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: B. Vasu

VTU NO: 30458

REG.NO: 24UEDC0008

BRANCH: CSE (Data Science)

YEAR/SEM: 2nd / III

SLOT: $S_{15} + L_{12}$

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**

1. What is Data ?

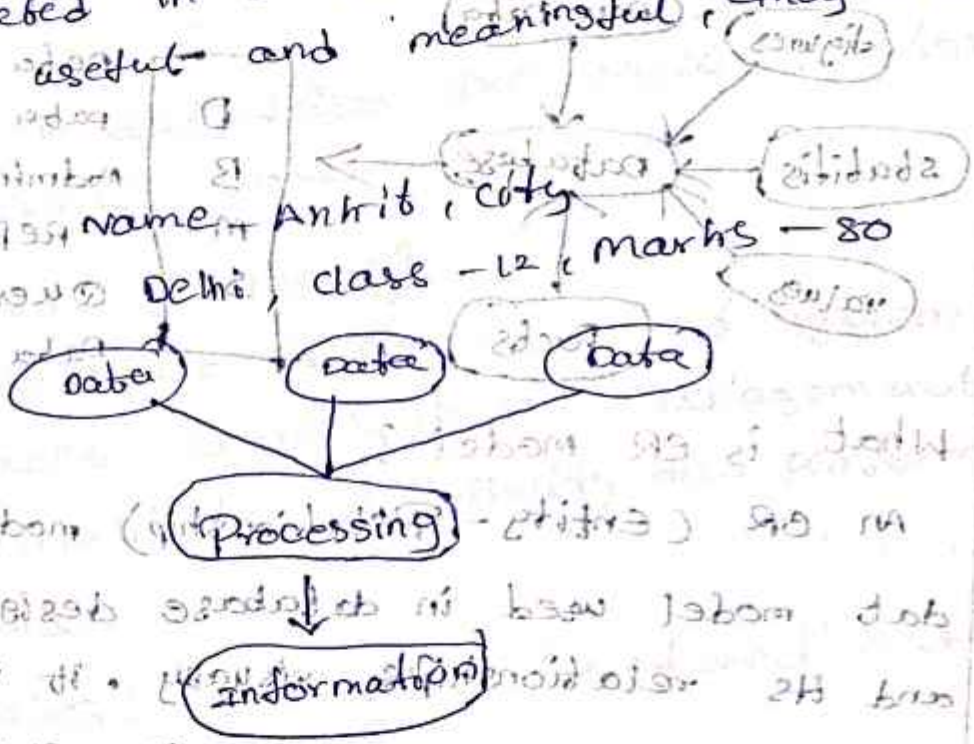
Data is a collection of a distinct small unit of information. it can be used in a variety of forms like text, number, media, bytes, etc, it can be stored in pieces of paper or electronic memory, etc

ex!- Anrit, Delhi

2. What is Information?

when data are Processed, organized, structured and interpreted in a given context so as to make them useful and meaningful, they are called information

ex! name - Anrit, city - Delhi, class - 12, marks - 80



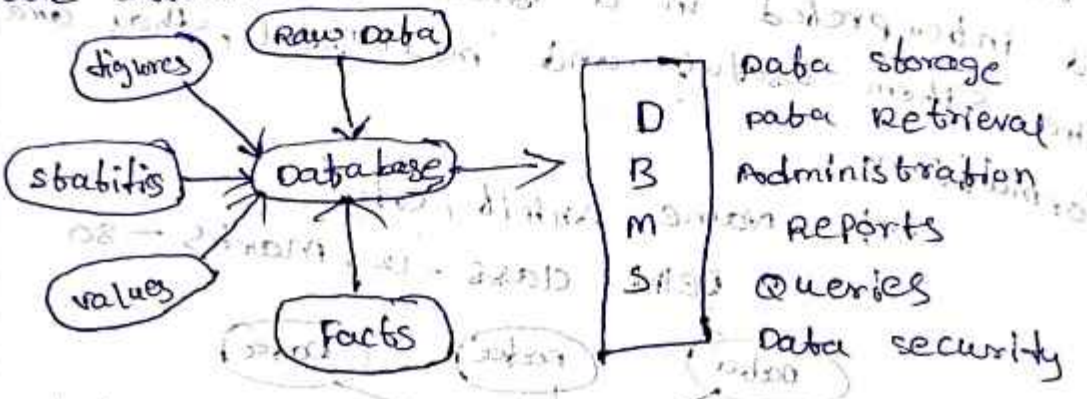
3. What is Database?

The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently. it is also used to organize the data in the form of a table, schema, views and reports etc.

ex! The college database organize the data about the admin, staff, student and faculty etc.

4. What is DBMS?

DBMS stands for database management system which is software for creating managing and manipulating database. It ensures data is stored securely retrieved efficiently and maintained consistently. key features include data storage, integrity, security, backup, recovery and support for multiple users, Ex:- mySQL, MS SQL server, oracle, SQL, DB2 etc are different types of database management system



5. What is ER model?

An ER (Entity-Relationship) model is a conceptual dat model used in database design to represent data and its relationships visually. it includes.

- Entities: objects or things (eg. customer, order)
- Attributes: properties of entities (e.g. customer ID, Name).
- Relationships: connections between entities (e.g. Customer places order).

ER diagrams use rectangles for entities, ovals for attributes and diamonds for relationship, this model helps in organizing and structuring data for databases.

6. What is SQL?

SQL (structured Query Language) is a programming language used to manage and manipulate relational databases. Key function include.

- Data Querying: Retrieve data (SELECT).
- Data manipulation: Add, update, and delete data (INSERT, UPDATE, DELETE).
- Data definition: create and modify database structures (Create, Alter, Drop).
- Data control: Manage access to data (Grant, Revoke).

SQL is used in databases like mySQL, PostgreSQL Oracle, and SQL server.

7. What is modern database?

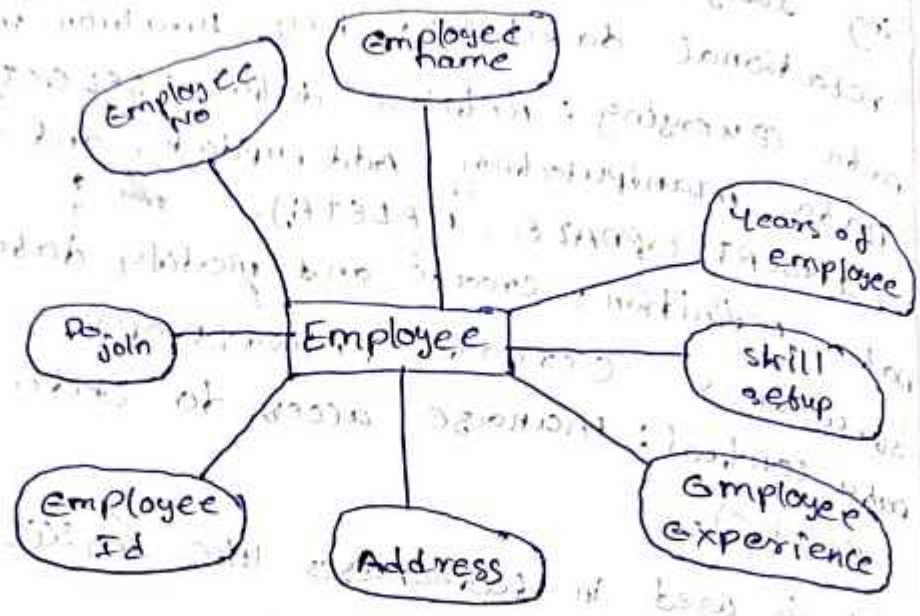
A modern database is an advanced system designed to meet current data management needs offering scalability, flexibility and performance

Key types include:

1. NoSQL Databases: handle unstructured data eg., mongoDB, Redis.
2. NewSQL databases: combine NoSQL scalability with SQL's ACID properties eg:. google spanner.
3. Distributed databases: spread data across multiple node eg.. Apache cassandra.
4. Cloud databases: hosted on cloud platforms e.g.. Amazon RDS.
5. In-memory databases: store data in RAM for fast access eg Arrang eg. Redis
6. Multi-model databases: support multiple data models e.g. ArangoDB.

Conceptual Design after ER

Identifying the attributes



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

Aim:

To draw conceptual design through ETR using drawio tool.

Procedure:

1a Identifying the entities for bank management system.

Sample output

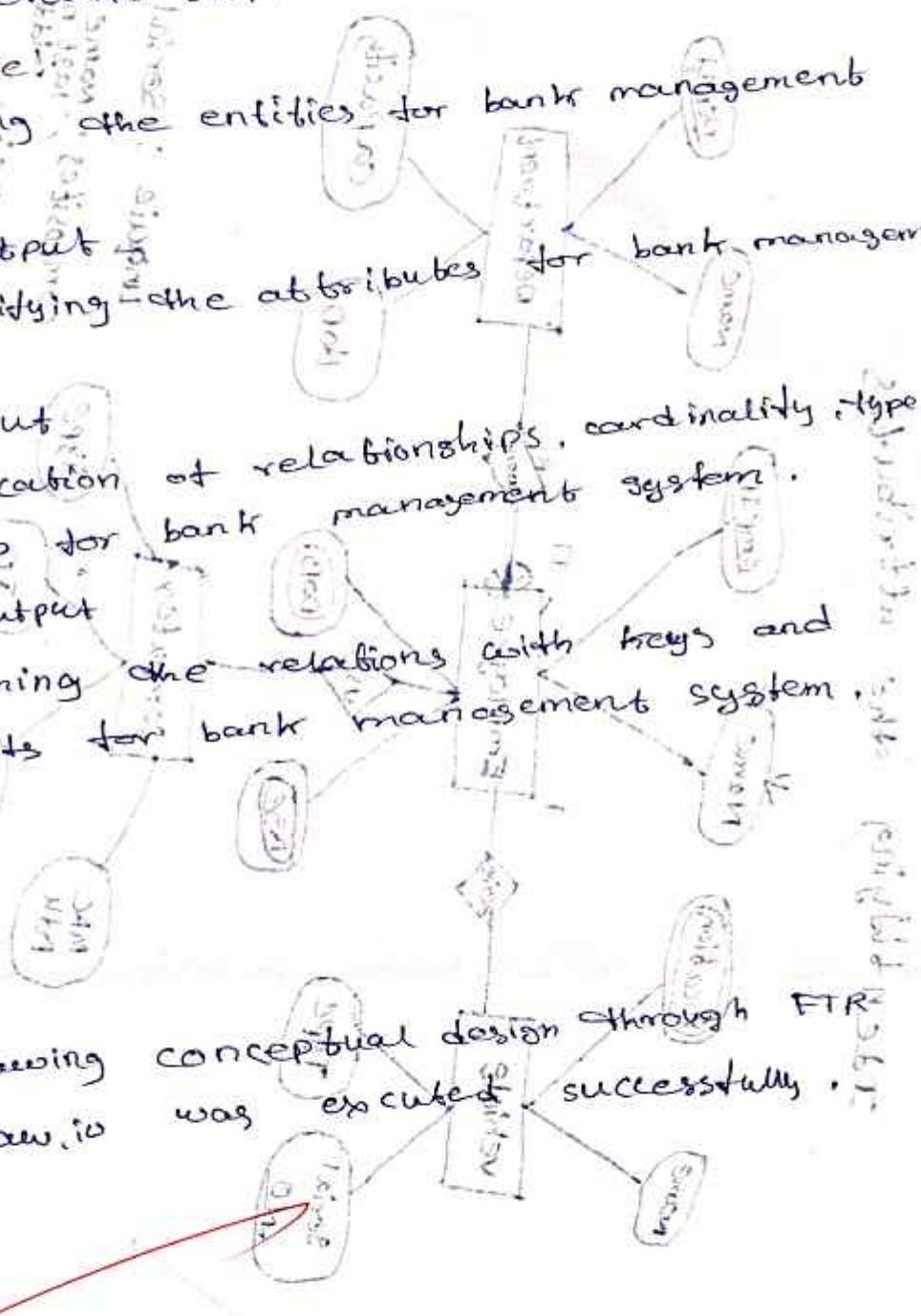
2. Identifying the attributes for bank management system.

sample output

c. Identification of relationships, cardinality, type of relationship for bank management system.

sample output

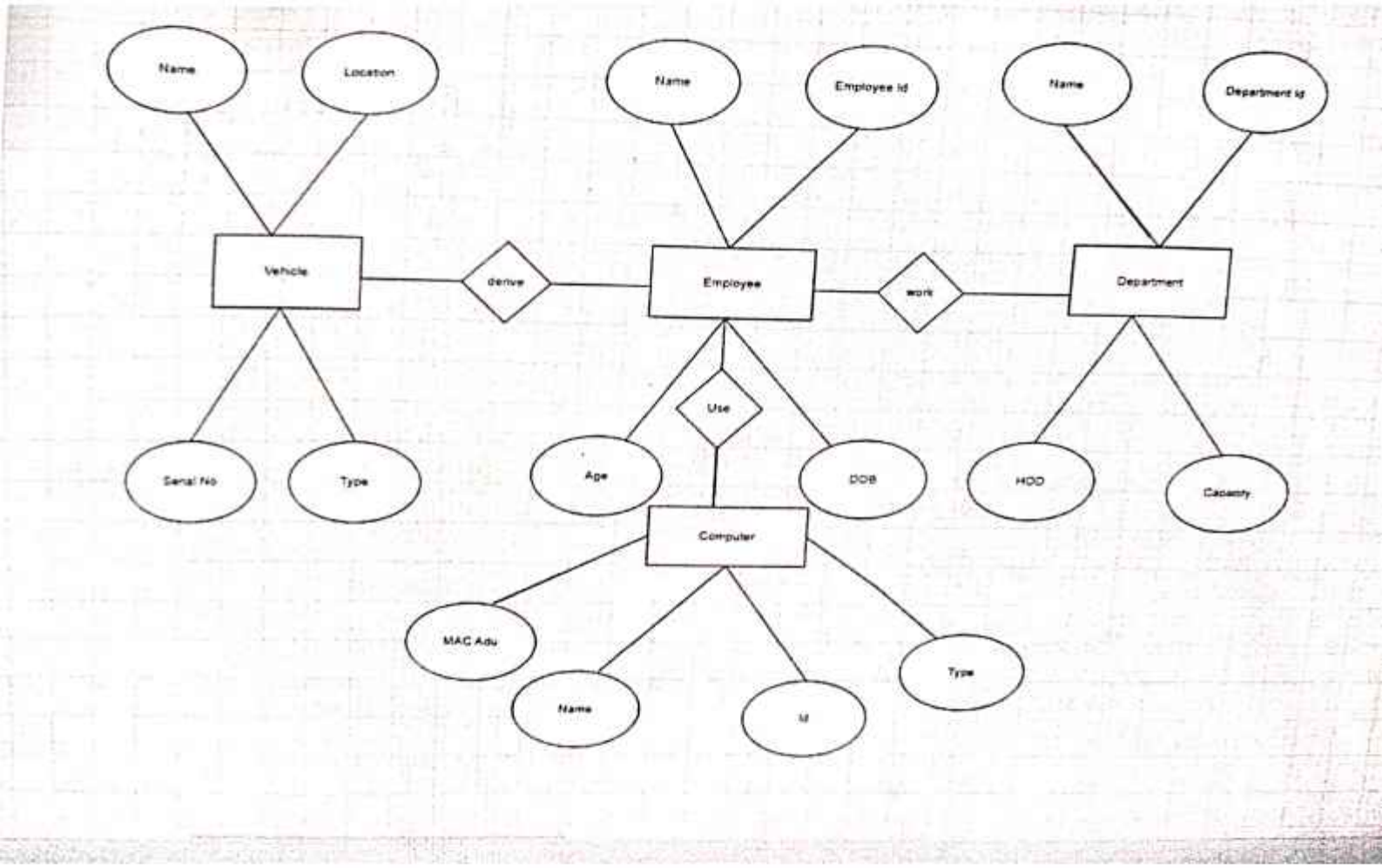
d. Reframing the relations with keys and constraints for bank management system.



Result:

Thus drawing conceptual design through ETR using draw.io was executed successfully.

Print


Name
Location
Vehicle
Serial No
Type
derive
Name
Employee Id
Employee
Use
Age
DOB
Computer
MAC Adu
Name
Id
Type
work
Name
Department Id
Department
HOD
Capacity

Procedure:

Sample output

1a: vehicle, employee, Department, computer

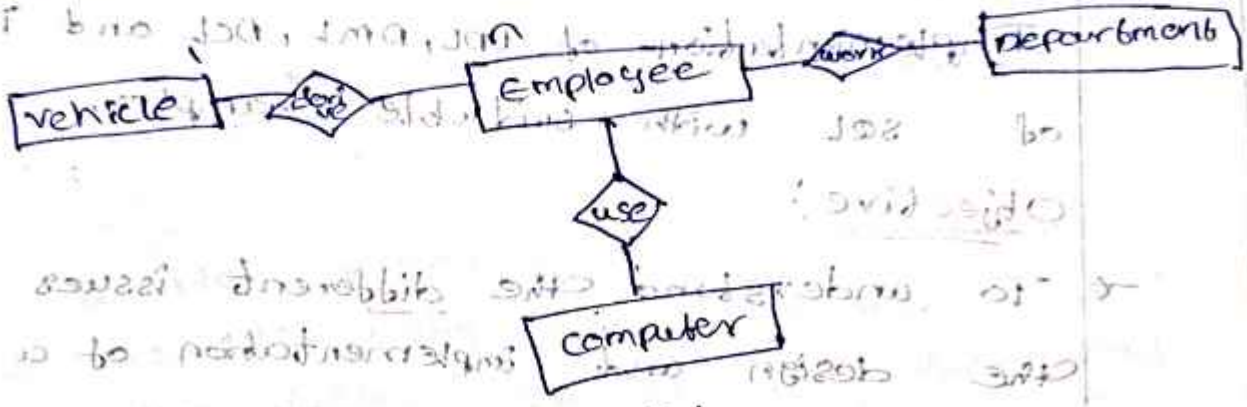
1b. vehicle → name, location, serial NO

employee → Name, emp ID, age, DOB, Type

Department → Name, Dep-Id, HOD, capacity

computer → MAC Add, Name, Id, Type.

1c.



Id: unique :- serial, emp Id

composites:- name, last name

multiple value:- DOB, age, location

Relationship:- work, derive, use

Result:

Thus drawing conceptual design through ETR using draw.io, was executed successfully.

TECH	1
	5
MANCE (5)	5
LT AND ANALYSIS (5)	5
VOICE (5)	5
RECORD (5)	25
TOTAL (20)	25
IGN WITH DATE	25/8/22

7/8/25

Task-2

Generating design of other traditional database model.

Implementation of DDL, DML, DCL and TCL commands of SQL

Aim:

Implementation of DDL, DML, DCL and TCL commands of SQL with suitable examples.

Objective:

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

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 distinct parts. These are.

Interactive SQL:

Interactive SQL is designed for create, access and manipulate data structure like tables and indexes.

PL/SQL:

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

Oracle forms:

This total allows you to create a data only screen along with the suitable menu objects. thus it is the oracle forms tool that handles data gathering and data validation in a commercial applications.

Report writter:

Report writter allows programmers to prepare innovative reports using data from the oracle structures like, tables view, etc. it is the repeat writer tool that handles the reporting section of commercial application.

Oracle graphics:

some of the data can be better repeated in the form of pictures the oracle graphices tools allows programmers to prepare graphs using data from, oracle structure like, tables view etc.

SQL (structured Query language):

structured query language, is a database computer language designed for managing data in relational database management systems (RDMS) 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 language for Edgar F. Codd's relational model and became most widely used language for relational database

IBM developed SQL in mid of 1970's

oracle incorporated in the year 1979 SQL by IBM/DB2 and DS database systems.

SQL adopted as standard language for RDBs ANSI in 1989.

Data types:

1. CHAR(size):
This data type is used to store character string values of fixed length. the size in bracelets determines the number of character the cell can hold. the maximum number of character is 255 character

2. VARCHAR(size)/VARCHAR2(size):
This data type is used to store variable length alphanumeric data. the maximum character can be is 2000 character.

3. Number (P,S):
the data type is used to store variable length alphanumeric data. the maximum character can be used to store number of virtual any magnitude may be stored up to 38 digits of precision. number as larger as $9.99 * 10^{124}$. this precision (P) determ the number of places to the right of the decimal. If scale is omitted then the default is zero.

4. Date:

This data type is used to represent data, and time the standard formed is, DD-MM-YY as in 17-SEP-2009-TO enter date other than the standard format we the.

appropriate function date time starts date in the 24-hours format. By default the time In a data field is 12:00:00 am if no time partic is specified the default date by for date field is this first day the current month.

5. LONG:

This data type is used to store variable length character strings containing up to 2GB. long data can be used to store arrays of binary data can be used to arrays of binary dates in ASCII format long values cannot be indexed, and the normal character function such as SUBSTR cannot be appli

6. RAW:

The raw data type is used to store binary date such as digitized picture or image data, as loade into columns of these data types are stored with out any further conversation. RAW data types can have a maximum length of 255 bytes.

DATA DEFINITION LANGUAGES (For shopping management system):

A Data definition language (DDL) statement are used to define the database more structure or schema.

Create:

To make a new database, table, index, or stored query. a create statement in SQL created an object inside of a relational database management system (RDBMS)

Syntax:

```
CREATE TABLE table_name
(column_name1 datatype (csize)
 column_name_n datatypes (csize)
  column_name_n data_type (csize));
```

Alter:

To modify on existing data base object Alter the structure of the data base

To add a column in a table.

syntax:

```
ALTER TABLE table_name ADD
  column_name datatype (csize)
```

alter table shopping customer: add Review varchar [20];

To dabe column in a table

syntax:

```
ALTER TABLE table_name DROP_column
  column_name;
```

* To modify a column in a table

Syntax:

ALTER TABLE table_name modify
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

TRUNCATE TABLE:

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

syntax

TRONCATE TABLE table_name

DATA MANIPULATION LANGUAGES:

Data manipulation language allows the users to query and manipulation data in existing schema in object. It object. following data to schema object.

Insert:

value can be inserted into table using insert commands. there are two types of insert command they are multiple value insert commands, single value insert commands.

Syntax!

INSERT INTO table-name values (value 1, value 2, value 3, ...);

(OR)

INSERT INTO table-name (column 1, column 2, column 3, ...) (value 1, value 2, ... value 3)

update!

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

syntax!

update <table-name> SET WHERE

<col 2 = value>

<column = value>;

Delete!

This allows you to delete the particular column value using 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 data base. the select statement can be used in many ways. they are.

select some column!

To select specified number of column 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 tables.

syntax:

select distinct column name(s) from table-name;

3. select using between:

Between can be used to get these items that fall within arange.

syntax:

select column name from table-name WHERE column name BETWEEN value 1 AND value 2;

4. Renaming:

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

syntax:

Renaming column;

SELECT column name as new name from table-name;

DATA CONTROL LANGUAGES;

1. create;

* create user kamal identified by kamal;

user created.

2. Grant;

* grant all privillges to kamal;

grant succeded.

3. Revoke;

Revoke all privileges from kamal;

Revoke succeded

TRANSACTION CONTROL LANGUAGE;

1. commit;

* commit;

commit complete

2. Save point;

* save point A1;

Save point created

3. Roll back;

* Roll back to A1;

Rollback complete.

Output:

SQL*Plus: Release 11.2.0.2.0 Production on
Thu Aug 7 11:56:57 2025

Copyright (c) 1982, 2011, Oracle. All rights reserved.

SQL> connect
Enter user-name: system
Enter password:
Connected.

SQL> create table student (full name varchar(18),
idno number(5), address varchar(20));

Table created

SQL> create faculty (name char(10), ttsid number(5),
mobileno number(10));

create faculty (name char(10), ttsid number(5),
mobile no number(10))

SQL> create table student (name varchar(18),
idno number(5), address varchar(20));

create table student (name varchar(18), idno
number(5), address varchar(20))

SQL> create table student (name varchar(18),
id no number(5), address varchar(20));

Table created.

SQL > Alter table faculty Add gmail varchar (30);

Alter table faculty Add gmail varchar (30);

SQL > Alter table faculty Add gmail varchar(30);

Table altered.

SQL > Alter table faculty drop column ttsid;

Alter table faculty drop column ttsid.

SQL > Alter table faculty drop column TTSID;

Alter table faculty drop column TTSID

SQL > Alter table faculty drop column mobileno;

Table altered.

SQL > DESC faculty;

Name	Null?	Type
NAME		VARCHAR2(15)
IDNO		NUMBER(5)
AGE		NUMBER(2)
DEPARTMENT		CHAR(3)
GMAIL		VARCHAR2(30)

SQL > Alter table faculty modify idno number(10);

Alter table faculty modify idno number(12)

SQL > Alter table faculty modify idno number(2);

Table altered.

SQL > DESC faculty

Name	Null ?	Type
NTU RO		VARCHAR2(18)
AGE		NUMBER(7)
DEPARTMENT		NUMBER(2)
GMAIL		CHAR(3)
SQL>		NARCHAR2(30)

VEL TECH	
EX NO.	2
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	
TOTAL (20)	14
SIGN WITH DATE	14/8/25

Result:

Thus Implementation of generating design of other traditional database model.

Thus program executed successfully.

28/8/25

Task: 3

Developing Queries with DML single-row function and operations.

Aim:

To perform the query processing on databases for different retrieval result of queries using DML, DRL single-row operations using aggregate, date, string, indent functions, set clauses and operators.

Procedure:

create table for employee schema and insert around 10 retail_store_retail_store_employees data in this relation perform multi row functions.

Note: These queries assume a sample database with an "retail_store_employees" table containing columns like "retailstore_employee_id", "storeid" and departmenit", "store_phonenumber "employee_phonenumber";

Aggregative operators:

In addition to simply retriving data, we often want to perform some computation or summarizati 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 keyword is used then it will return only

the count of unique tuple in the column. other wise it will return count of all the tuples (including duplicates) count(*) indicates all the tuples of the column

syntax:

count (column name)

example:

select count(*) from retail_store_employees;

2. SUM:

sum followed by a column name returns the sum of all the values in that column.

syntax: sum (column name)

example: select sum (salary) from retail_store_employees;

3. AVG:

AVG followed by a column name returns the avarage value of that column values.

synfax: AVG (n1, n2)

Example: select AVG (salary) from retail_store_employees.

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

MAX (colu

Syntax: MAX (column name)

Example; select retailstore_

select MAX (salary) from retail_store_employees;

SQL-> select retailstore_employee_name, max(salary) from retail_store_employees group by retail_store_employee_name;

DEPTNO MAX (salary)

select retailstore_employee_name, max (salary) from retail_store_employees group by retail store_ employee_name having max (salary) < 3000;

5. MIN:

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

Syntax: MIN (column name)

Example: select MIN (salary) from emp;

select retail store_employee_name min (salary) from retail_store_employees group by retail_store employee_name having min (salary) > 1000;

String functions:

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

1. UPPER()

Query:

```
SELECT
UPPER (retailstore_employee_name)
FROM
retail_store_employees
WHERE
retail store_employee_id=1;
```

2. LOWER()

Query:

```
SELECT
LOWER (retail store_employee_name)
FROM
retail_store_employees
WHERE
retail store_employee_id =1;
```

3. Length()

Query:

```
SELECT
LENGTH (retailstore_employee_name)
FROM
    retail store_employees
WHERE
    retail store_employee_id = 1;
```

4. SUBSTR()

Query:

```
SELECT
SUBSTR (retailstore_employee_name, 1, 5)
FROM
    retail_store_employees
WHERE
    retail store_employee_id = 1;
```

5. CONCAT()

Query:

```
SELECT
CONCAT (retail store_employee_name, deperatment)
FROM
    retail_store_employees
WHERE
    retail store_employee_id = 1;
```

Data and Time functions:

The data & time functions are built-in function in the SQL. These function can be used in SQL queries to perform various date and time operation such as filtering records based on dates, calculating date difference, and formatting dates for display purposes.

For storing a date or a date and time value in a database, My SQL offers the following data 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.

CAK 3

28

CURDATE()

query: select current date from dual;

CURTIME()

query: select current_time() from dual;

Add date (days, date)

sql > select Add date ('2018-8-1 31');

day of month (date)

select day of month ('2018-8-15');

day of week (date)

select day of week ('2018-8-15');

day of year (date)

select day of year ('2018-2-15');

month (date)

select month ('2018-8-01');

TIME (expr)

select time ('2018-8-1 11:33:25');

sysdate:

select sysdate from dual;

next-day:

select

next_day (sysdate, 'wed') from dual;

add_months:

select

Add_months (sysdate, 2) from dual;

last_day:

select last_day (sysdate) from dual;

months_between:

select

months_between (sysdate, hiredate) from emp;

Least:

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

Greatest:

select
greatest ('10-JAN-07', '12-oct-07') from dual;

Trunc:

select trunc (sysdate, 'date', day') from dual;

Round:

select Round (sysdate, 'day') from dual;

to-date:

SQL > select to date (sysdate, "dd|mm|yy") from dual;

Character function:

init cap (char): select initcap("hello") from dual;

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

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

Itrim (char, [set]): select Itrim ('cseit', 'cse')
from dual;

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

replace (char, search); select replace ('sachrand
sue', 's', 'b') from dual;

Cay 3

28-

string functions:

concat:

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

select concat ('ORACLE', 'CORPORATION) from dual;

ORACLECORPORATION

lpad:

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

select LPAD ('ORACLE, 15, '*') from dual;

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

select Rpad ('ORACLE', 15, '*') from dual.

Ltrim:

Return a character expression after removing leading blanks.

select ltrim ('SSMITHSS', 'S') from dual;

lower:

Returns a character expression after converting uppercase character data to lower case.

select lower ('DBMS') from dual;

upper:

Returns a character expression with lower case character data converted to uppercase

```
select upper ('dbms') from dual;
```

length:

Return the number of characters, rather than the number of bytes, of the given string expression excluding trailing blanks.

```
select length ('DATA BASE') from dual;
```

substr:

Returns part of a character, binary, text, or image expression.

```
select
    substr ('ABCDEFGHIJ', 3,4) from dual;

select
    substr ('retailstore_employee_name', 1,4) from
    retail_store, employees;
```

Instr:

The INSTR functions search string for substring the function returns an integer indicating the position of the character in string that is the first character of this occurrence.

```
select instr ('CORPORATE FLOOR', 3, 2)
    From dual;
```

28:

Output:

```
create table retail_store_employees
(retail_store_employee_id number, store_id
retail_store_employee varchar (34), salary
number, department varchar (23), store_ph
no number);
```

Table created.

```
desc retail_store_employees;
```

Name	Null ?	Type
Retail_store_employee_Id		Number
store_Id		Number
Retail_store_employee		varchar(34)
salary		Number
department		varchar(23)
store_phone_no		Number

```
insert into retail_store_employees values
(25780, 12345, 'vasu', 20000, 'ravi', 7416615
```

1 row created.

```
insert into retail_store_employees values
(56787, 66278, 'vishnu', 68000, 'ravi', 90656
```

1 row created.

```
insert int retail_store_employee values
(89809, 56765, 'shiva', 'ravi', '2695
```

1 row created.

salary	department	name	store_phone_no
25780		vasu	7411661594
56787		nishma	9063007191
89809		shiva	7696577710

```
select count(*) from retail_store_employees;

count(*)
--------
       3

select count sum(salary) from retail_store_
employees;

sum(salary)
-----------
     153788

select avg(salary) from retail_store_employees;

Avg(salary)
-----------
   51267.6667

select min(salary) from retail_store_employees;

min(salary)
-----------
   51262.6667
   29000

select max(salary) from retail_store_employees;

max(salary)
-----------
      69000

select upper(retail_store_employeename) from
retail_store_employees, retail_store_employee_id
=2580;

upper(retail_store_employeename)
--------------------------------
VASU
```

select lower (retail_store_employeename) from retail_store_employees where retail_store_employee_id = 56787;

lower (Retail_store_employeename)

5

select sysdate from dual;

sysdate

21-AUG-25

select next_day (sysdate, 'WED') from dual;

Next day

27-AUG-25

select add_months (sysdate, 2) from dual;

add_month

21-OCT-25

select last_day ('sysdate') from dual;

last_day

31-AUG-25

select least (10-feb-07, 12-oct-07) from dual;

least

10-feb-17

select greatest (10-feb-7, 12-oct-10) from dual;

Greatest

12-oct-07

```
select trunc (sysdate, 'day') from dual;

TRUNC (SYS)
-----------
24-Aug-25

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

ROUND (SYS
----------
31-AUG-25

select to_char (sysdate, 'dd/mm/yy') from dual;

To_char(
--------
28/8/25

select to_date (sysdate, 'dd/mm/yy') from dual;

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

select concat ('oracle', 'corporation') from
    dual;
    concat ('ORACLE', '
    ------------------
    oracle corporation

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

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

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

LTRIM(
------
mithss
```

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

RTRIM(
---
ssmith

select lower ('DBMS') from dual;

LOWE
---
dbms

select upper ('DBMS') from dual;

UPPE

DBMS

select length ('DATA BASE') from dual;

length ('DATABASE')

9

Select substr ('dbms', 1, 3) from dual;

SUB
---
dbm

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

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

14

select * from retail_store_employees where salary is NULL;

no rows selected

select * from retail_store_employees where salary is not null;

RETAIL_STORE_EMPLOYEE_ID

	store_id	retail_store_employee
29000 ravi	25780	12345 vasu 7416615941
64000 ravi	56787	66778 vishnu 9063007191
56788 ravi	898809	56765 shiva 7696577710

6 rows selected.

