* ***end; declare***

*a number(5):=&a; b number(5):=&b;*

***begin***

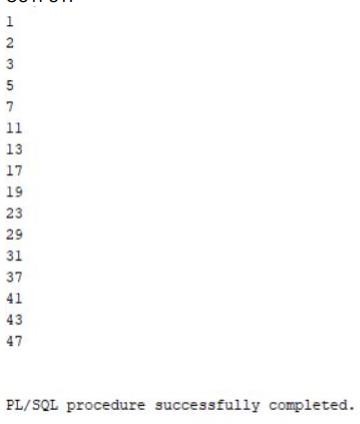
*for j in a..b loop if prime(j) then*

*dbms\_output.put\_line(j); end if;*

*end loop;*

***end;***

***OUTPUT:***



***To check the given number is prime:***

*create or replace function prime(num number) return boolean*

*is*

***begin***

*for i in 2..num-1 loop*

*if(mod(num,i)=0) then return false;*

*end if;*

*end loop;*

*return true;* ***end; declare***

*num1 number(10):=31;*

***begin***

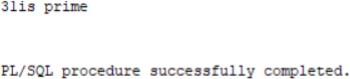
*if prime(num1) then*

*dbms\_output.put\_line(num1||'is prime'); else*

*dbms\_output.put\_line(num1||'not is prime'); end if;*

***end;***

***OUTPUT:***



***To check the given number is palindrome:*** *create or replace function palin(num number) return number*

*as*

*temp integer:=num; rev number:=0;*

*rem number:=0;*

***begin***

*while (temp>0) loop rem:=mod(temp,10); rev:=rev\*10+rem;*

*temp:=floor(temp/10); end loop;*

*return rev;*

***end; declare***

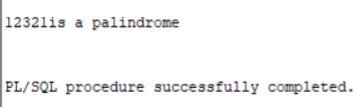
*num1 number:=12321; rev integer;*

***begin***

*rev:=palin(num1); if rev=num1 then*

*dbms\_output.put\_line(rev||'is a palindrome'); else*

*dbms\_output.put\_line(num1||'is not a palindrome'); end if;*

***end; Output:***

|  |  |  |  |
| --- | --- | --- | --- |
| **DBMS LAB MANUAL** | | | |
| IMG_256 | **Course Name.: DBMS Lab** | **Experiment No. 10** | |
| **Course Code : 20CSC17**  **Faculty : Smt.Ch.Vijaya Lakshmi** | **Branch: CSE** | **Semester: IV** |

# Packages:

Aim: To execute and evaluating packages.

# Theory:

A PL/SQL package is a named collection of functions, procedures, variables, cursors, user-defined types, and records that are referenced using a common qualifier, the package name.

# Queries:

create or replace package operation is

function add(a number , b number) return number; function sub(a number , b number) return number; function mul(a number , b number) return number; function div(a number , b number) return number; end operation;

create or replace package body operation as function add(a number ,b number) return number is begin return a+b; end;

function sub(a number ,b number) return number is begin return a-b; end;

function mul(a number ,b number) return number is begin return a\*b; end;

function div(a number ,b number) return number is begin return a/b; end;

end operation; declare x number:=10;

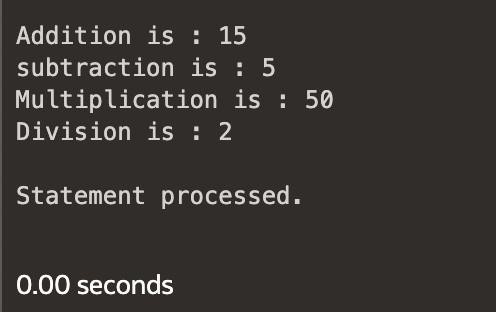
y number:=5; begin

dbms\_output.put\_line('Addition is : '||operation.add(x,y));

dbms\_output.put\_line('subtraction is : '||operation.sub(x,y)); dbms\_output.put\_line('Multiplication is : '||operation.mul(x,y)); dbms\_output.put\_line('Division is : '||operation.div(x,y));

end;

# Output:



**EXCEPTION HANDLING:**

Aim: To write SQL queries to execute exception handling in PL/SQL.

# Theory:

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: One of the approximately 20 errors that occurs in oracle PL/SQL block/code.
* Non predefined exceptions: Needs to declare the exception explicitly in the declarative section.
* User defined exceptions: In this, the user has to explicitly declare the exception, raise an exception as well as handle the exception.

# Queries:

1. **Using built-in exception:**

Syntax:

declare

var1 number (3):=0;

var2 number (4):=100; vsal ann\_sal.sal%type; begin update ann\_sal

set sal= sal+ (var2/var1); exception when ZERO\_DIVIDE then

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

# A black background with white text Description automatically generatedOutput:

1. **User-defined exceptions:**

# Syntax:

declare v\_dep ann\_sal.dep\_name%type; vsal ann\_sal.sal%type; no\_raise exception; begin select dep\_name,sal into v\_dep,vsal from ann\_sal where dep\_name='HR';

if (vsal>25000) then dbms\_output.put\_line('Raise possible'); else raise no\_raise; end if; exception when no\_raise then dbms\_output.put\_line('Exception no\_raise raised...'); end;

# A black background with white text Description automatically generatedOutput: