Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)

NAAC A++

Recognised as CATEGORY-1 DEEMED TO BE UNIVERSITY by the UGC University Grants Commission

## School of Computing

## Department of CSE (Data Science)

## ACADEMIC YEAR 2025 – 2026 (Summer Semester)

# BONAFIDE CERTIFICATE

**NAME:** G. Jagan Mohan Reddy

**VTU NO:** 14203

**REG.NO:** 24UEDC0024

**BRANCH:** CSE (Data science)

**YEAR/SEM:** 2nd | IIIrd

**SLOT:** S15L12

Certified that this is a bonafide record of work done by above student in the **"10211DS207 – Database Management Systems"** during the year 2025-2026.

**SIGNATURE OF FACULTY INCHARGE** **SIGNATURE OF INCHARGE**

*Submitted for the Semester Model Examination held on_____at Vel Tech Rangarajan Dr. Sagunthala R & D Institute of Science and Technology.*

**EXAMINER 1** **EXAMINER 2**

| Sl.No | Date. | Title | marks | Sign |
|---|---|---|---|---|
| 1 | ~~24~~-7-25 | conceptual design | 20 | |
| 2 | ~~31~~-7-25 | generating design of a Traditional database model | 20 | |
| 3. | ~~7~~/08/25 | Developing Queries with DML single row function and operation. | 19 | |
| 4 | ~~18~~-9-25 | Developing queries with multirow functions and operations | ~~20~~ 15 | |
| 5 | ~~21~~-9-25 | writing join Queries, Equivalent ANDP/W Recursive queries | 20 | |
| 6. | ~~28~~-9-25 | PL/SQL procedures, function, loops | 19 | 28/9 |
| 7 | 4-9-25 | PL/SQL program using loops | 18 | 4/9 |
| 8 | 11-9-25 | Normalising databases using functional dependence upto BCNF. | 20 | 11/9 |
| 9 | 18-9-25 | Working up and recovery in database | 20 | 18/9 |
| 10 | 25-9-25 | CRUD operations | 20 | 25/9 |
| 11. | 9-10-25 | CRUD operations IN GRAPH DATABASES | 20 | 9/10 |
| 12. | 16-10-25 | minor project. | 20 | |

Task-1.

Unique ID  composite  multivalued

conceptual design after ER.

Using basic data base design methodology and ER modeler. design Entity Relationship diagram by satisfying the following sub tasks.

1. a. Identifying the entities
   b. Identifying the attributes
   c. Identifying the relationships, cardinality, Type of Relationship
   d. Reforming the Relations with keys and constraints
   e. Develop ER diagram for stated case of Task.

A Bank management system is a comprehensive software solution designed to manage and streamline banking operations. It covers various aspects of banking including customer account management, transaction processing, loans and mortgage management and more.

Aim:- To draw conceptual design through ER using with draw io tool.

procedure:- Identify the entities for bank management system.

1) a. Employ

   2. Branch.

1.b Identify the attributes for Bank management system

Sample output

1.c. Identification of relationships, cardinality, Type of year for bank management system

Sample output

1.d: Reframing the Relationship with keys and constraints for bank management system.

sample output:

1a) Entity

1) Employ

2) Branch.

1b) Attributes:-

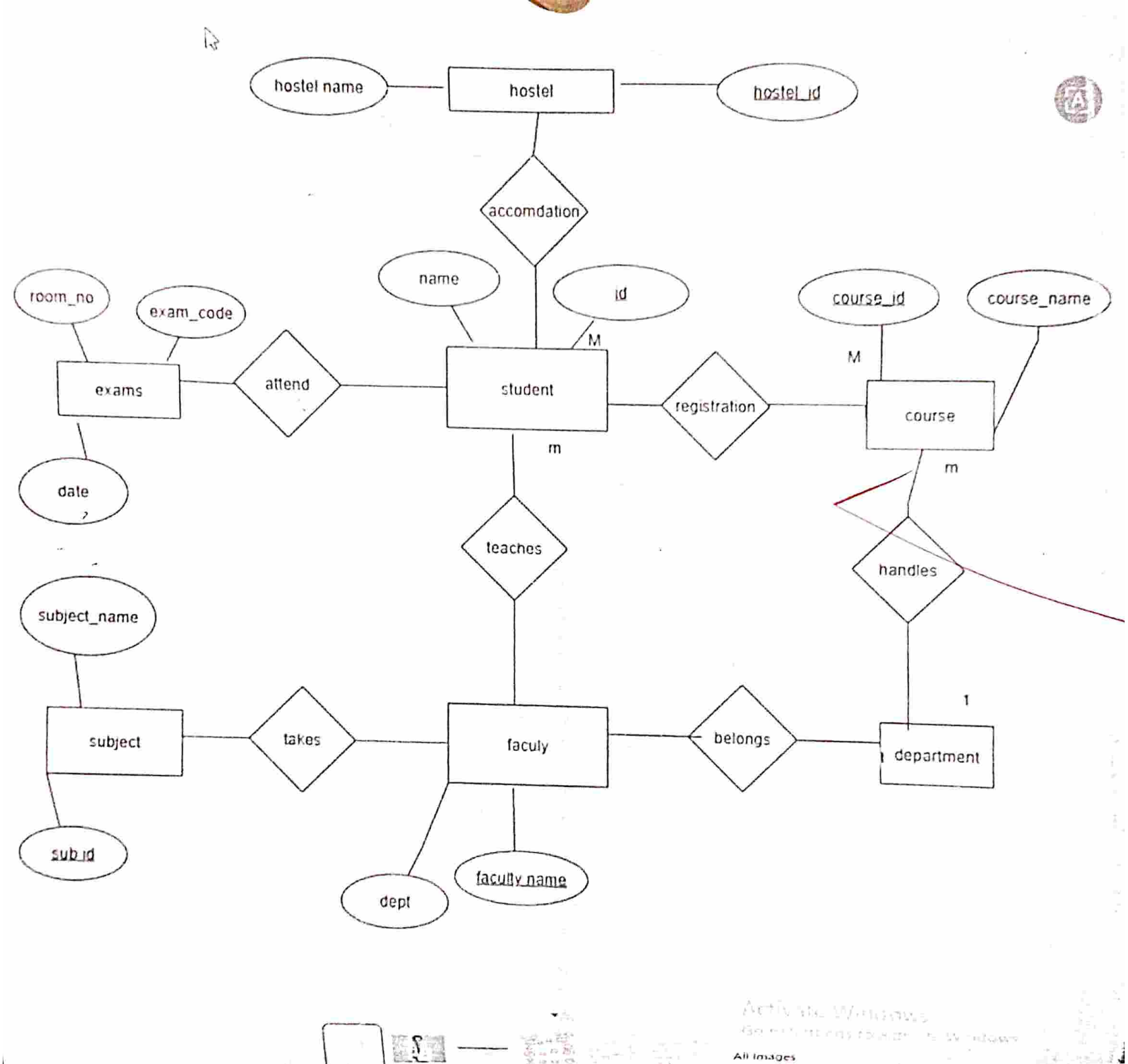
Employ
Age
AAdhar Number
ID
phone number
Address

Branch
Branch name

Branch Addres

1.c) Relationship

1) Deposit
2) with drawl
3) loan.


hostel name
hostel
hostel_id
accomdation
name
id
room_no
exam_code
course_id
course_name
M
M
exams
attend
student
registration
course
m
m
date
teaches
handles
subject_name
subject
takes
faculy
belongs
department
1
sub_id
dept
faculty name

d). keys and constraints.

Unique key:-

→ ID.

AAdhar Number.

→ Derived

Age

DOB.

→ multivalued.

phone number.

→ composite attribute

first Name

middle Name

last Name.

| VEL TECH | |
|---|---|
| EX NO. | 1 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYSIS (5) | 5 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | |
| TOTAL (20) | 18 |
| SIGN WITH DATE | |

Result:- Thus drawing concept design Through ER using drawio was executed Successfully

Task-2 31-7-25

Generating design of other traditional database model

Creating Hierarchical / Network model of the database by enhancing the sound abstract data by performing tasks using form of inheritance:

2.a identify the specificity of each relationship find and form surplus relations.

2.b. check is-a hierarchy / has-a hierarchy and performs generalization and/or specialization relationship

2.c find the domain of the attribute and perform a check constraint to the applicable

2.d Rename the relations

2.e perform SQL Relations using DDL, DCL commands.

Implementation of DDL, DCL, and TCL commands of SQ

AIM:- Implementation of DDL, DCL and TCL commands of SQL with suitable examples.

Objective:-

* To understand the different issues involved in the design and implementation of a database system
* To understand and use data definition language to write query for a database.

Theory:-

Oracle has many tools such as SQL*plus, Oracle forms, oracle Report writer, Oracle graphics etc.

SQL* PLUS: The SQL* plus tool is made up of two distict parts they are

Interactive SQL: Interactive SQL is designed create, access and manipulate data str like tables and indexes.

PL/SQL: PL/SQL can be used to developed program for different applications.

Oracle forms:- This tool allows you to create a data entry screen along with the suitable menu objects. Thus it is the oracle forms tool that handles data gathering and data validation in a commaial application.

Report writer:- Report writer allows programmas to prepare innovative reports using data from the oracle structures like tables, views etc. It is the Report writer tool that handles the Reporting section of commacial application.

Oracle graphics:- Some of the data can be better represented in the form of pictures. The oracles graphics tool allows programmas to prepare graphics using data from oracle structure like tables, view etc..

SQL (structured Query language):

It is a database computer language designed for managing data in relational data base management systems (RDBMS), and originally based upon Relational Algebra its scope includes data query and update, schema creation and modification, and data access control

SQL was one of the first languages for Edgar F. codd's relational model and became the most widely used language for relational databases.

- IBM developed SQL in mid of 1970's
- Oracle incorporated in the year 1979.
- SQL used by IBM/DB2 and DS Database Systems
- SQL adopted as standard language for RDBS by ANSI in 1989.

# DATA TYPES:-

1. CHAR (size): This data type is used to store character strings values of fixed length. The size in brackets determine the number of characters the cell can hold. The maximum no. of character is 255 characters.
2. VARCHAR (size) / VARCHAR (size):- The datatype is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.
3. NUMBER (P, S): The number data type is used to store number (fixed or floating point). No. of virtually any magnitude may be stored up to 38 digits of precision. number of large as $9.99 * 10^{124}$. The precision (P) determines the number of places to the Right of the decimal. if scale is omitted then the default is zero. if precision is omitted, values are stored with their original precision up to the maximum of 38 digits.
4. DATE:- This data type is used to represent data in the 24 Hours format. By appropiate functions date time to store date in the 24-Hour format. By default the time in a date field is 12:00:00 am, if no time portion is specified
5. LONG:- This data type is used to store data in 24 Hour, format By) variable length character strings containing up to 2GB long data can be used to store arrays of binary data in a ASCII format LONG values cannot be indexed and the normal character funtions such as SUBSTR

RAW:- The RAW data type is used to store binary data, such as digitized picture or image data loaded into columns of. These data types are stored in without any future conversion. RAW data can have a maximum length of 255 bytes. LONG RAW data type can contain up to 2GB.

DDL (Data definition languages:-
for shipping management system.

→ DDL statement are used to define the data base structure or schema.

Create:- To make a new database, table, index or stored query, A create statement in SQL creates an object inside of a relational database management (RDBMS)

Syntax:-

→ CREATE TABLE table_name.
(column_name1 data_type ([size]), column_name2 data_type([size]), .... column_namen data_type([size]));

Alter:- to modify an existing object. Alter the structure of the database.

- To add a column in a table:

syntax:-

ALTER TABLE table_name ADD column_name datatype([size]);

- To delete a column ina table.

Syntax:-

ALTER TABLE table_name DROP column column_name;

- To modify a column ina table

Syntax:-

ALTER TABLE table-name mudify column-name data-type ([new-size]);

Rename:-

Syntax:-

ALTER TABLE table-name RENAME column old-column-name to new-column-name;

Describe:-

Syntax:-

DESC table-name;

- desc shopping customers;

TRUNCATE TABLE:

Remove all records from a table, including all spaces allocated for the records are removed.

Syntax:-

TRUNCATE TABLE table-name;

Data MANIPULATION LANGUAGES:-(DML)

Data manipulation languages allows the users to query and manipulate data in existing schema in object. It allows following data to insert, delete, update and recovery data in schema object.

insert:-

values can be inserted into table using insert commands. There are two types of insert commands. They are multiple value insert commands (using '&' symbol) single

single value insert command (without using '&' symbol)

Syntax:-

INSERT INTO table_name VALUES (value1, value2, ..);

(OR)

INSERT INTO table_name (column1, column2, column3, ..)
VALUES (value1, value2, value3, ...);

- Select * from table_name;

UPDATE:-

This allows the user to update the particular column values using the where clause condition.

Syntax:-

UPDATE <table_name> SET <col1=value> WHERE <column2=value>;

Delete:-

This allow you to delete the particular column values using the where clause condition

Syntax:-

DELETE FROM <table_name> WHERE <condition>;

Select:-

The select statement is used to query a database. This statement is used to retrieve the information from the database. The SELECT statement can be used in many ways. They are;

1. Selecting some columns:

To select specified number of columns from the table the following command is used.

Syntax;

SELECT column_name FROM table_name;

2. Select using DISTINCT:-

The Distinct keyword is used to return only different values (i.e.) This command does not select the duplicate values from the table.

Syntax:-

SELECT DISTINCT column name(s) FROM table-name;

3. Select using BETWEEN:

BETWEEN can be used to get those items that fall with ina range.

Syntax:-

SELECT column name FROM table_name WHERE column name BETWEEN value 1 AND value2;

4. Renaming:-

The select statement can be used to rename either a column or the entire table

Syntax:

Renaming a column:

SELECT column name AS new name FROM table_name;

5) sorting:-

The select statement with the order by clause is used to sort the contents.

Table either in ascending or descending order.

Syntax:-

SELECT column name FROM Table_name WHERE condition ORDER BY column name ASC/DESC;

6). To select by matching some patterns:

The select statement along with like clause is used to match strings.

The like condition is used to specify a search

pattern in a column.

Syntax:-

SELECT column name FROM table-name WHERE column name like "%.or-";

% : matches any sub string.

- : matches a single character.

7. Select using AND, OR, NOT:

We can combine one or more conditions in a SELECT statement using the logical operations AND, OR, NOT.

Syntax:-

SELECT column name FROM table-name WHERE condition 1 logical OPERATOR condition 2.

DATA CONTROL LANGUAGES:-

1. Create:-

2. Grant:

3. Revoke:

TRANSACTION CONTROL LANGUAGES:-

1) Commit:-

* commit;

2) Save point:

• save point k1;

3. Roll back:

• roll back to k1;

Model

```
Connected.
SQL> create table student (name varchar(10), vtu_no number(5),address
varchar(15));

Table created.

SQL> create table shoping (sl_no number(2), product_name varchar(25),cost
number(20));

Table created.

SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 NAME                                               VARCHAR2(10)
 VTU_NO                                             NUMBER(5)
 ADDRESS                                            VARCHAR2(15)

SQL> desc shoping;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SL_NO                                              NUMBER(2)
 PRODUCT_NAME                                       VARCHAR2(25)
 COST                                               NUMBER(20)

SQL> alter table student add phonenumber number(10);

Table altered.

SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 NAME                                               VARCHAR2(10)
 VTU_NO                                             NUMBER(5)
 ADDRESS                                            VARCHAR2(15)
 PHONENUMBER                                        NUMBER(10)

SQL> insert into student values('jagan',29203,'ap',9275898999);

1 row created.

SQL> select* from student
  2
SQL> select* from student;

NAME           VTU_NO ADDRESS         PHONENUMBER
---------- ---------- --------------- -----------
jagan           29203 ap               9275898999

SQL> insert into shoping values(1,'t_shirt',500);

1 row created.

SQL> select* from shoping;
```

narendra 28995 ap 5679247678

```
SQL> select distinct name , phonenumber from student;

NAME       PHONENUMBER
---------- -----------
chandu      9275898999
narendra    5679247678

SQL>  insert into student values('yaswanth',29234,'ap',7075301267);

1 row created.

SQL>  insert into student values('shiva',30140,'tn',9391244512);

1 row created.

SQL> select* from student;

NAME           VTU_NO ADDRESS          PHONENUMBER
---------- ---------- -------------- -----------
chandu          29203 ap              9275898999
narendra        28995 ap              5679247678
yaswanth        29234 ap              7075301267
shiva           30140 tn              9391244512

SQL> select* from student where vtu_no between 2900 and 3000;

no rows selected

SQL> select name, phonenumber from student;

NAME       PHONENUMBER
---------- -----------
chandu      9275898999
narendra    5679247678
yaswanth    7075301267
shiva       9391244512

SQL> select address as addresses from student;

ADDRESSES
--------------
ap
ap
ap
tn
```

```
    SL_NO PRODUCT_NAME                                   COST
--------- ------------------------------ ----------
        1 t_shirt                                       500

SQL> insert into student values ('narendra',28995,'ap',56792476

1 row created.

SQL> insert into student ('gajini',29146,'ap',1234567890);
insert into student ('gajini',29146,'ap',1234567890)
                     *
ERROR at line 1:
ORA-00928: missing SELECT keyword


SQL> insert into student values ('gajini',29146,'ap',3456247678);

1 row created.

SQL>  select* from student;

NAME            VTU_NO ADDRESS          PHONENUMBER
---------- ---------- -------------- -----------
jagan            29203 ap                9275898999
narendra         28995 ap                5679247678
gajini           29146 ap                3456247678

SQL> update student set jagan
  2
SQL>  update student set chandu where name=jagan;
 update student set chandu where name=jagan
                           *
SQL>  update student set name='chandu' where name='jagan';

1 row updated.

SQL> select* from student;

NAME            VTU_NO ADDRESS          PHONENUMBER
---------- ---------- -------------- -----------
chandu           29203 ap                9275898999
narendra         28995 ap
```

| VEL TECH | |
|---|---|
| EX NO. | 2 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYSIS (5) | 5 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | |
| TOTAL (20) | 15 |
| SIGN WITH DATE | |

Result:- Thus the Implementation of DDL, DML, DCL, and TCL commands with suitable examples has successfully executed

Date: 28-25.

# Task-3.

Developing queries with DML single rows function and operators.

Perform the query processing on database for different results of queries using, DML, DRL, Single rows operation.

Using aggregate, data, string, student functions set columns and operations

Using aggregation data, string, st

create table employee schema, and insert, around details retail - store - employee data in this relation perform multiple rows function.

NOTE:- These queries asume a sample database with an retails - store-employee"; Table sumilating column table

"retail - store- employee -id", "store - id",
"retail - store - employee -name", "salary",
and "department", "store - plant - numbers",
"employee - plant number".

Aggregative operations:- In addition to simply retrieving data, we often want to perform some computation or summarization. SQL allows the use of arithmetic expressions. we now consider a powerful class of constructs for computing aggregate values such as MIN and SUM.

1. count :- count following by a column name returns the count of tuple in that column. If DISTINCT key word. is used then it will return only the count of unique tuple in the column. otherwise, it will return count of all the tuples.

Syntax:- COUNT ( column name)
Ex:- Select count(*) from retail - store - employees;

SUM:- Sum followed by a column name returns the sum of all the values in the column.

Syntax:- SUM (column name)

Ex:- select sum (salary) from retail store_employees;

Avg : Avg followed by a column name returns the avg value of that column values.

Syntax:- AVG (n1, n2, ...)

Example : SELECT AVG (salary) from retail_store_employee;

4. MAX:- MAX followed by a column name returns the maximum value of that column

Syntax:- MAX (column name)

Example:- select MAX (salary) from retail_store_employees;

5. MIN:- MIN followed by column name returns the minimum value of that column.

Syntax:- MIN (column name)

Ex:- sclect min (salary) from empi

SQL string functions:-

String functions are used to perform an operation on input string and return an output string. following are the string functions defined for SQL

1. UPPER ()

Query:- SELECT UPPER (retail store employee name) from retail- store- employees WHERE retailstore-employee id=1;

2. LOWER ():-

Query:- SELECT LOWER (retail store-employee_name) from retail_store_employee WHERE retail store_employee.id 2';

3. LENGTH()

Query:- SELECT LENGTH(retailstore_employee_name) from retail store_employees WHERE retailstore_employee_id=1;

4) SUBSTR()

Query:- SELECT SUBSTR(retail store_employee.name, 1, 5) from retail_store_employee WHERE retail store_employee_id=1;

5. CONCAT()

QUERY:- SELECT CONCAT(retailstore_employee_name, ' ', department) from retail_store_employee where retail store_employee_id=1;

SQL Date and Time functions:-

→ The date and time functions are built in functions in the SQL. These functions can be used in SQL queries to perform various date and time operations, such as filtering records based on dates, calculating date differences, and formatting dates for display purposes

for storing a date or a date and time value in a database, MySQL offers the following date types.

| | |
|---|---|
| Date | format YYYY-MM-DD |
| DATETIME | format YYYY-MM-DD HH:MI:SS |
| TIMESTAMP | format: YYYY-MM-DD HH:MI:SS |
| YEAR | format YYYY or YY |

CURDATE():-

Query:- SELECT CURRENTDATE from dual;

CURTIME()

Query: SELECT current_time() from dual;

* ADDDATE (DATE, DAYS)

SQL> select ADDDATE ('2018-08-01', 31);

* ADDTIME (expr1, expr2)

SQL> select ADDTime ('2018-08-01 23:59:59.999999', '1 1:1:1.000002');

* DAY OF MONTH (date)

SQL> SELECT DayofMonth ('2018-02-15');

* Day of week (date)

SQL> select day of week ('2018-02-15');

* Day of year (date)

SQL> select day of year ('2018-02-15');

* MONTH (date)

SQL> SELECT MONTH ('2018-08-01');

* TIME (expr)

SQL> Select time ('2018-08-01 11:33:25');

* Sysdate:

SQL: select sysDate from Dual;

* next_day:

SQL: select Next_Day (sysdate, 'wed') from Dual;

* add_months:

SQL> select ADD_months (sysdate, 2) from Dual;

last_day:

SQL> select months_between(sysdate, HIREDATE) from Emp;

* least:

SQL> select least('10-JAN-07', '12-oct-07') from dual;

* Greatest

SQL> select greatest('10-JAN-07', '12-oct-07') from DUAL;

* Trunc:

SQL> select Trunc(sysdate, 'DAY') from DUAL;

* Round:

SQL> select Round(SYSDATE, 'DAY') from DUAL;

* TO_char:

SQL> select to_char(sysdate, "dd/mm/yy") from dual;

* to_date:

SQL> select to_date(sysdate, "dd/mm/yy") from dual;

CHARACTER FUNCTION:-

initcap(char): select initcap('hello') from dual;

lower (char): select lower('Hello') from dual;

Upper (char): select upper('Hello') from dual;

ltrim(char, [set]): select ltrim('seit', 'se') from dual;

rtrim(char, [set]): select rtrim('seit', 'it') from dual;

replace(char, search): select replace('Jack and jill', 'j', 'bl') from dual;

# STRING FUNCTIONS:-

Concat:-

Concat returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes.

SQL> select concat ('Oracle', 'corporation') FROM DUAL;

Oraclecorporation

Lpad:-

Lpad returns expr1, left-padded to length n characters with the sequence of characters in expr2.

SQL> select LPAD ('oracle', 15, '*') FROM DUAL;

Rpad:- RPAD returns expr1, right-padded to length n characters with expr2, replicated as many times as necessary

SQL> SELECT RPAD ('oracle', 15, '*') from DUAL;

Ltrim:- Returns a character expression after removing leading blanks:

SQL> select Ltrim ('SSmiTHSS', 'S') from DUAL;

Rtrim :- Return a character string after truncating all trailing blanks.

SQL> select RTRIM ('SSMITHSS', 'S') FROM DUAL;

Lower:- Returns a character expression after converting uppercase character data to lowercase

SQL> select lower ('PBMS') from DUAL;

Upper:- returns a character expression with lower case character data converted to uppercase

SQL> select upper ('dbms') from DUAL;

length :- Return the no. of characters, rather than the no of bytes of the given string expression excluding trailing blanks

SQL > select length ('database') from dual;

Substr :- Return part of character, binary, text, or image expression

SQL> Select SUBSTR ('ABCDEFGHIJK', 4) from dual;

Select substr('retail store-employee.name', 1, 4) from retail-store-employees;

instr :- The instr functions search string for sub string The function returns an integer indicating the position of the character in string that is the first character of this occurence.

SQL > select INSTR ('corporate floor', 'or', 3, 2) from dual;

Single Row operations :-

1. is NULL

Query :- select * from retail-store-employees where salaryay is NULL;

2) is NOT NULL

Query :- Select * from retail-store employees where salaryay is not NULL;

3) Like

Query :- select * from retail-store-employees where retailstore_employee_name like '%. Tohn%';

4) Not Like

Query :- Select * from retail-store-employees where retail store.employee-name NOT like '%.John%';

4-8-25

Task 3.

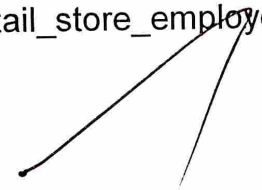
Developing queries with DML single row functions and operations.

```
SQL> select * from retail_store_employees;

RETAIL_STORE_EMPLOYEE_ID   STORE_ID RETAIL_STORE_EM     SALARY
DEPARTMENT
---------------------- ---------- --------------- ---------- ---------------
STORE_PHONENO EMPLOYEE_PHONENO
------------ ----------------
                28129       1569 koteswarao          50000 manager
 9515221655       9876543210

                28578       1569 chandrajith         40000 superwiser
 9515221655       9866671022

                27830       1569 nithish             60000 hr
 9515221655       9963892694


RETAIL_STORE_EMPLOYEE_ID   STORE_ID RETAIL_STORE_EM     SALARY
DEPARTMENT
---------------------- ---------- --------------- ---------- ---------------
STORE_PHONENO EMPLOYEE_PHONENO
------------ ----------------
                27578       1569 mahalakshmi         35000 salesman
 9515221655       9381746779


SQL> select count(*)from retail_store_employees;
```

```
 COUNT(*)
----------
         4
```

```
SQL> select sum(salary)from retail_store_employees;

 SUM(SALARY)
 -----------
      185000
```

```
 SQL> select avg(salary)from retail_store_employees;

 AVG(SALARY)
 -----------
       46250
```



```
  SQL> select max(salary)from retail_store_employees;

  MAX(SALARY)
  -----------
       60000
```

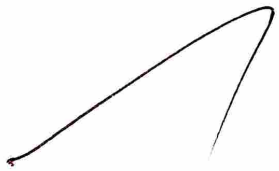
```
   SQL> select min(salary)from retail_store_employees;

   MIN(SALARY)
   -----------
        35000
```

SQL> select upper(retail_store_employee_name)from retail_store_employees where retail_store_employee_id=27830;

```
UPPER(RETAIL_ST
---------------
NITHISH
```

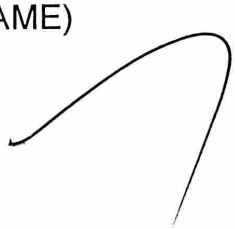
SQL> select lower(retail_store_employee_name)from retail_store_employees where retail_store_employee_id=27830;

```
LOWER(RETAIL_ST
---------------
nithish
```



SQL> select length(retail_store_employee_name)from retail_store_employees where retail_store_employee_id=27830;

```
LENGTH(RETAIL_STORE_EMPLOYEE_NAME)
----------------------------------
                                 7
```



SQL> select substr(retail_store_employee_name,1,4)from retail_store_employees where retail_store_employee_id=28129;

```
SUBSTR(RETAIL_ST
----------------
kote
```

```
SQL> select concat('retail_store_employee_name','salary')from retail_store_employees
where retail_store_employee_id=27578;

CONCAT('RETAIL_STORE_EMPLOYEE_NA
--------------------------------
retail_store_employee_namesalary

SQL> select sysdate from dual;

SYSDATE
---------
27-AUG-25

SQL> select next_day(sysdate,'wed')from dual;

NEXT_DAY(
---------
03-SEP-25

SQL> select add_months(sysdate,2)from dual;

ADD_MONTH
---------
27-OCT-25

SQL> select last_day(sysdate)from dual;
```

LAST_DAY(

---

31-AUG-25

SQL> select least('10-jan-07','12-oct-07')from dual;

LEAST('10

---

10-jan-07

SQL> select greatest('10-jan-07','12-oct-07')from dual;

GREATEST(

---

12-oct-07

SQL> select trunc(sysdate,'day')from dual;

TRUNC(SYS

---

24-AUG-25

SQL> select round(sysdate,'day')from dual;

ROUND(SYS

---

31-AUG-25

```
SQL> select to_char(sysdate,'dd\mm\yy')from dual;

TO_CHAR(
--------
27\08\25

SQL> select to_date(sysdate,'dd\mm\yy')from dual;

TO_DATE(S
---------
27-AUG-25

SQL> select concat('oracle','corpuration')from dual;

CONCAT('ORACLE','
-----------------
oraclecorpuration

SQL> select lpad('oracle',15,'*')from dual;

LPAD('ORACLE',1
---------------
*********oracle

SQL> select rpad('oracle',15,'*')from dual;
```

RPAD('ORACLE',1

---

oracle*********

SQL> select ltrim('ssmithss','s')from dual;

LTRIM(

---

mithss

SQL> select lower('dbms')from dual;

LOWE

---

dbms

SQL> select upper('dbms')from dual;

UPPE

---

DBMS

SQL> select rtrim('ssmithss','s')from dual;

RTRIM(

---

ssmith

SQL> select length('database')from dual;

LENGTH('DATABASE')
------------------
8

SQL> select substr('abcdefghij',3,4)from dual;

SUBS
----
cdef

SQL> select instr('corporate floor','or',3,2)from dual;

INSTR('CORPORATEFLOOR','OR',3,2)
--------------------------------
14

SQL> select * from retail_store_employees where salary is null;

no rows selected

SQL> select * from retail_store_employees where salary is not null;

RETAIL_STORE_EMPLOYEE_ID STORE_ID RETAIL_STORE_EM SALARY
DEPARTMENT

---

STORE_PHONENO EMPLOYEE_PHONENO

---

28129 1569 koteswarao 50000 manager
9515221655 9876543210

28578 1569 chandrajith 40000 superwiser
9515221655 9866671022

27830 1569 nithish 60000 hr
9515221655 9963892694

RETAIL_STORE_EMPLOYEE_ID STORE_ID RETAIL_STORE_EM SALARY
DEPARTMENT

---

STORE_PHONENO EMPLOYEE_PHONENO

---

27578 1569 mahalakshmi 35000 salesman
9515221655 9381746779

SQL> select * from retail_store_employees where retail_store_employee_name like '%koteswarao%';

RETAIL_STORE_EMPLOYEE_ID STORE_ID RETAIL_STORE_EM SALARY DEPARTMENT

---------- --------- ---------- --------- ----------

STORE_PHONENO EMPLOYEE_PHONENO

---------- ----------

28129 1569 koteswarao 50000 manager

9515221655 9876543210

SQL> select * from retail_store_employees where retail_store_employee_name not like '%koteswarao%';

RETAIL_STORE_EMPLOYEE_ID STORE_ID RETAIL_STORE_EM SALARY DEPARTMENT

---------- --------- ---------- --------- ----------

STORE_PHONENO EMPLOYEE_PHONENO

---------- ----------

28578 1569 chandrajith 40000 superwiser

9515221655 9866671022

27830 1569 nithish 60000 hr

9515221655 9963892694

27578 1569 mahalakshmi 35000 salesman

9515221655 9381746779

SQL> select * from retail_store_employees where salary between 30000 and 70000;

RETAIL_STORE_EMPLOYEE_ID STORE_ID RETAIL_STORE_EM SALARY DEPARTMENT

---

STORE_PHONENO EMPLOYEE_PHONENO

---

28129 1569 koteswarao 50000 manager

9515221655 9876543210

28578 1569 chandrajith 40000 superwiser

9515221655 9866671022

27830 1569 nithish 60000 hr

9515221655 9963892694

RETAIL_STORE_EMPLOYEE_ID STORE_ID RETAIL_STORE_EM SALARY DEPARTMENT

---

STORE_PHONENO EMPLOYEE_PHONENO

---

27578 1569 mahalakshmi 35000 salesman

9515221655 9381746779

| VEL TECH | |
|---|---|
| EX NO. | 3 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYSIS (5) | 5 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | 5 |
| TOTAL (25) | |
| SIGN WITH DATE | |

Result :

Thus the development of single row function and

5. Between:-

Query :- select * from retail_store_employees where salary between 50000 and 100000;

Task 1:- Using clauses, operators and functions in queries perform The query processing on database for different retrieval results of queries using DML, DRL operations using aggregate date, string, inbuilt functions, set clauses and operations.

a) Retrieve all the author who wrote in dbms.

b) Retrieve total number of books offered in the category program core.

c) Retrieve all authno and name who published books after 2000

d) Retrieve readers name and with letter 'a'

e) Retrieve number of readers studied in each department

sample output:-

| | |
|---|---|
| ECE | 800 |
| CSE | 850 |
| EEE | 1000 |

f) Retrieve all the female readers

g. Retrieve all the staff who came library yesterday.

| VEL TECH | |
|---|---|
| EX NO. | 3 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYSIS (5) | 5 |
| VIVA VOCE (5) | 4 |
| RECORD (5) | 5 |
| TOTAL (20) | 19 |
| SIGN WITH DATE | |

Result:- Thus The development of a queries with DML single-row function and operation has been executed successfully

14-8-25

Task-4.

Developing queries with DML multi-row functions and operators perform the advanced query processing (multirow abilities) and test its heuristics. Using the designing of optimal correlated and nested subqueries as finding summary statistics.

consider the schema for:-

Employees (emp_no, emp_name, department, deptno, salary, age)

Orders (emp_no, orderid, price, qty_ord, qty_hand)

item file (item id, item name, qty_ord, qty_hand, itemrate)

Queries using UNION, INTERSECT, MINUS.

* UNION:-

The union operator returns all distinct rows selected by two or more queries

SQL> select emp_no from employees;

output:

SQL> select emp_no from orders;

output:-

SQL> select emp_no from employees union select emp_no orders;

output:

* UNION ALL:-

SQL> select emp_no from employees union all select emp_no from orders

* intersect:-

SQL:- select emp_no from employees intersect select emp_no from orders;

* minus:-

SQL> select emp_no from employees minus select emp_no from orders;

Queries using group by, Having clause and order clause

GROUP BY:- This query is used to group to all the records into relation together for each and every value of a specific key(s) and then display them for a selected set of fields the relation.

SQL> select deptno, count(*) from employes group by dept no;

GROUP BY- HAVING:-

The Having clause was added to SQL because the where keyword could not be used with aggregate functions. The having clause must follow the group By clause in a query and must also precede the order by clause if used.

SQL> select deptno, count(*) from employes group by deptno having Dept no is not null;

ORDER By:-

This query is used to display a selected set of fields from a relation in an ordered manner base on some field.

Syntax:-

Select <column(s)> from <table Name> where [condition(s)] [order by <column name> [asc/] desc];

SQL> select empno, ename, salary from employees order by salary;

SQL> select empno, emp-name, salary from employes orderby salary desc;

SQL * plus having following operators:-

SQL> select salary + comm from emp-master;

salary + comm

SQL > select salary + comm net-sal from emp-master;

SQL> select 12*(salary+comm) annual-netsal from emp-master;

## Subqueries:-

SQL> Select * from employees

SQL> insert into employes select * from employes where emp-id in (select emp-id from employes);

SQL> update employes set salary = salary*10 where department in (select department from employes where department = 'sales';

Delete from employes where depart ment in (select department from employes where department = 'sales');

IN:

Query: SELECT * FROM employes WHERE department IN ('sales', 'marketing');

NOT IN:

Query:- select * from employes where department NOT IN ('sales', marketing');

EXISTS:-

Query:- select * from empbyes where Exists (select * from orders where orders.emp-no = (link unavailable));

NOT EXISTS:-

Query:- select * FROM employes where NOT EXISTS (select * from orders where orders.emp-no = (link unavailable));

ALL:-

Query:- Select * FROM employes WHERE salary > ALL (SELECT salary FROM employes WHERE department='sales');

ANY:-

Query:- SELECT * FROM employes WHERE salary > ANY (SELECT salary FROM employes WHERE department = 'sales');

Task-4.

Developing queries with DML multi-row functions and operators

```
SQL> create table employees(emp_no number(4),emp_name varchar(15),department
varchar(5),dept_no number(5),salary number(5),age number(2));
Table created.
SQL> desc employees
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 EMP_NO                                             NUMBER(4)
 EMP_NAME                                           VARCHAR2(15)
 DEPARTMENT                                         VARCHAR2(5)
 DEPT_NO                                            NUMBER(5)
 SALARY                                             NUMBER(5)
 AGE                                                NUMBER(2)
SQL> create table orders(emp_no number(4),order_id number(5),price
number(5),qty_ord
number(5),qty_hand number(5));
Table created.
SQL> desc orders
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 EMP_NO                                             NUMBER(4)
 ORDER_ID                                           NUMBER(5)
 PRICE                                              NUMBER(5)
 QTY_ORD                                            NUMBER(5)
 QTY_HAND                                           NUMBER(5)
SQL> create table itemfile(item_id number(5),item_name varchar(10),qty_ord
number(5),qty_hand number(5),item_rate number(5));
Table created.
SQL> desc itemfile
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 ITEM_ID                                            NUMBER(5)
 ITEM_NAME                                          VARCHAR2(10)
 QTY_ORD                                            NUMBER(5)
 QTY_HAND                                           NUMBER(5)
 ITEM_RATE                                          NUMBER(5)
SQL>  insert into employees values(21,'koti','cseds',126,60000,21);
1 row created.
SQL>  insert into employees values(22,'shivapriya','csed',128,80000,19);
1 row created.
SQL>  insert into employees values(23,'maha','aids',142,70000,18);
1 row created.
SQL>  insert into employees values(24,'bhasker','cseai',123,50000,20);
1 row created.
SQL>  insert into employees values(25,'chandrajith','cseai',123,90000,21);
1 row created.
SQL> select * from employees;
    EMP_NO EMP_NAME        DEPAR     DEPT_NO     SALARY        AGE ----------
---------- ----- ---------- ---------- ---------
        21 koti            cseds         126      60000
        22 shivapriya      csed          128      80000          21
        23 maha            aids          142      70000          19
                                                                  18
```

```
        24 bhasker                cseai
        25 chandrajith            cseai           123          50000          20
                                                  123          90000          21
SQL>  insert into orders values(21,789,500,5,4);
1 row created.
SQL>  insert into orders values(22,456,400,6,5);
1 row created.
SQL>  insert into orders values(23,123,600,4,3);
1 row created.
SQL>  insert into orders values(24,159,200,8,7);
1 row created.
SQL>  insert into orders values(25,357,900,3,2);
1 row created.
SQL> select * from orders;
    EMP_NO   ORDER_ID      PRICE    QTY_ORD   QTY_HAND ---------- ----------
---------- ---------- ----------
        21        789        500          5          4
        22        456        400          6          5
        23        123        600          4          3
        24        159        200          8          7
        25        357        900          3          2
SQL>  insert into itemfile values(1236,'dog',5,5,20000);
1 row created.
SQL>  insert into itemfile values(14324,'gold',4,4,30000);
1 row created.
SQL>  insert into itemfile values(5478,'pigy',5,6,10000);
1 row created.
SQL>  insert into itemfile values(9874,'bajji',10,9,1000);
1 row created.
SQL>  insert into itemfile values(2854,'silve',6,5,60000);
1 row created.
SQL> select * from itemfile;
   ITEM_ID ITEM_NAME     QTY_ORD   QTY_HAND  ITEM_RATE ---------- ----------
---------- ---------- ----------
      1236 dog                5          5      20000
     14324 gold               4          4      30000
      5478 pigy               5          6      10000
      9874 bajji             10          9       1000
      2854 silve              6          5      60000
SQL> select 22 from employees;
        22 ----------
        22
        22
        22
        22
        22
SQL> select emp_no from employees
  2  ;
    EMP_NO ----------
        21
        22
        23
        24
        25
SQL> select emp_no from orders;
```

```
   EMP_NO ----------
        21
        22
        23
        24
        25
SQL> select emp_no from employees union select emp_no from orders;
    EMP_NO ----------
        21
        22
        23
        24
        25
SQL> select emp_no from employees union all select emp_no from orders;
    EMP_NO ----------
        21
        22
        23
        24
        25
        21
        22
        23
        24
        25
10 rows selected.
SQL> select emp_no from employees intersect select emp_no from orders;
    EMP_NO ----------
        21
        22
        23
        24
        25
SQL> select emp_no from employees minus select emp_no from orders;
no rows selected
SQL> select dept_no,count(*) from employees group by dept_no;
   DEPT_NO   COUNT(*) ---------- ----------
       123          2
       128          1
       142          1
       126          1

SQL> select dept_no,count(*) from employees group by dept_no having dept_no is
not null;
   DEPT_NO   COUNT(*) ---------- ----------
       123          2
       128          1
       142          1
       126          1
SQL> select  emp_no,emp_name,age,salary from employees order by salary desc;
    EMP_NO EMP_NAME                  AGE     SALARY ---------- ----------
---------- ----------
        25 chandrajith                21      90000
        22 shivapriya                 19      80000
```

```
        23 maha
        21 koti                        18      70000
        24 bhasker                     21      60000
                                       20      50000
SQL> select salary+age from employees;
SALARY+AGE ----------
     60021
     80019
     70018
     50020
     90021
SQL> select * from employees;
    EMP_NO EMP_NAME        DEPAR     DEPT_NO     SALARY        AGE ----------
---------- ----- ---------- ---------- ----------
        21 koti               cseds        126      60000         21
        22 shivapriya         csed         128      80000         19
        23 maha               aids         142      70000         18
        24 bhasker            cseai        123      50000         20
        25 chandrajith        cseai        123      90000

SQL> insert into employees select * from employees where emp_no in (select
emp_no from
employees);
5 rows created.
SQL> select * from employees;
    EMP_NO EMP_NAME        DEPAR     DEPT_NO     SALARY        AGE ----------
---------- ----- ---------- ---------- ----------
        21 koti               cseds        126      60000         21
        22 shivapriya         csed         128      80000         19
        23 maha               aids         142      70000         18
        24 bhasker            cseai        123      50000         20
        25 chandrajith        cseai        123      90000         21
        21 koti               cseds        126      60000         21
        22 shivapriya         csed         128      80000         19
        23 maha               aids         142      70000         18
        24 bhasker            cseai        123      50000         20
        25 chandrajith        cseai        123      90000         21
10 rows selected.
SQL> update employees set salary=salary*10 where department in (select
department from
employees where department='sales');
0 rows updated.
SQL> update employees set salary=salary*1 where department in (select department
from
employees where department='csed');
2 rows updated.
SQL> select * from employees;
    EMP_NO EMP_NAME        DEPAR     DEPT_NO     SALARY        AGE ----------
---------- ----- ---------- ---------- ----------
        21 koti               cseds        126      60000         21
        22 shivapriya         csed         128      80000         19
        23 maha               aids         142      70000         18
        24 bhasker            cseai        123      50000         20
        25 chandrajith        cseai        123      90000         21
        21 koti               cseds        126      60000         21
```

```
        22 shivapriya           csed            128         80000         19
        23 maha                 aids            142         70000         18
        24 bhasker              cseai           123         50000         20
        25 chandrajith          cseai           123         90000         21
10 rows selected.
SQL> select * from employees where department in ('cseds','csed');
    EMP_NO EMP_NAME           DEPAR     DEPT_NO     SALARY        AGE ----------
-------------- ----- --------- --------- ----------
        21 koti                 cseds           126         60000         21
        22 shivapriya           csed            128         80000         19
        21 koti                 cseds           126         60000         21
        22 shivapriya           csed            128         80000         19
SQL> select * from employees where department not in ('cseds','csed');
    EMP_NO EMP_NAME           DEPAR     DEPT_NO     SALARY        AGE ----------
-------------- ----- --------- --------- ----------
        23 maha                 aids            142         70000         18
        24 bhasker              cseai           123         50000         20
        25 chandrajith          cseai           123         90000         21
        23 maha                 aids            142         70000         18
        24 bhasker              cseai           123         50000         20
        25 chandrajith          cseai           123         90000         21
6 rows selected.
SQL> select * from employees where exists (select * from orders where
orders.emp_no=22);
    EMP_NO EMP_NAME           DEPAR     DEPT_NO     SALARY        AGE ----------
-------------- ----- --------- --------- ----------
        21 koti                 cseds           126         60000         21
        22 shivapriya           csed            128         80000         19
        23 maha                 aids            142         70000         18
        24 bhasker              cseai           123         50000         20
        25 chandrajith          cseai           123         90000         21
        21 koti                 cseds           126         60000         21
        22 shivapriya           csed            128         80000         19
        23 maha                 aids            142         70000         18
        24 bhasker              cseai           123         50000         20
        25 chandrajith          cseai           123         90000         21
10 rows selected.
SQL> select * from employees where not exists (select * from orders where
orders.emp_no=22);
no rows selected
SQL> select * from employees where not exists (select * from orders where
orders.emp_no=24);
no rows selected
SQL> select * from employees where not exists (select * from orders where
orders.emp_no=21);
no rows selected
SQL> select * from employees where not exists (select * from orders where
orders.emp_no=22);
no rows selected
SQL> select * from employees where exists (select * from orders where
orders.emp_no=24);
    EMP_NO EMP_NAME           DEPAR     DEPT_NO     SALARY        AGE ----------
-------------- ----- --------- --------- ----------
        21 koti                 cseds           126         60000         21
```

```
        22 shivapriya             csed                128          80000          19
        23 maha                   aids                142          70000          18
        24 bhasker                cseai               123          50000          20
        25 chandrajith            cseai               123          90000          21
        21 koti                   cseds               126          60000          21
        22 shivapriya             csed                128          80000          19
        23 maha                   aids                142          70000          18
        24 bhasker                cseai               123          50000          20
        25 chandrajith            cseai               123          90000          21
10 rows selected.
SQL> select * from employees where salary > all (select salary from employees
where
department='csed');
    EMP_NO EMP_NAME                  DEPAR     DEPT_NO     SALARY          AGE ----------
---------- ----- ---------- ---------- ----------
        25 chandrajith            cseai               123          90000          21
        25 chandrajith            cseai               123          90000          21
SQL> select * from orders where order_id=(select order_id from orders where
order_id='123');
    EMP_NO   ORDER_ID      PRICE    QTY_ORD   QTY_HAND ---------- ----------
---------- ---------- ----------
        23        123        600          4          3
SQL> select * from orders where order_id=any(select order_id from orders);
    EMP_NO   ORDER_ID      PRICE    QTY_ORD   QTY_HAND ---------- ----------
---------- ---------- ----------
        21        789        500          5          4
        22        456        400          6          5
        23        123        600          4          3
        24        159        200          8          7
        25        357        900          3          2
SQL> select * from orders where order_id in(select order_id from orders);
    EMP_NO   ORDER_ID      PRICE    QTY_ORD   QTY_HAND ---------- ----------
---------- ---------- ----------
        21        789        500          5          4
        22        456        400          6          5
        23        123        600          4          3
        24        159        200          8          7
        25        357        900          3          2
SQL> select salary+price from employees,orders
  2  ;
SALARY+PRICE ------------
       60500
       80500
       70500
       50500
       90500
       60500
       80500
       70500
       50500
       90500
       60400
SALARY+PRICE ------------
       80400
```

```
        70400
        50400
        90400
        60400
        80400
        70400
        50400
        90400
        60600
        80600
SALARY+PRICE ------------
        70600
        50600
        90600
        60600
        80600
        70600
        50600
        90600
        60200
        80200
        70200
SALARY+PRICE ------------
        50200
        90200
        60200
        80200
        70200
        50200
        90200
        60900
        80900
        70900
50900
SALARY+PRICE
```

| | |
|---|---|
| PERFORMANCE (5) | 4 |
| RESULT AND ANALYSIS (5) | 4 |
| VIVA VOCE (5) | 4 |
| RECORD (5) | 3 |
| TOTAL (20) | 15 |
| SIGN WITH DATE | |

Result:- Thus the Developing with DML Multiple function, and operators was successfully executed

:-

Pi

O

SQL> ~~select~~ ~~from~~ order-master where order-no = (select order-no from orders where order-no = 'o001');

SQL > select * from order-master where order-no = (select order-no from orders );

SQL > select * from order-master where order-no = any (select order-no from order-detail);

SQL > select * from order-master where order-no in (select order-no from order-detail);

SQL > select * from order-detail where qty-ord = all (select qty-hand from item file where item-rate = 250

| VEL TECH | |
|---|---|
| EX NO. | |
| PERFORMANCE (5) | |
| RESULT AND ANALYS'S (5) | |
| VIVA VOCE (5) | |
| RECORD (5) | |
| TOTAL (20) | |
| N WITH DATE | |

Result:-

Developing queries with DML multi row functions and operators was successfully executed.

TASK-5

# WRITING JOIN QUERIES, EQUIVALENT, AND/OR RECURSIVE QUERIES.

Title:- implementation of different types of joins and recursive queries.

- A SQL JOIN combines records from two tables
- A JOIN locates related column values in the two tables
- A query can contain zero, one, or multiple JOIN operations
- INNER JOIN is the same as JOIN; the keyword INNER is optional.

Objective:-

To implement different types of Joins and recursive queries.

Theory:-

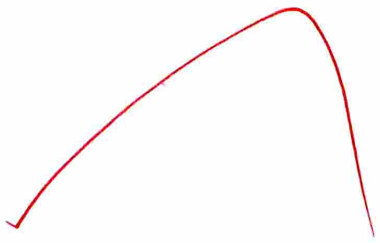
The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each. The join is actually performed by the 'where' clause which combines specified rows of tables

Syntax:-

Select column1, column2, column3,.. FROM table_name1, table_name2 WHERE Table-name1.column name = tablename2.column name;

Types of Joins:-

1. Simple join
2. Self join
3. outer join



Simple join:-

It is the most common type of join. It retrieves the rows from 2 tables having a common column and is further classified into equi join:

A join, which is based on a equalities, is called equi-join.

Example:-

SELECT * from item, cust where item.id = cust.id;

In the above statement, item_id = cust_id, performs the join statement. it retrieves rows from both the tables provided they both have the same id as specified by the where table clause. Since the where clause uses the comparison operator (=) to perform a join, it is said to be equijoin. It combines the matched rows of tables It can be used as follows;

- To insert records in the target table.
- to create tables and insert records in this table
- to update records in the target table
- to create views.

Non Equi-join:

It specifies the relationship between columns belonging to different tables by making use of relational operators other than '='.

Example:-

SELECT * from item, cost where item.id < cust.id;

table Aliases

It was used to make multiple-table queries shorted and more readable. we give an alias name to the table in the 'from' clause and use it instead of the name throughout the query.

Self join:

Joining of a table to itself is known as self-join. It joins one row in a table to another.

It can compare each (other) row of the table to itself and also with other rows of the same table.

Example:-

Select * from emp x, emp y where x.salary >= (select avg(salary) from x.emp where x.deptno2 y.deptno1;

## Outer Join:-

It extends the result of a sample Join. An outer join returns all the rows returned by simple Join as well as those rows from one table that do not match any row from the table. The symbol (+) represents outer join.

Different types of SQL JOIN's.

Here are the different types of Joins in SQL:

(INNER) JOIN: Returns records that have matching values in both tables.

SELECT column-name(s) FROM table INNER JOIN table2 ON Table 1 column_name = table2. column_name;

LEFT (OUTER) JOIN:- Returns all records from the left table, and the matched records from the right table.

SELECT column_name(s) FROM table LEFT JOIN Table2 ON Table 1. column-name = table 2. column_name;

RIGHT (OUTER) JOIN: Return all records from the right-table, and the matched records from the left table.

SELECT column name(s) FROM table1 RIGHT Join table2 ON table 1. column_name = table2. (dumn_name;

FULL (OUTER) JOIN: Return all records when There is a match in either left or right table SELECT Column_name (s) FROM Table1.

FULL OUTER JOIN Table2 ON table 1. column name = tb2. (dumn_name;

Consider the following two tables - member and borrowed

INNer Join Query:-

```
SELECT borrowed.membno,member.NAME FROM member
INNER JOIN borrowed
ON member.membNO = borrowed.membNO;
```

left JON QUILY:-

```
SELECT member.NAME, borrowed.membno FROM member.
LEFT JOIN borrowed or borrowed.membno=member.membNO;
```

SQL RIGHT JOIN keyword:-

```
SELECT member.NAME, borrowed.membNO FROM member.
RIGHT JOIN borrowed ON borrowed. membNO=mumber-membNO;
```

SQL FULL OUTER JOIN keyword;

Tip:- full outer join and full join are the same.

```
SELECT member.NAME, borrowed.membNO FROM mumber.
FULL JOIN borrowed ON borrowed.membNO=member.membNO;
```

LAB practic assignment:-

Using the tables "DEPARTMENTS" and "Employee" perform the following queries.

a) Display the employee details, departments that the departments are same in both the emp and dept.

b) Display the employee name and Department name by implementing a left outer join.

c) Display the employee name and Department name by implementing a right outer join.

d) Display the details of those who draw the salary greater than the average salary.

## Recursive Queries:-

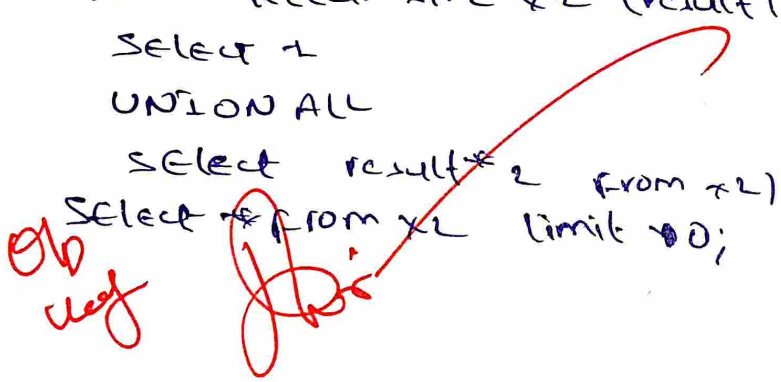
Syntax:-

```
With RECURSIVE [cte-name] (column n,...) AS(
        [non-recursive_term]
    union ALL
    [Recursive_term])
SELECT ... FROM [cte-name];
```

1: write a recursive query to create a multiplication table by 2.

Syntax:-

```
with recursive [cte-name] (colum)
With Recursive x2 (result) AS(
 SELECT 1
 UNION ALL
  SELECT result*2 from x2)
Select * from x2 limit 10;
```



21-8-25

# Writing Join Queries, Equivalent AND for Recursive Queries

```
SQL> create table member(memno number(2),name varchar(8));
Table created.
SQL> desc member;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MEMNO                                              NUMBER(2)
 NAME                                               VARCHAR2(8)
SQL> create table borrowed(memno number(2),bookid number(4));
Table created.
SQL> desc borrowed;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MEMNO                                              NUMBER(2)
 BOOKID                                             NUMBER(4)
SQL> insert into member values(1,'alice');
1 row created.
SQL> insert into member values(2,'bob');
1 row created.
SQL> insert into member values(3,'charlie');
1 row created.
SQL> insert into member values(4,'david');
1 row created.
SQL> desc member;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MEMNO                                              NUMBER(2)
 NAME                                               VARCHAR2(8)
SQL> select* from member;
     MEMNO NAME
---------- --------
         1 alice
         2 bob
         3 charlie
         4 david
SQL> insert into borrowed values(2,101);
1 row created.
SQL> insert into borrowed values(3,102);
1 row created.
SQL> insert into borrowed values(5,103);
1 row created.
SQL> select* from borrowed;
     MEMNO     BOOKID
---------- ----------
         2        101
         3        102
         5        103
SQL> select borrowed.memno, member.name from member inner join borrowed on
member.memno=borrowed.memno;
     MEMNO NAME
---------- --------
         2 bob
         3 charlie
SQL> select borrowed.memno, member.name from member left join borrowed on
member.memno=borrowed.memno;
```

```
     MEMNO NAME
---------- --------
         2 bob
         3 charlie
           david
           alice
SQL> select borrowed.memno, member.name from member right join borrowed on
member.memno=borrowed.memno;
     MEMNO NAME
---------- --------
         2 bob
         3 charlie
         5
SQL> select borrowed.memno, member.name from member full join borrowed on
member.memno=borrowed.memno;
     MEMNO NAME
---------- --------
           alice
         2 bob
         3 charlie
           david
         5
SQL> create table department(deptname varchar(5),deptid number(5));
Table created.
SQL> create table employee(empname varchar(15),empid number(5),salary
number(5));
Table created.
SQL> desc department;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNAME                                           VARCHAR2(5)
 DEPTID                                             NUMBER(5)
SQL> desc  employee;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNAME                                            VARCHAR2(15)
 EMPID                                              NUMBER(5)
 SALARY                                             NUMBER(5)
SQL> insert into department values('pyt',1);
1 row created.
SQL> insert into department values('eng',2);
1 row created.
SQL> insert into department values('tel',3);
1 row created.
SQL> insert into department values('java',4);
1 row created.
SQL> insert into department values('c',5);
1 row created.
SQL> desc department;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNAME                                           VARCHAR2(5)
 DEPTID                                             NUMBER(5)
SQL> select* from department;
```

```
DEPTN      DEPTID
-----  ----------
pyt             1
eng             2
tel             3
java            4
c               5
SQL> insert into employee values('karthikamam',2,50000);
1 row created.
SQL> insert into employee values('kalyan',3,10000);
1 row created.
SQL> insert into employee values('jagan',5,15000);
1 row created.
SQL> insert into employee values('mohan',6,15000);
1 row created.
SQL> insert into employee values('reddy',1,15000);
1 row created.
SQL> select* from employee;
EMPNAME                 EMPID     SALARY
--------------- ---------- ----------
karthikamam              2      50000
kalyan                   3      10000
jagan                    5      15000
mohan                    6      15000
reddy                    1      15000
SQL> select employee.empname,department.deptname from department inner join
employee on employee.empid=department.deptid;
EMPNAME         DEPTN
--------------- -----
karthikamam     eng
kalyan          tel
jagan           c
reddy           pyt
SQL> select employee.empname,department.deptname from department left join
employee on employee.empid=department.deptid;
EMPNAME         DEPTN
--------------- -----
karthikamam     eng
kalyan          tel
jagan           c
reddy           pyt
                java
SQL> select employee.empname,department.deptname from department right join
employee on employee.empid=department.deptid;

EMPNAME         DEPTN
--------------- -----
reddy           pyt
karthikamam     eng
kalyan          tel
jagan           c
mohan
```

| VEL T | |
|---|---|
| EX NO. | 5 |
| PERFORMA | 5 |
| RESULT AND | 5 |
| VIVA VOCE | 5 |
| RECORD | 5 |
| TOTAL (25) | 20 |
| SIGN WITH DATE | |

Result:- Thus, the Implementation of Queries, Equivalent, AND, OR Recursive Queries was executed successfully

Task-6

28-02-25

# PL / SQL procedures, functions, loops.

Aim:-

To implement PL/SQL procedures, functions and loops on number Theory and business scenarios.

procedure:-

PL/SQL is a combination of SQL along with the procedural features of programming languages it was developed by the Oracle corporation in the early 90's to enhance the capabilities of SQL. PL/SQL is one of the three key programming languages embedded in the oracle database, along with SQL itself and java

S.NO. Solution and Description:-

1. Declaration:

This section starts with the keyword DECLARE. It is an optional section and defines all variables cursors, sub programs, and other elements to be used in the program.

2. Execute command

The section is enclosed between the BEGIN and END and it is a mandatory section. It contains of the executable PL/SQL statement of the program. It should take atleast one executable line of code, which may be just a NULL command to indicate that nothing should be executed.

3. Exception HANDLING:

This section starts with the key word Exception. This optional section contains exceptions that handle errors in the program.

Task-6.

9-10-25

Simple program to print sentence:-

Syntax:-

```
DECLARE
<declaration section>
BEGIN
<Executable commands>
EXCEPTION
<Exception handling>
END;
```

Program:-

```
DECLARE
    message varchar2(20) := 'booking closed';
BEGIN
    dbms_output.put_line(message);
END;
```

Static Input:-

```
SQL > set serveroutput on
SQL > declare
  2  x number(5);
  3  y number(5);
  4  z number(9);
  5  begin
  6  x:=10;
  7  y:=12;
  8  z:= x+y;
  9  dbms_output.put_line('sum is' ||z);
 10  end;
 11  /
Sum is 22
PL/SQL procedure successfully completed
```

```
SQL> declare
  2  var1 integer;
  3  var2 integer;
  4  var3 integer;
  5  begin
  6  var1 := &var1;
  7  var3 := &var2;
  8  var3 := var1+var2;
  9  dbms_output.put_line(var3);
 10  end;
 11  /
Enter the value for var1: 20
old   6: var1 := &var1;
new   6: var1 := 20;
Enter value for var2: 30
old   7: var2 := &var2;
new   7: var2 := 30;
50
PL/SQL procedure successfully completed
```

```
declare
   hid number(3) := 100;
BEGIN
    IF (hid = 10) THEN
       dbms_output.put_line('value of hid is 10');
    ELSE (hid = 20) THEN
       dbms_output.put_line('value of hid is 20');
    else
       dbms_output.put_line('none of the value is matching');
```

```
    END IF;
    dbms_output.put_line('Exact value of hid is: ' || hid);
END;
/
```

None of the value is matching
Exact value of hid is: 100

PL/SQL procedure successfully completed.

```
DECLARE
    hid number(1);
    old number(1);
BEGIN
    <<outer_loop>>
    for hid IN 1..3 loop
        <<inner_loop>>
        for old IN 1..3 loop
            dbms_output.put_line('hid is:' || hid || 'and old is:'
|| old);
        END loop inner_loop;
    END loop outer_loop;
END;
/
```

hid is: 1 and old is: 1
hid is: 1 and old is: 2
hid is: 1 and old is: 3
hid is: 2 and old is: 1
hid is: 2 and old is: 2
hid is: 2 and old is: 3
hid is: 3 and old is: 1
hid is: 3 and old is: 2
hid is: 3 and old is: 3

PL/SQL proccdure successfully completed

sample program for only procedure:

```
SQL> Create or replace procedure csinformation
  2  (c_id in number, c_name in varchar2)
  3  is
  4  begin
  5  dbms_output.put_line('ID: ' || c_id);
  6  dbms_output.put_line('Name: ' || c_name);
  7  end;
  8  /
procedure created.
SQL> exec csinformation(101, 'raam');
PL/SQL procedure successfully completed.
SQL> exec csinformation(101, 'raam');
ID: 101
Name: raam
PL/SQL procedure successfully completed
```

Sample program for only function:

```
SQL> create or replace function csinformation
(c_id in number, c_name in varchar2)
Return varchar2
is
Begin
if c_id > 200 then
Return ('no booking available');
Else
Return ('Booking open');
End if;
End;
```

/

function created.

```
SQL> declare
  2  msg varchar2(200);
  3  begin
  4  msg := cs_information2(102,'raam');
  5. dbms_output.put_line(msg);
  6. end;
```

vehicle available

```
SQL> declare
  2  msg varchar2(200);
  3  begin
  4  msg := cs_information2(206,'raam');
  5  dbms_output.put_line(msg);
  6  end;
  7  /
```

NO vehicle available

PL SQL procedure successfully completed

| VEL TECH | |
|---|---|
| EX NO. | 6 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYSIS (5) | 4 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | 5 |
| TOTAL (20) | 19 |
| SIGN WITH DATE | |

Result:- To Implement pl/sql program, procedure function and loops on number Theory and business queries.

Task-6

PL/SQL procedures / functions, loops.

Simple program to print sentence

```
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
  2    x NUMBER(5);
  3    y NUMBER(5);
  4    z NUMBER(9);
  5  BEGIN
  6    x := 10;
  7    y := 12;
  8    z := x + y;
  9    DBMS_OUTPUT.PUT_LINE('Sum is: ' || z);
 10  END;
 11  /
Sum is: 22
PL/SQL procedure successfully completed.

SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
  2    var1 INTEGER;
  3    var2 INTEGER;
  4    var3 INTEGER;
  5  BEGIN
  6    var1 := &var1;
  7    var2 := &var2;
  8    var3 := var1 + var2;
  9    DBMS_OUTPUT.PUT_LINE('Sum is: ' || var3);
 10  END;
 11  /
Enter value for var1: 20
old   6:   var1 := &var1;
new   6:   var1 := 20;
Enter value for var2: 30
old   7:   var2 := &var2;
new   7:   var2 := 30;
Sum is: 50

PL/SQL procedure successfully completed.
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
  2    hid NUMBER(3) := 100;
  3  BEGIN
  4    IF hid = 10 THEN
  5      DBMS_OUTPUT.PUT_LINE('Value of hid is 10');
  6    ELSIF hid = 20 THEN
  7      DBMS_OUTPUT.PUT_LINE('Value of hid is 20');
  8    ELSIF hid = 30 THEN
  9      DBMS_OUTPUT.PUT_LINE('Value of hid is 30');
 10    ELSE
 11      DBMS_OUTPUT.PUT_LINE('None of the values is matching');
 12    END IF;
 13
```

```
  14    DBMS_OUTPUT.PUT_LINE('Exact value of hid is: ' || hid);
  15  END;
  16  /
None of the values is matching
Exact value of hid is: 100


SQL> DECLARE
  2    hid NUMBER(1);
  3    oid NUMBER(1);
  4  BEGIN
  5    <<outer_loop>>
  6    FOR hid IN 1..3 LOOP
  7      <<inner_loop>>
  8      FOR oid IN 1..3 LOOP
  9        dbms_output.put_line('hid is: ' || hid || ' and oid is: ' || oid);
 10      END LOOP inner_loop;
 11    END LOOP outer_loop;
 12  END;
 13  /
hid is: 1 and oid is: 1
hid is: 1 and oid is: 2
hid is: 1 and oid is: 3
hid is: 2 and oid is: 1
hid is: 2 and oid is: 2
hid is: 2 and oid is: 3
hid is: 3 and oid is: 1
hid is: 3 and oid is: 2
hid is: 3 and oid is: 3
PL/SQL procedure successfully completed.

SQL> SET SERVEROUTPUT ON;
SQL> CREATE OR REPLACE PROCEDURE csinformation (
  2      c_id   IN NUMBER,
  3      c_name IN VARCHAR2
  4  ) IS
  5  BEGIN
  6      dbms_output.put_line('ID: ' || c_id);
  7      dbms_output.put_line('Name: ' || c_name);
  8  END;
  9  /

Procedure created.
SQL> exec csinformation(101,'raam');
ID: 101
Name: raam

PL/SQL procedure successfully completed.


SQL> CREATE OR REPLACE FUNCTION csinfo_func (
  2      h_id   IN NUMBER,
  3      c_name IN VARCHAR2
  4  ) RETURN VARCHAR2
```

Task No:- 7

4-9-25

Writing PL/SQL using loops

Aim:-

TO write PL/SQL program using loops for printing prime number customer Id's and for demonstrating loop control in different scenarios.

procedure:-

1. Start a PL/SQL block or procedure
2. use a cursor (if required) to fetch customer Id's from a table.
3. For each id, chck whether it is prime number using a loop
5. print the result using DBMS_output.put_line
6. End the block.

Example 1:- Using while loop with cursor prime check using while loop.

```
Create OR Replace PROCEDURE print_prime_customerIds
   CURSOR cust_cur IS
      SELECT customer_id FROM customers;
   v_id NUMBER;
   v_is_prime BOOLEAN;
   v_i NUMBER;
BEGIN
   OPEN cust_cur;
   loop
     FETCH cust_cur INTO v_id;
      EXIT WHEN cust_cur%NOTFOUND;
     IF v_id < 2 THEN
        v_is_prime := FALSE;
     ELSE
        v_is_prime := TRUE;
        v_i := 2;
        WHILE v_i <= TRUNC (SQRT(v_id)) loop
           IF MOD (v_id, v_i) = 0 THEN
              v_is_prime := false;
              EXIT;
           END IF;
           v_i := v_i + 1;
        END loop;
     END IF;
     IF v_is_prime THEN
        DBMS_output.put_line ('prime customer ID: ' || v_id );
```

```
    END LOOP;
    close cust_cur;
END
```

-this procedure check all customer IDs and print the prime ones using a WHILE loop

Example:-

Using FOR Loop for first N prime numbers.

```
create OR REPLACE PROCEDURE print_first_n_prims(n NUMBER) IS
    v_num NUMBER := 2;
    v_count NUMBER := 0;
    v_is_prime BOOLEAN;
BEGIN
  WHILE v_count < n LOOP
      v_is_prime := TRUE;
      -- prime check using for loop
      FOR i IN 2..TRUNC(SQRT(v_num)) LOOP
          IF MOD(v_num, i) = 0 THEN
              v_is_prime := FALSE;
              EXIT;
          END IF;
      END LOOP;
      IF v_is_prime THEN
          DBMS_OUTPUT.PUT_LINE('prime: ' || v_num);
          v_count := v_count + 1;
      END IF;
      v_num := v_num + 1;
  END LOOP;
END;
```

This procedure print the first N prime numbers using a FOR LOOP

FOR EXAMPLE:-

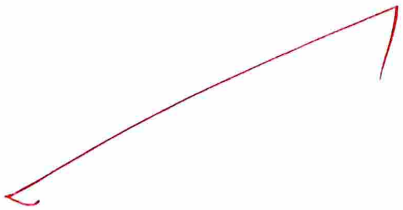
```
BEGIN
    print_first_n_primes(10); --prints first 10 prime numbers.
END;
```

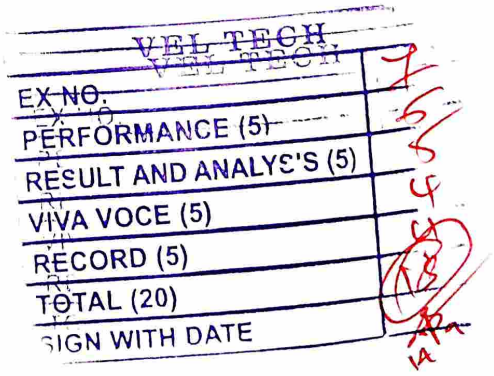
| VEL TECH | |
|---|---|
| EX NO. | |
| PERFORMANCE (5) | |
| RESULT AND ANALYS'S (5) | |
| VIVA VOCE (5) | |
| RECORD (5) | |
| TOTAL (20) | |
| IGN WITH DATE | |

Result:- Thus to write a PL/SQL program using loop for printing prime number customer Id's and for demonstrating loop control in different scenarios, was executed successfully

```
declare
  2  lo number(3);
  3  hi number(3);
  4  n number(2);
  5  m number(2);
  6  c number(20);
  7  begin
  8  dbms_output.put_line('enter the customer id from to limit:');
  9  lo:=&lo;
 10  hi:=&hi;
 11  for n in lo.. hi
 12  loop
 13  c:=0;
 14  for m in 1.. n
 15  loop
 16  if mod(n,m)=0 then
 17  c:=c+1;
 18  end if;
 19  end loop;
 20  if c<=2 then
 21  dbms_output.put_line(n||'\n');
 22  end if;
 23  end loop;
 24  end;
 25  /
Enter value for lo: 101
old   9: lo:=&lo;
new   9: lo:=101;
Enter value for hi: 120
old  10: hi:=&hi;
new  10: hi:=120;
enter the customer id from to limit:
101\n
103\n
107\n
109\n
113\n
113\n
PL/SQL procedure successfully completed.

SQL> declare
  2  bk number(5);
  3  s number:=0;
  4  r number;
  5  len number;
  6  m number;
  7  begin
  8  bk:=&bk;
  9  m:=bk;
 10  len:=length(to_char(bk));
 11  while bk>0
 12  loop
 13  r:=mod (bk,10);
 14  s:=s+power(r,len);
```

```
15  bk:=trunc(bk/10);
16  end loop;
17  if
18  m=s
19  then
20  dbms_output.put_line('given number is armstrong');
21  else
22  dbms_output.put_line('given number is not an armstrong');
23  end if;
24  end;
25  /
Enter value for bk: 1634
old   8: bk:=&bk;
new   8: bk:=1634;
given number is armstrong
PL/SQL procedure successfully completed.
```






Result:-

Thus to write pl/sql program using loop for printing prime number customgrid's and for demonstrating loop control in different scenarios was executed

TASK 8:-

Normalizing databases using functional dependence upto BCNF (TOOL: GUI Table Normalization Tool, ALM: Jigsaw)

Upon relational table created in task 2, perform normalization upto BCNF based on given Dependences as following for the assumed relations specified below.

Employee Database:

1. Identify employee attribute: Employee ID, Name, Department, Job_title manger_ID, Hire_Date, salary.
2. Define relational schema: Employee (Employee ID, Name, Department, Job_title, manager ID, Hire_Date, salary).
3. Determine functional dependence (FD's) between attributes:
   - Employee ID -> Name, Department, Job_title, Manger_ID, Hire_Date, salary
   - Department -> manager ID
   - manager ID -> Name

Step 2:- convert to 1NF

1. Eliminate repeating group or arrays (none in this example)
2. Create seperate table for each repeating group (none in this example).

Step 3: convert to 2NF.

1. Ensure each non key attribute depends on the entire primary key
2. move non-key attribute to seperate tables if they depends on only part of the primary key.

→ Create department table: (Department ID, manage ID, Name).

- Create employe table: Employe (Employee ID, Name, Department_ID, Job_title, Hire_Date, salary).

Step 4:- Convert to 3NF:-

1. Ensure there are no transitive dependencies.
2. Move non-key attributes to separate tables if they depend on another non-key attribute.
   - Create manager table: Manager (manager_ID, Name).
   - Update Department table: Department (Department_ID, manager_ID).

Step 5: Convert to BCNF:-

1. Ensure every determinant is a candidate key
2. Check for overlapping candidate keys
3. Decompose relations to eliminate redundancy.
   - NO. further decomposition needed

Using Griffith Tool.

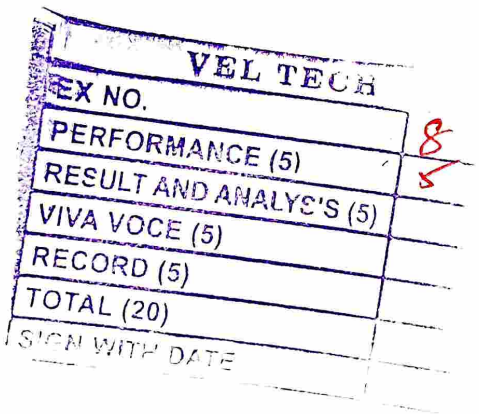
1. Input relational schema and functional dependencies.
2. Griffith tool generates a dependency graph.
3. Analyse the graph to identify normalization graph
4. Apply normalization rules to transform the schema
5. Verify the resulting schema meets BCNF criteria

Griffith Tool's Steps:-

1. Create a new project in griffith
2. Define the relational schema and FDs
3. Run the "Dependency Schema Graph" tool.
4. Analyse the graph for normalization issues
5. Apply transformations using the "Normalise" tool.
6. Verify BCNF compliance using the "BCNF check" tool.

Normalized schema:

1. Employee (Employee_ID, Name, Department_ID, Job_Title, Hire_Date, Salary).
2. Department (Department_ID, manager_ID).
3. manager (manager_ID, Name)




Result:-

Thus the Normalizing database using functional dependence upto BINF was Run and executed Successfully

11-9-25

TASK-8

Normalizing database using functional dependencies upto BCNF.

# FUNCTIONAL DEPENDENCY :

## Attributes in Table

*Separate attributes using a comma ( , )*

employee_id, name, department, job_title, manager_id, hire_date, salary

## Functional Dependencies

employee_id × → name × department × job_title × hire_date × manager_id × salary ×

Delete

Add Another Dependency

# NORMAL FORM :

## Check Normal Form

**2NF**
The table is in 2NF

**3NF**
The table is in 3NF

**BCNF**
The table is in BCNF

## Show Steps

2NF

find all candidate keys. The candiates keys are { employee_id}, The set of key attributes are: { employee_id }
for each non-trivial FD, check whether the LHS is a proper subset of some candidate key or the RHS are not all key attributes
checking FD: employee_id –> name,department,job_title,hire_date,manager_id,salary

3NF

find all cadnidate keys. The candiates keys are { employee_id}, The set of key attributes are: { employee_id }
for each FD, check whether the LHS is superkey or the RHS are all key attributes
checking functional dependency employee_id –> name,department,job_title,hire_date,manager_id,salary

BCNF

A table is in BCNF if and only if for every non-trivial FD, the LHS is a superkey.

Result
Thu,

# CONVERT 2NF :

## Normalize to 2NF

Attributes

employee_id | name | department | job_title | manager_id | hire_date | salary

Functional Dependencies

employee_id → name, department, job_title, hire_date, manager_id, salary

Show Steps

```
First, find the minimal cover of the FDs, which includes the FDs
employee_id --> name
employee_id --> department
employee_id --> job_title
employee_id --> hire_date
employee_id --> manager_id
employee_id --> salary
Initially rel[1] is the original table:
Round1: checking table rel[1]
***** The table is in 2NF already, send it to output *****
```

# CONVERT 3NF :

## 1NF to 3NF

Attributes

employee_id | name | department | job_title | manager_id | hire_date | salary

Functional Dependencies

```
employee_id → name
employee_id → department
employee_id → job_title
employee_id → hire_date
employee_id → manager_id
employee_id → salary
```

Show Steps

```
Table already in 3NF
```

Result :-

dependence

## CONVERT BCNF :

### Normalize to BCNF

Attributes

employee_id | name | department | job_title | manager_id | hire_date | salary

Functional Dependencies

employee_id → name department job_title hire_date manager_id salary

### Show Steps

Table already in BCNF, return itself.

| VEL TECH | |
|---|---|
| EX NO. | 6 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYSIS (5) | 5 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | 5 |
| TOTAL (20) | 20 |
| SIGN WITH DATE | |

Result :- Thus the Normalizing database using functional dependence upto BCNF was Run and executed Successfully

Backing up and recovery in database.

perform following backup and recovery scenarios

a) Recovering a NOARCHIVELOG Database with Incremental Backups

b) Restoring the server parameter file.

c) performing Recovery with a Backup control file

scenario 1:- Recovery with a NOARCHIVELOG Database with Incremental Backups

step 1:- Backup Database

BACKUP DATABASE [database_name] TO Disk = 'backup_file.bak' WITH NOFORMAT, NOINIT, NAME = "Full Database Backup', SKIP, REWIND, NOUNLOAD, STATS = 10

-- step 2:- create incremental Backup.

BACKUP DATABASE [database_name] TO DISK = 'incremental-backup.bak' WITH DIFFERENTIAL, NOFORMAT, NOINIT, NAME : 'Incremental Database Backup', SKIP REWIND, NOUNLOAD, STATS = 10.

-- step 3: Simulate Data loss

-- Incremental delete or modify data

-- step 4: Restore Database

RESTORE DATABASE [database_name] FROM DISK = 'backup_file.bak', WITH REPLACE

step 5:- Apply Incremental Backup

RESTORE DATABASE [database_name] FROM DISK = 'incremental_backup.bak' WITH REPLACE.

STEP 6:- Recover Database

RECOVER DATABASE [database_name]

STEP 7:- open Database

ALTER DATABASE [data_base_name] SET ONLINE

Scenario 2:-

Restoring the server parameter file (spfile)

Step 1: Back up spfile

BACK UP SERVER PARAMETER FILE TO FILE 'spfile.bak';

Step 2: Simulate spfile loss

- Delete or modify spfile

Step 3:- Restore spfile

STARTUP MOUNT

Restore SERVER PARAMETER FILE from file 'spfile.bak';

Scenario 3:-

performing Recovery with a Backup control file

Step 1:- Back up control file

Back up CONTROLFile TO file 'control file.bak';

step 2:- simulate control file

Delete or modify control file

Step 3:- Restore control file

STARTUP MOUNT

Restore controlfile from file 'control file.bak';

ALTER CONTROLFILE REUSE;

Step 4:-

Recover database

Recover database using Backup controlfile;

Step 5:- Open database

ALTER DATABASE OPEN RESETLOGS;

SQL Server Commands;

- BACK UP DATABASE
- RECOVER DATA BASE
- RECOVER DATA BASE
- ALTER DATA BASE
- BACK UP PARAMETER FILE
- RESTORE SERVER PARAMETER FILE
- BACKUP CONTROLFILE
- RESTORE CONTROLFILE

| VEL TECH | |
|---|---|
| EX NO. | 9 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYSIS (5) | 5 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | 20 |
| TOTAL (20) | |
| SIGN WITH DATE | |

RESULT:-

Thus the Implement of Backing up and Recovery in data base was run and executed successfully

CRUD OPERATIONS IN DOCUMENT DATABASES.

Aim:-

To perform mongoose using NPM design own Mongo DB designing document database and performing CRUD operations like creating, inserting quering finding and removing operations

steps:-

step1:- install Mongo db using following link.

http://www.mongodb.com/try/download/community

step2:- install Mongosh using the below ling

https://www.mongodb.com/docs/mongodb-shell/#download-and-install-mongosh

step3:- To add the mongo DB shell binary's location to your path environment vaiable. open the control panel.

In the system and security category click System

step4:- click the advanced system setting the system properties modal displays click environment Vaiables

In the system vaiables section, select path and click edit. The edit environment variable model display

step4:- open mongo shell 4.0 from c:/program files/mongo DB/server/bin/mongod.exe

step5:-

Type the CRUD commands given IN TEXT FILE

# CRUD OPERATIONS:-

```
db.create collection ("mylab")
{"ok":1}
>db.mylab.insertOne({item:"canvas",qty:100,tags:
["cotton"],size:{h:28,w:35.5,uom:"cm"}})
{
"acknowledged": true
"insertedId": objectId("627d1jacc73990c074e
}                    6397c")
>db.mylab.find({item:"canvas"})
{"_id": objectId("627d1jacc73990c074e
6397c"), "item": "canvas", "qty". 1000. "tags":[
"cotton"], "size": {"h":28;"w" 35.5, "uom": "cm"}}
>
db.mylab.insertMany([{item:"Journal",qty:25,tags:
["blank","red"],"red"], size:{h:14,w:21,uom:"cm"}},
{item:"mat", qty:85, tags:{"gray"}, size:{h:27.9,w:35.5,
uom:"cm"}}, {item:"mousepad:" mousepad;qty:25,tags:["gel
","blue"], size:{h:19,w:22.85,uom:"com"}}])
{
"acknowledged":True
"insertedIds":[
        objectId("627d1598c73990c074e6397d"),
        objectId("627d1598c73990c74e6397c"),
        objectId("627d1598c73990c074e6397e")
    ]
]
>db.mylab.find({},{item:1,qty:1})
{"_id":objectId("627d1jacc73990c074e6397c"),"item":
"canvas", "qty":100}
{"_id": objectId("627d1598c73990c07e6397d"), "item":
"Journal", "qty", :25}
```

{"_id": objectId("627d1598c7399074 6397c"),
e"), "item": 'mat", "qty": 85}

{"_id": objectId("627d1598c73990c074e6 3974e
6397f"), "item": mousepad", "qty": 25}

{
"_id": objectId("627d13acc7990acc074e6397c")
"item": "canvas",
"qty": 100
}
{
"id": objectId("627d1598c73990c074e6397c"),
"item": "mat", "qty": 85}
}
"_id": objectId("627d1598c73990c074e6397f"),
"item", :"mousepad",
"qty": 25
}
> db.mylab.find({item:"canvas"}).pretty().sort({item:-1})
{
"_id": objectId("627d13acc73990c074e6397c"),
"item": "canvas",
"qty": 100,
"tags": [
"cotton"],
"size": {
"h": 28,
"w": 35.5
"uom": "cm"
}
}
>db.mylab.delete one({item: "Journal"})
...
...
>db.mylab.find({}, {item:1, qty:1}).pretty()
{

"_id": objectId("627d173acc77900c074c6797c"),
"item": "conor",
"qty": 10
}
{
"_id": objectId("627d198c77990c074c6797d"),
"item": "Tournal",
"qty": 25
}
{"_id": objectId("627d1598c77990c074c6797e"), "item":
"mat", "qty": 85}
{
"_id": objectId("627d1598c77990c074c6797f") "itemlie":
"mousepad", "qty": 25}

| VEL TECH | |
|---|---|
| EX NO. | 10 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYS'S (5) | 5 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | 5 |
| TOTAL (20) | 20 |
| SIGN WITH DATE | |

Result:-

The implementation of CRUD operation like creating inserting, finding and removing operations using MongoDB is Successfully executed

Task 10

Taking up and recovery in database.
CRUD operation IN DOCUMENT DATABASES.

```
db.createCollection("mylab")

db.mylab.insertOne({item:"canvas",qty:100,tags:["cotton"],size:{h:28,w:35.5,uom:"cm"}})

db.mylab.find({item:"canvas"})

db.mylab.insertMany([{item:"journal",qty:25,tags:["blank","red"],size:{h:14,w:21,uom:"cm"}
},

{item:"mat",qty:85,tags:["gray"],size:{h:27.9,w:35.5,uom:"cm"}},

{item:"mousepad",qty:25,tags:["gel","blue"],size:{h:19,w:22.85,uom:"cm"}}])

db.mylab.find({},{item:1,qty:1})

db.mylab.find({},{item:1,qty:1}).pretty()

db.mylab.find({item:"canvas"}).pretty().sort({item:-1})

db.mylab.deleteOne({item:"journal"}

db.mylab.find({},{item:1,qty:1}).pretty()
```

output:

Output:

```
{ "ok" : 1 }
{
        "acknowledged" : true,
        "insertedId" : ObjectId("68efcd6e4b9c34878362b38e")
}
{ "_id" : ObjectId("68efcd6e4b9c34878362b38e"), "item" : "canvas", "qty" : 100, "tags" : [ "cotton" ], "size" : { "h" : 28, "w" : 35.5, "uom" : "cm" } }
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("68efcd6e4b9c34878362b38f"),
                ObjectId("68efcd6e4b9c34878362b390"),
                ObjectId("68efcd6e4b9c34878362b391")
        ]
}
{ "_id" : ObjectId("68efcd6e4b9c34878362b38e"), "item" : "canvas", "qty" : 100 }
{ "_id" : ObjectId("68efcd6e4b9c34878362b38f"), "item" : "journal", "qty" : 25 }
{ "_id" : ObjectId("68efcd6e4b9c34878362b390"), "item" : "mat", "qty" : 85 }
{ "_id" : ObjectId("68efcd6e4b9c34878362b391"), "item" : "mousepad", "qty" : 25 }
{
        "_id" : ObjectId("68efcd6e4b9c34878362b38e"),
        "item" : "canvas",
        "qty" : 100
```

```
{       "_id" : ObjectId("68efcd6e4b9c34878362b38f"),
        "item" : "journal",
        "qty" : 25
}
{ "_id" : ObjectId("68efcd6e4b9c34878362b390"), "item" : "mat", "qty" : 85 }
{
        "_id" : ObjectId("68efcd6e4b9c34878362b391"),
        "item" : "mousepad",
        "qty" : 25
}
{
        "_id" : ObjectId("68efcd6e4b9c34878362b38e"),
        "item" : "canvas",
        "qty" : 100,
        "tags" : [
                "cotton"
        ],
        "size" : {
                "h" : 28,
                "w" : 35.5,
                "uom" : "cm"
        }
}
```

```
{
"_id" :ObjectId("627d13acc73990c074e6397c"),
"item" : "canvas",
"qty" : 100
}
{
"_id" :ObjectId("627d1598c73990c074e6397d"),
"item" : "journal",
"qty" : 25
}
{ "_id" :ObjectId("627d1598c73990c074e6397e"), "item" : "mat", "qty" :85 }
{
 "_id" :ObjectId("627d1598c73990c074e6397f"),"item" : "mousepad","qty" : 25}
```

| VEL TECH | |
|---|---|
| EX NO. | 10 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYSIS (5) | 5 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | 5 |
| TOTAL (20) | 20 |
| SIGN WITH DATE | 25-9-25 |

Result:-
Thus implementation of inserting, finding operation (Run operation like creating

16-10-25

# TASK11 - CRUD OPERATIONS IN GRAPH DATABASES

Aim:- To perform CRUD operations like creating, inserting, quering, finding, deleting, operations on graph spaces.

## * CREATING A NEW PROPERTIES.

properties are the key values pair using a node store data. You can create a node with properties using CREATE clause, you need to specify these properties separated by commas with in the flower brackets "{ }"

Syntax:-

following is the syntax to create with properties

CREATE (node: label {key1: value key2: value, ...})

## * Returning the CREATED Node.

To verify the creation of the node, type and execute the following query in the dollar prompt.

Match (n) RETURN n

## * Creating Relationships:

we can create a relationship using the CREATE clause. we will specify relationship with in the square braces "[]" depending on the relationships it is placed between hyphen "-" and arrow "->" as shown in the following syntax

Syntax:-

* Creating a Relationship between the Existing nodes

-> you can also create a relationship between the existing nodes using the MATCH clause

SYNTAX:-

following the syntax to delete a particular node from Neo 4j using the DELETE clause

MATCH (node: label {properties ... })
DETACH DELETE node.

create a graph database from student course registration, create student and dept node and insert values of properties

create (n: student {sid: "VTU 14 500",
sname: "John",
deptname: "ISE"}
)

-> create (n: student {sid" VTU 14 501",
Sname: "Dharsana",
dept name: "EEE"})

-> create (n: student {sid: "VTU 14 502",
sname: "vijay",
dept name: "CSE"
})

* create (n: dept {dept name: "CSE", deptid: "d001"})

* select all the nodes in your DB using match Command

* match (n) return (n)

-> match (n: student) return (n)

a) create relation ship b/w student and CSE.

MATCH (s: student), (d: dept) WHERE s.name = "vijay" AND d.department = 'CSE'
CREATE (s)-[st: STUDIED_AT] -> (d)
return s,d

MATCH (

AND d.depetname :' CSE'

CREATE (s)-[:STUDIED_AT]->(d)

return s, d

- Match(n) return(n)

b) Delete a node from student

Match (n: student {sname:' Dhaq sena'}) DELETE (n)

| VEL TECH | |
|---|---|
| EX NO. | |
| PERFORMANCE (5) | |
| RESULT AND ANALYSIS (5) | |
| VIVA VOCE (5) | |
| RECORD (5) | |
| TOTAL (20) | |
| SIGN WITH DATE | |

Result:-

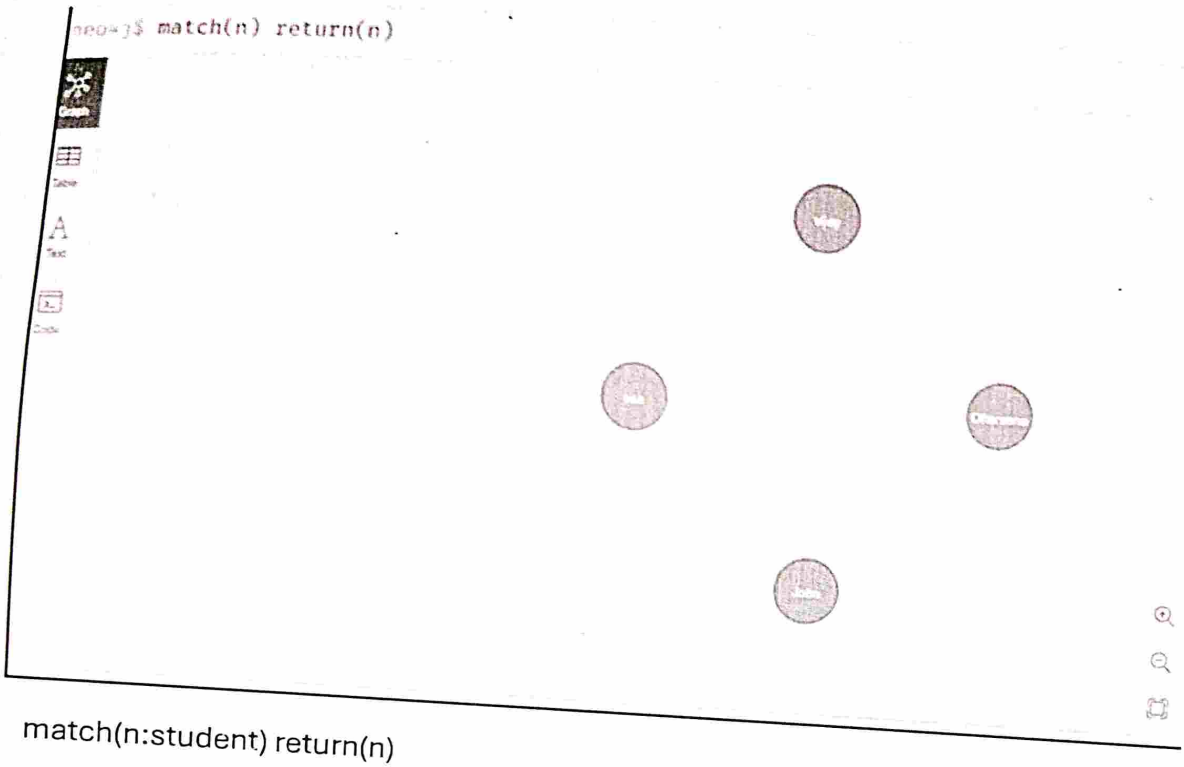
The implementation of CRUD operations like creating, inserting, finding, and removing operations using