VEL TECH	
EX NO.	5
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	28/6/24

Output:

This, developing queries with DML, single row function and operation is executed.

14/9/23

Task: 4

Developing queries with DML and operators

Perform the advanced query processing (multi row abilities) and test its heuristics using the design of optimal correlated and nested subqueries such as finding summary statistics.

Consider the schema for

EMPLOYEES (emp_no, emp_name, department, dept salary, ase)

orders (emp_no, order id, price, qty_ord, qty_hand

Itemfile (itemid, itemname, qty_ord, qty_hard, itemrate)

Queries using UNION, Intersect, Minus:-

Union: the union operator returns all distinct fu 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 sele emp_no from orders;

output;

Union All;

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

Intersect;

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

output:

Minus;

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

output:

Practice questions;

1. Find the emp_no of employees whose name starts with 'S' and ends with 'H'
2. Find the names of the employees whose age is between 20 and 40
3. display all the name of the employees beginning with 'r'
4. Display the sorted list of employees names.

Queries using Group by, having clause and order clause

Group by;

This query is used to group to all the records in a 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 employees group by dept no;

output:

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 id used.

SQL -> select deptno, count(*) from empolyees group by deptno having deptno is not null;

output:

ORDER BY:

This query is used to display a selected set of fields from a relation in an ordered manner based 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;

output:

SQL> select empno, emp_Name, salary from empolyees order by salary desc;

output:

from

SQL * plus having following operators.

SQL> select salary + comm from emp_master;
ph_num + ~~vasu~~ from vasu;
salary + comm s_id

output;

SQL -> select salary + comm net_sal from emp_master;

output;

SQL -> select 12* (salary + comm) annual_net sal from emp_master;

output;

Subqueries;

SQL -> select * from employees

SQL -> Insert into employees select * from employees where emp_id in (select emp_id from employees);

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

Delete from employees where department in (select department from employees where department = 'sales');

IN;

query; select * from employees where department IN ('sales', 'marketing');

output;

NOT IN;

Query; select * from employees where department NOT IN ('sales', 'marketing');

output;

Exists!

query; select * from employees where Exists
(select * from orders where orders.emp_no =
(link unavailable));

output

NOT EXISTS!

query: select * from employees where not exists
(select * from order where
orders.emp_no = (link unavailable));

output;

All:

query: select * from employees WHERE salary
All (select salary from employees where
department = 'sales');

output!

ANY;

query: select * from employees where salary
ANY (select salary from employees where
department = 'sales');

output;

sql-> select * from order_master where order_no
(select order_no from orders where
order_no = 'o01');

output;

Task:04

Developing queries with DML multi-row function and operators.

```
SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> desc vasu;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 S_ID                                               NUMBER
 S_NAME                                             VARCHAR2(10)
 S_ADDRESS                                          VARCHAR2(12)
 PH_NO                                              NUMBER

SQL> desc course1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 C_ID                                               NUMBER
 C_NAME                                             VARCHAR2(11)

SQL> desc vasu_course1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 S_ID                                               NUMBER
 C_ID                                               NUMBER

SQL> select*from vasu;

10 rows selected
```

no rows selected

SQL> select*From course1;

no rows selected

SQL> select*from vasu course1;

no rows selected

SQL> insert into vasu values(12,'hari','yyyyyyyy',9063007191);

1 row created.

SQL> insert into vasu values(10,'ravi','uuuuuuuuu',123456788);

1 row created.

SQL> insert into vasu values(11,'ram','xyzu',123458);

1 row created.

SQL> insert into vasu values(1,'vishnu','xyzu',245658);

1 row created.

10 rows selected

```
SQL> insert into vasu values(2,'kalyan','xu',90876553);

1 row created.

SQL> select*from vasu;

      S_ID S_NAME     S_ADDRESS          PH_NO
---------- ---------- ---------- ----------
        12 hari       yyyyyyyy   9063007191
        10 ravi       uuuuuuuuu  123456788
        11 ram        xyzu       123458
         1 vishnu     xyzu       245658
         2 kalyan     xu         90876553

SQL> insert into course1 values(122,'physical');

1 row created.

SQL> insert into course1 values(152,'english');

1 row created.

SQL> insert into course1 values(112,'social');

10 rows selected
```

1 row created.

SQL> insert into course1 values(192,'math');

1 row created.

SQL> insert into course1 values(232,'fun');

1 row created.

SQL> select*from course1;

C_ID	C_NAME
122	physical
152	english
112	social
192	math
232	fun

SQL> insert into vasu_course1 values(45,55);

1 row created.

SQL> insert into vasu_course1 values(25,67);

10 rows selected

1 row created.

SQL> insert into vasu_course1 values(75,87);

1 row created.

SQL> insert into vasu_course1 values(85,90);

1 row created.

SQL> insert into vasu_course1 values(5,10);

1 row created.

SQL> select*from vasu_course1;

S_ID	C_ID
45	55
25	67
75	87
85	90
5	10

10 rows selected

SQL> select s_id from vasu;

```
      S_ID
----------
        12
        10
        11
         1
         2
```

SQL> select c_id from course1;

```
      C_ID
----------
       122
       152
       112
       192
       232
```

SQL> select c_id from vasu_course1;

```
      C_ID
----------
```

10 rows selected

55

67

87

90

10

SQL> select s_id from vasu union select c_id from course1;

S_ID

1

2

10

11

12

112

122

152

192

232

10 rows selected.