TASK 11:- 16-10-25

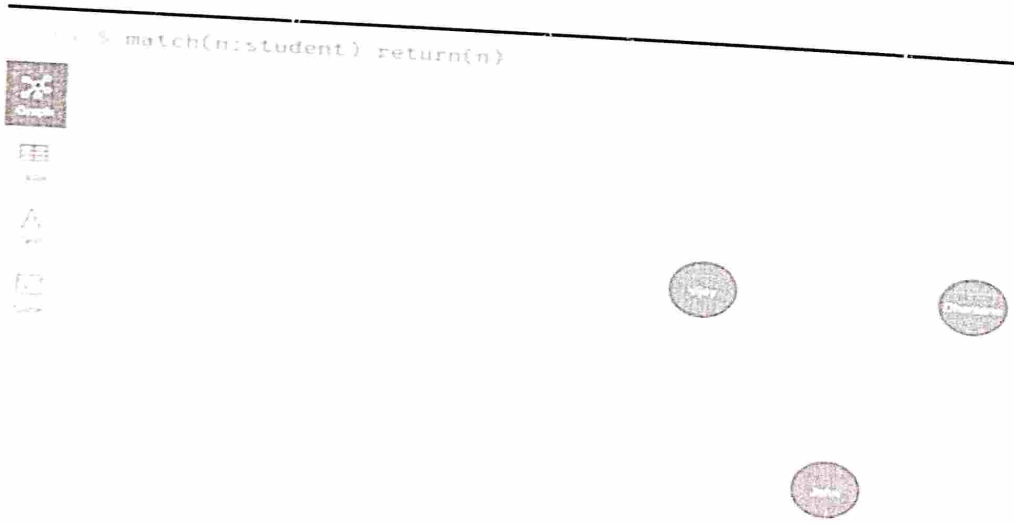
CRUD operation IN Graph DATA TIANEJ.

**Select all the nodes in your database using match command.**

*match(n) return(n)*




match(n:student) return(n)




**a) Create relationship between student and cse .**

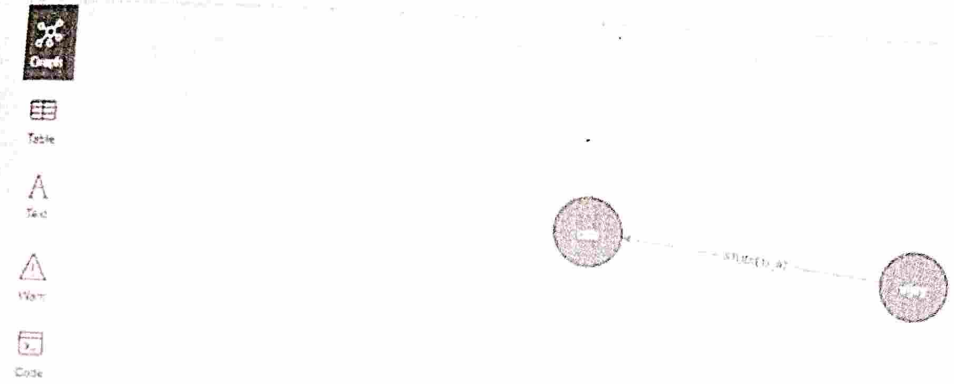
MATCH(s:student),(d:dept) WHERE s.Sname ='vijay' AND d.deptname='cse'

CREATE(s)-[st:STUDIED_AT]->(d)

return s,d

Result:-

```
MATCH(s:student),(d:dept) WHERE s.Sname ='vijay' AND d.deptname='cse'
CREATE(s)-[st:STUDIED_AT]->(d)
return s,d
```

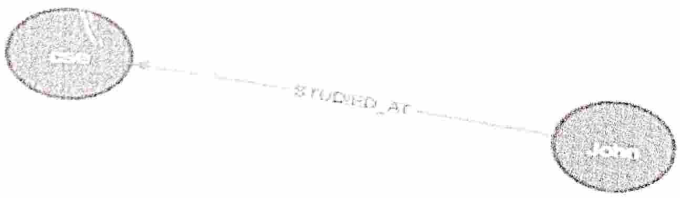



MATCH(s:student),(d:dept) WHERE s.Sname ='John' AND d.deptname='cse'

CREATE(s)-[st:STUDIED_AT]->(d)

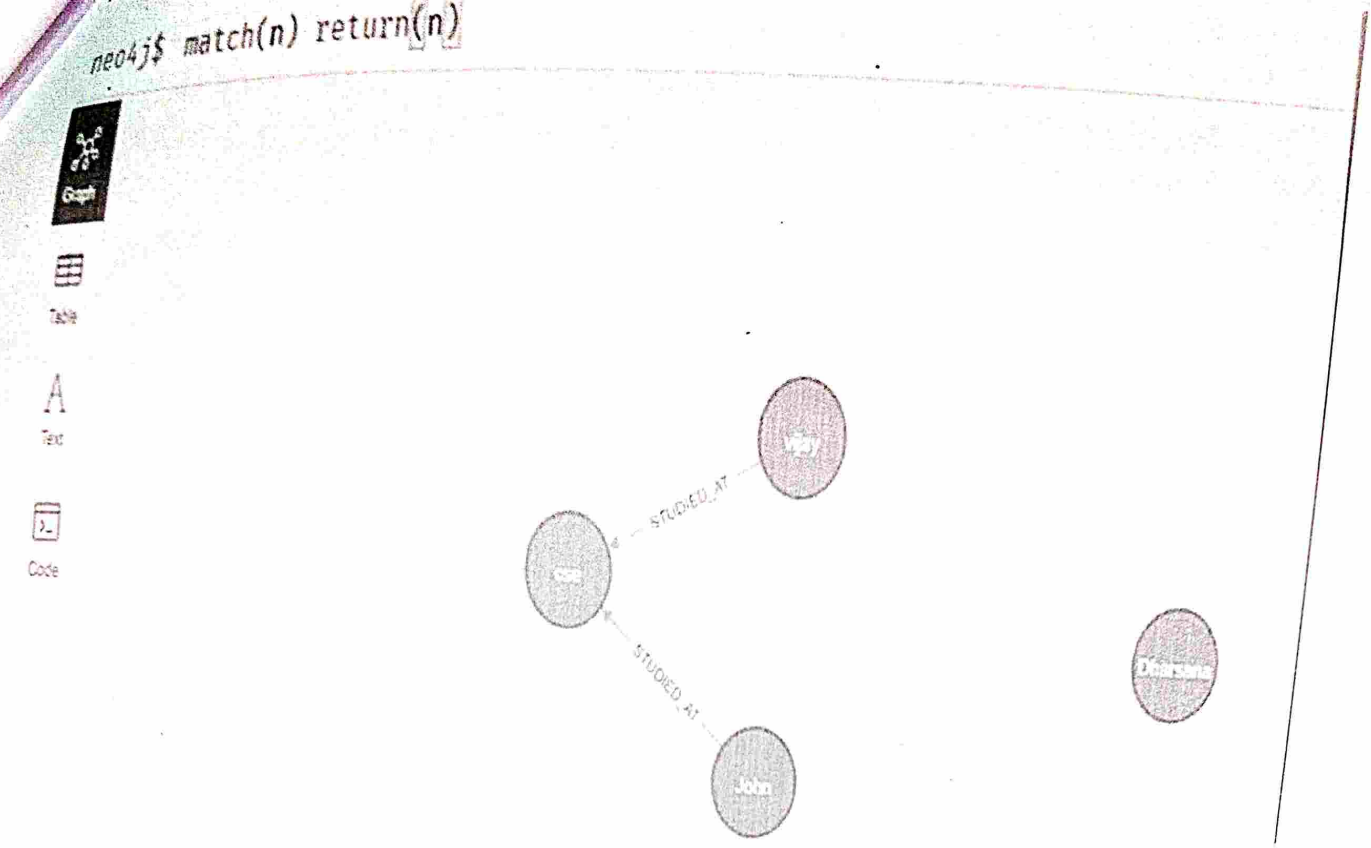
return s,d

---



match(n) return(n)

```
neo4j$ match(n) return(n)
```




**b) Delete a node from student**

match(n:student{Sname:'Dharsana'}) Delete(n)

```
neo4j$ match(n)return(n)
```


| VEL TECH | |
|---|---|
| EX NO. | 11 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYS'S (5) | 5 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | 5 |
| TOTAL (20) | 20 |
| SIGN WITH DATE | |

Result:

The implementation of CRUD operation like creating Inserting, finding and removing operations using graph database is successfully executed

# DATABASE MANAGEMENT SYSTEMS
# (10211DS207)

## TASK:12

## IRTC Bus booking system

## Team details :

**Team leader:** G. Jagan Mohan Reddy

**Reg No:24UEDC0024**

## Team members:

| Names: | VTU NO: | REG NO: |
|---|---|---|
| p.Abhinay | vtu28956 | 24UEDC0050 |
| B. kanaka babu | vtu29144 | 24UEDC0011 |
| k. jagadeeswara Reddy | vtu29544 | 24UEDC0030 |
| T.Jahnavi | vtu29550 | 24UEDC0063 |

# DATABASE MANAGEMENT SYSTEMS
# (10211DS207)

## TASK:12

## IRTC Bus booking system


Team details :

Team leader: G. Jagan Mohan Reddy

Reg No: 24UEDC0024

Team members:

| Names: | VTU NO: | REG NO: |
|---|---|---|
| p. Abhinay | vtu28956 | 24UEDC0050 |
| B. kanaka babu | vtu29144 | 24UEDC0011 |
| k. jagadeeswara Reddy | vtu29544 | 24UEDC0030 |
| T Jahnavi | vtu29550 | 24UEDC0063 |

# AIM:

To develop a microproject on IRTC Bus booking system.

# 1.ER Diagram

An Entity -Relationship (ER) model for a IRTC Bus booking system would involve identifying and defining entities, their attributes, and the relationships between them. Here is a simplified example:

Entities:

1.Passengers

-attributes: passenger_id(pk), nameI, Email, phone_no, Gender.

2.booking:

-attributes: booking date, seatnumber, fare, booking_id, payment_status, seat_no.

3.bus

-attributes: bus_id(pk), bus number, source, destination, departuretime

4.route:

-attributes: routeid(pk), licence_no, contact_no, experience.

5. drive

-attributes: name, driver_id(pk).

6.driver:

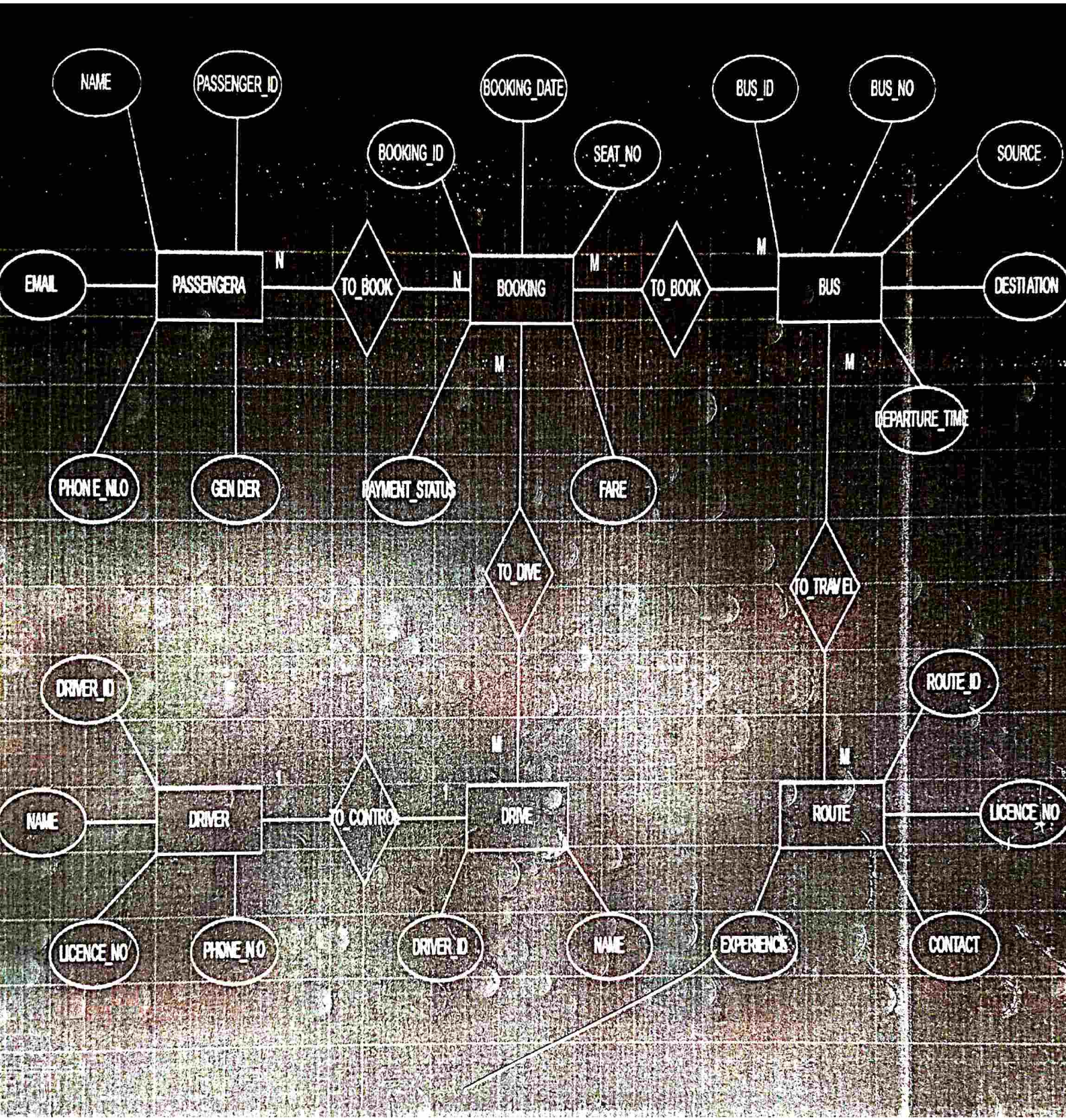
-attributes: contact_no, licence_no(pk), name, driver_id

# Relationships:

## 2. Relationships:

| | |
|---|---|
| PASSENGER-BOOKING | N-TO-N |
| BOOKING-BUS | N-TO-M |
| BOOKING-DTOVE | M-TO-M |
| BUS-ROUTE | M-TO-M |
| DRIVE-DRIVER | 1-TO-1 |

- Each box represents an entity with its attributes inside.
- PK stands for Primary Key,which uniquely identifies each record in the table.
- FK stands for Foreign Key,which establishes a relationship between tables.
- The line connecting the entities represent the relationships between them.
- The notation "1" and "N" indicates the cardinality of the relationships.


NAME
PASSENGER_ID
BOOKING_DATE
BUS_ID
BUS_NO
BOOKING_ID
SEAT_NO
SOURCE
EMAIL
PASSENGERA
N
TO_BOOK
N
BOOKING
M
TO_BOOK
M
BUS
DESTIATION
M
M
DEPARTURE_TIME
PHONE_NLO
GEN DER
PAYMENT_STATUS
FARE
TO_DIVE
TO_TRAVEL
DRIVER_ID
ROUTE_ID
M
M
NAME
DRIVER
TO_CONTROL
DRIVE
ROUTE
LICENCE_NO
LICENCE_NO
PHONE_NO
DRIVER_ID
NAME
EXPERIENCE
CONTACT

# 2.SQL Queries &relational operations :

## 1.select operation

```
SQL> select* from booking;
```

Output:

```
PASSENGER_ID NAME          AGE   PHONE_NO
------------ ------- ---------- ---------
           1 babu            19        99
           2 abhi            20        98
           3 jagades         18        97
           3 jagan           21        97
           3 jahnavi         21        96
```

## 2.Project operation:

```
SQL> select name from booking;
```

Output:

```
PASSENGER_ID NAME          AGE   PHONE_NO
------------ ------- ---------- ---------
           1 babu            19        99
           2 abhi            20        98
           3 jagades         18        97
           3 jagan           21        97
           3 jahnavi         21        96
```

## 3.UNION ALL:

```
SQL> select name from booking union all select startpoint from bus;
```

Output:

```
NAME
---------
babu
abhi
jagades
jagan
jahnavi
atp
bangolore
vijaywada

8 rows selected.
```

**4.intersect:**

```
SQL> select name from booking intersect select startpoint from bus;
```

**output:**

```
no rows selected
```

5.Sum :

```
SQL> select sum(age) as totalage from booking;
```

OUTPUT:

```
  TOTALAGE
----------
        99
```

6.Count :

```
SQL> select count(*) as passengercount from booking;
```

OUTPUT:

```
PASSENGERCOUNT
--------------
             5
```

7.AVG :

```
SQL> select avg(age) as averageage from booking;
```

OUTPUT:

```
AVERAGEAGE
----------
      19.8
```

8.Minimum :

```
SQL> select min(age) as minimumage from booking;
```

Output:

```
MINIMUMAGE
----------
        18
```

9.Maximum :

```
SQL> select max(age) as maximumage from booking;
```

Output:

```
MAXIMUMAGE
----------
        21
```

10.Nested Queries :

a.innerjoin

```
SQL> select booking.name, bus.startpoint from booking inner join bus on booking.passenger_id=bus.bus_id;
```

Output:

```
NAME     STARTPOIN
-------- ---------
babu     atp
abhi     vijaywada
abhi     bangolore
```

11. left outer join:

```
SQL> select booking.name, bus.startpoint from booking left outer join bus on booking.passenger_id=bus.bus_id;
```

Output:'

```
NAME     STARTPOIN
-------- ---------
babu     atp
abhi     bangolore
abhi     vijaywada
jahnavi
jagan
jagades
```

12.right outer join:

```
SQL> select booking.name, bus.startpoint from booking right outer join bus on booking.passenger_id=bus.bus_id;

NAME     STARTPOIN
-------- ---------
babu     atp
abhi     vijaywada
abhi     bangolore
```

# 13.full outerjoin:

```
SQL> select booking.name, bus.startpoint from booking full outer join bus on booking.passenger_id=bus.bus_id;
```

Output:

```
NAME       STARTPOIN
---------- ---------
babu       atp
abhi       vijaywada
abhi       bangolore
jagades
jagan
jahnavi

6 rows selected.
```

# 4.Normalization :

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity.

We normally go through stages called Normal Forms (1NF → 2NF → 3NF → BCNF).

For our Dairy Farming System, we can start from a sample normalized relation and use the Griffith Normalization Tool to automate the process.

1. First Normal Form (1NF) A relation (table) is in 1NF if all its attributes contain atomic (indivisible) values, and each row is unique. Key point: No repeating groups or arrays.

2. Second Normal Form (2NF) A relation is in 2NF if it is in 1NF and every non-prime attribute is fully functionally dependent on the whole primary key.

Key point: No partial dependency on a part of a composite key.

3. Boyce-Codd Normal Form (BCNF) A relation is in BCNF if it is in 2NF and every determinant is a superkey.

Key point: Even stricter than 2NF; eliminates anomalies caused by non-key attributes determining other attributes.

In this databse we perform normalization using Griffith university normalization tool.

## First Normal Form :

### Attributes in Table

Separate attributes using a comma ( , )

passenger_id,name,email,phone_no,gender

### Functional Dependencies

passenger_id ×  →  name × email × gender × phone_no ×  Delete

Add Another Dependency

# Check Normal Form

## 2NF

The table is in 2NF

## 3NF

The table is in 3NF

## BCNF

The table is in BCNF

# Show Steps

2NF

```
find all candidate keys. The candiates keys are { passenger_id}, The set of key attributes are: { passenger_id }
for each non-trivial FD, check whether the LHS is a proper subset of some candidate key or the RHS are not
all key attributes
checking FD: passenger_id --> name,email,gender,phone_no
```

3NF

```
find all cadnidate keys. The candiates keys are { passenger_id}, The set of key attributes are: { passenger_id }
for each FD, check whether the LHS is superkey or the RHS are all key attributes
checking functional dependency passenger_id --> name,email,gender,phone_no
```

BCNF

```
A table is in BCNF if and only if for every non-trivial FD, the LHS is a superkey.
```

## Normalize to 2NF :

# Normalize to 2NF

Attributes

passenger_id | name | email | phone_no | gender

Functional Dependencies

passenger_id → name email gender phone_no

## Show Steps

First, find the minimal cover of the FDs, which includes the FDs
passenger_id -> name
passenger_id -> email
passenger_id -> gender
passenger_id -> phone_no

Initially rel[1] is the original table:

Round1: checking table rel[1]

***** The table is in 2NF already, send it to output *****

## Normalize to 3NF :

1NF to 3NF

Attributes

passenger_id | name | email | phone_no | gender

Functional Dependencies

passenger_id → name
passenger_id → email
passenger_id → gender
passenger_id → phone_no

Show Steps

Table already in 3NF

## Normalize to BCNF :

Normalize to BCNF

Attributes

passenger_id | name | email | phone_no | gender

Functional Dependencies

passenger_id → name email gender \ phone_no

Show Steps

Table already in BCNF, return itself.

# 5.Document database using MONGODB :

CRUD,Which stands for Create,Read,Update,and Delete,represents a set of fundamental operations used to insert with and manipulate data stored in a database.These operations serve as the building blocks upon which countless applications,from simple to highly complex,rely.

## CREATE:

db.createCollection("irtc")

Output:

{ "ok" : 1 }

## INSERT ONE:

db.irtc.insertOne({item:"name",qty:24,tags:["character"],size:{h:28,w:35.5,uom:"cm"}})

**Output:**

{

"acknowledged" : true,

"insertedId" : ObjectId("68f7a50c328226b6c747cc88")

}

## FIND():

db.irtc.find({item:"name"})

**Output:**

{ "_id" : ObjectId("68f7a5d0e9c83b3f4509eced"), "item" : "name", "qty" : 24, "tags" : [ "character" ], "size" : { "h" : 28, "w" : 35.5, "uom" : "cm" } }

## INSERT MANY:

db.irtc.insertMany([{item:"gender",qty:6,tags:["black","red"],size:{h:14,w:21,uom:"cm"}},{item:"phone_no",qty:2,tags:["stable"],size:{h:19,w:22,uom:"cm"}},{item:"passenger_id",qty:25,tags:["solid","siver"],size:{h:20,w:23,uom:"cm"}}])

**Output:**

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("68f7a6ea2e66dff570e9e109"),

ObjectId("68f7a6ea2e66dff570e9e10a"),

ObjectId("68f7a6ea2e66dff570e9e10b")

]

}

## FIND() with Projection:

db.irtc.find({},{item:1,qty:1}).pretty()

Output:

{ "_id" : ObjectId("68f7a79b3aa6d2e1fb712d70"), "item" : "name", "qty" : 24 }

{ "_id" : ObjectId("68f7a79b3aa6d2e1fb712d71"), "item" : "gender", "qty" : 6 }

{ "_id" : ObjectId("68f7a79b3aa6d2e1fb712d72"), "item" : "phone_no", "qty" : 2 }

{ "id" : ObjectId("68f7a79b3aa6d2e1fb712d73"), "item" : "passenger_id", "qty" : 25 }

db.irtc.find({},{item:1,qty:1}).pretty()

**Output:**

```
{
        "_id" : ObjectId("68f7a8a8f7331254e4f3157d"),
        "item" : "gender",
        "qty" : 6
}
{
        "_id" : ObjectId("68f7a8a8f7331254e4f3157e"),
        "item" : "phone_no",
        "qty" : 2
}
{
        "_id" : ObjectId("68f7a8a8f7331254e4f3157f"),
        "item" : "passenger_id",
        "qty" : 25
}
```

## FIND() with Sort:

db.irtc.find({item:"name"}).pretty().sort({item:-1})

**Output:**

{

```
        "_id" : ObjectId("68f7a93e2f547bb56c7fa290"),
        "item" : "name",
        "qty" : 24,
        "tags" : [
            "character"
        ],
        "size" : {
            "h" : 28,
            "w" : 35.5,
            "uom" : "cm"
        }
}
```


## DELETE:

```
db.irtc.deleteOne({item:"name"})
```

**Output:**


```
{ "acknowledged" : true, "deletedCount" : 1 }
```


## FIND() after DELETE:

```
db.irtc.find({},{item:1,qty:1}).pretty()
```

**output:**


```
{
        "_id" : ObjectId("68f7aa317353abb7f17ea765"),
        "item" : "gender",
        "qty" : 6
}
{
        "_id" : ObjectId("68f7aa317353abb7f17ea766"),
        "item" : "phone_no",
        "qty" : 2
}
{
```

```
    "_id" : ObjectId("68f7aa317353abb7f17ea767"),
    "item" : "passenger_id",
    "qty" : 25
}
```

# 6.Graph Database using MONGODB(using neo4j online compiler)

## a)Create a graph database

create(p1:Passenger{id:"p001",name:"Jagan"})

**Output:**

Created 1 node, set 2 properties, added 1 label

**CREATE (n1:passengername {id: "n001", paymentstatus: "cash", gender: "Female", age:22})**

**Output:**

**Created 1 node, set 4 properties, added 1 label**

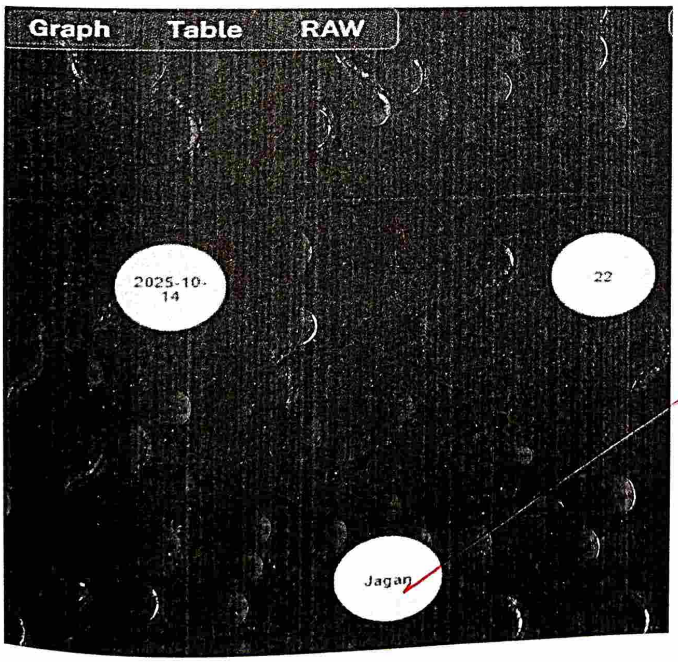
**CREATE (b1:bookingstatus {id: "b001", seat_no: 15, date: "2025-10-14"})**

**Output:**

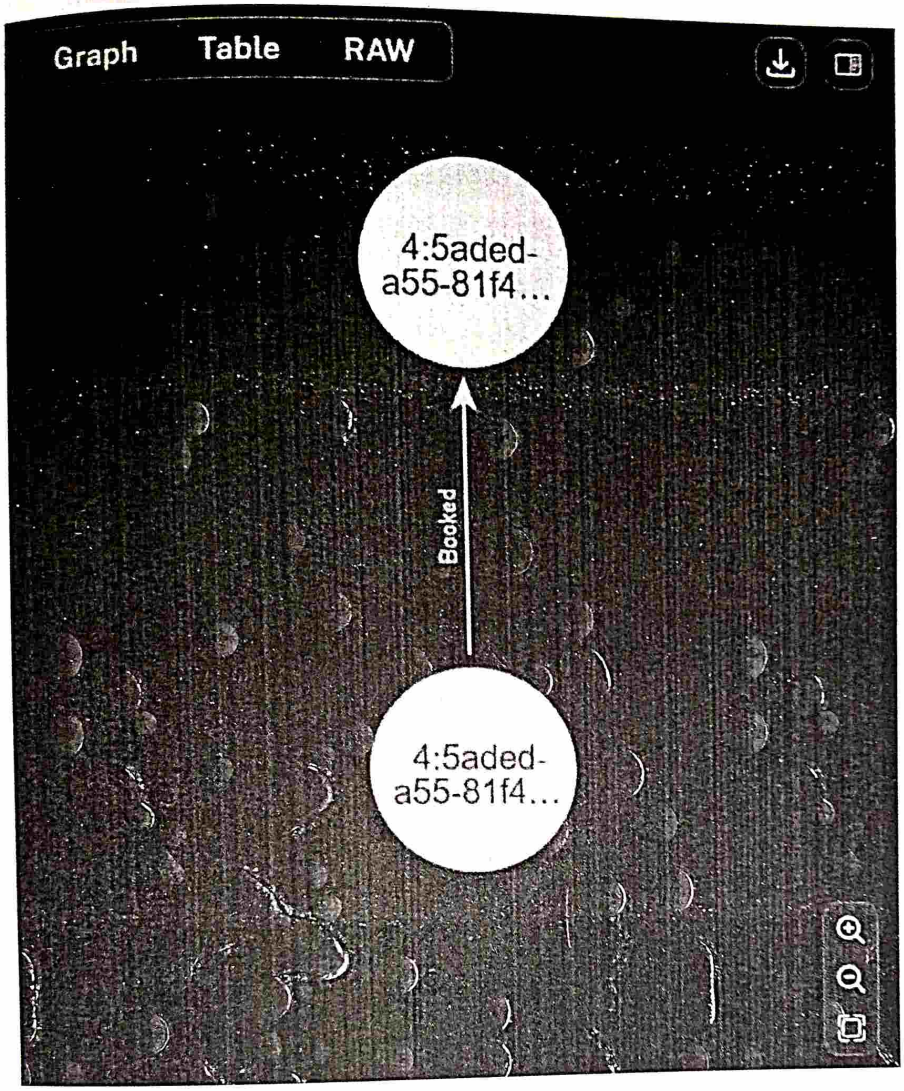
**Created 1 node, set 3 properties, added 1 label**

**match(n) return(n)**

**Output:**



create (p)-[:Booked]->(b)




match (p)-[r:Booked]->(b) DELETE r

Deleted 1 relationship

| VEL TECH | |
|---|---|
| EX NO. | 12 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYSIS (5) | 5 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | 20 |
| TOTAL (20) | |
| SIGN WITH DATE | 23/10/25 |

## Result:

Thus, the micro project for IRTC Bus booking system was developed and implemented successfully completed.