SQL> select s_id from vasu union all select c_id from course1;

10 rows selected

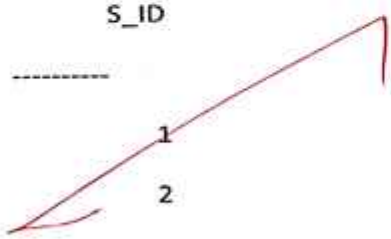
```
      S_ID
----------
        12
        10
        11
         1
         2
       122
       152
       112
       192
       232

10 rows selected.
```

SQL> select s_id from vasu intersect select c_id from course1;

no rows selected

SQL> select s_id from vasu minus select c_id from course1;

```
      S_ID
----------
         1
         2
```



10 rows selected

10

11

12

SQL> select s_id,count(*)from vasu group by s_id;

S_ID	COUNT(*)
1	1
11	1
2	1
12	1
10	1

SQL> select c_id,count(*)from course1 group by c_id having c_id is not null;

C_ID	COUNT(*)
152	1
112	1
232	1
192	1
122	1

SQL> select s_id,s_name,ph_no from vasu order by s_id;

10 rows selected

S_ID	S_NAME	PH_NO
1	vishnu	245658
2	kalyan	90876553
10	ravi	123456788
11	ram	123458
12	hari	9063007191

SQL> select s_id,s_name,ph_no from vasu order by s_id desc;

S_ID	S_NAME	PH_NO
12	hari	9063007191
11	ram	123458
10	ravi	123456788
2	kalyan	90876553
1	vishnu	245658

SQL> select ph_no+s_id from vasu;

PH_NO+S_ID

---

10 rows selected

123456798

123469

245659

90876555

SQL> insert into vasu select*from vasu where s_id in(select s_id from vasu);

5 rows created.

SQL> select*from vasu where s_id in(12,10);

S_ID	S_NAME	S_ADDRESS	PH_NO
12	hari	yyyyyyyy	9063007191
10	ravi	uuuuuuuuu	123456788
12	hari	yyyyyyyy	9063007191
10	ravi	uuuuuuuuu	123456788

SQL> select*from vasu where s_id not in(12,10);

S_ID	S_NAME	S_ADDRESS	PH_NO
11	ram	xyzu	123458
1	vishnu	xyzu	245658

10 rows selected

2 kalyan	xu	90876553
11 ram	xyzu	123458
1 vishnu	xyzu	245658
2 kalyan	xu	90876553

6 rows selected.

SQL> select*from vasu where exists(select*from course1 where course1.c_id=12);

no rows selected

SQL> select*from vasu where not exists(select*from course1 where course1.c_id=12);

S_ID	S_NAME	S_ADDRESS	PH_NO
12	hari	YYYYYYYY	9063007191
10	ravi	uuuuuuuuu	123456788
11	ram	xyzu	123458
1	vishnu	xyzu	245658
2	kalyan	xu	90876553
12	hari	YYYYYYYY	9063007191
10	ravi	uuuuuuuuu	123456788
11	ram	xyzu	123458
1	vishnu	xyzu	245658
2	kalyan	xu	9087655

10 rows selected

SQL->
select * from order_master where order_no =
(select order_no from orders);
output;

SQL-> select * from order_master where order_no=
any (select order_no from order_detail);

output;

SQL-> select* from order_master where order_no
in (select order_no from order_detail);

output;

SQL-> select * from order_detail where qty_ord
=all (select qty_hand from itemfile where
itemrate = 250);

output;

queries

[illegible] (5)	5
[illegible] AND ANALYSIS (5)	5
[illegible] VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	

Results;

Thus the developing sites with DML multi-row function and operators.

18/9/25

Task:05

Writing join queries, equivalent, and/or reducrsive queries.

Title: Implementation of different types of joins and recursive queries.

- A SQL join combines records from two in 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 = table_name2. columnname;

queries.

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 equalities, is called equi-join.

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 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 table 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, cust where item.id < cust.id;
```

Table Aliases:

Table aliases are used to make multiple table queries shorter and more readable. We give an aliases 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 row of the table to itself and also with other rows of the same table.

Example:

```
select * from empx, empy where x.salary >=
(select avg(salary) from x.emp where x.
depth=y.depth);
```

Outer join:

It extends the result of a simple join. An outer join returns all the rows returned by simple join as well as those rows from one SQL joins.

queries-

Inner join:

returns records that have matching values in both tables.

select column_name(s) from table1 Inner join table2 on table1.column_name = table2.column_name;

left outer join:

Return all recorded from the left table and the matched records from the right table.

select column_name(s) from table left join table2 on table1.column_name = table2, column_name;

Right (outer) join:

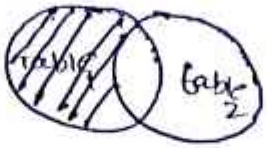
select column_name(s) from table1 right join table2 on table2.column_name = table column_name;

Full (outer) join:

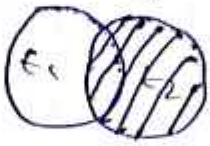
Return all records when there is an match in either left or right table select column_name(s) from table1.

full outer join table 2 on table 1. column_name = table 2. column_name;

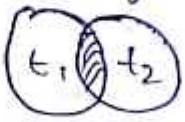
left join



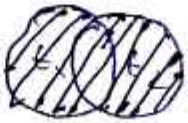
Right join



Inner join



full Outer join



consider the following two tables_ member and borrowed.

Inner join Query:

```
select borrowed. member, member, name from
member Inner join borrowed. on member
  member borrowed . membno;
```

left join Query:

```
select member. name, borrowed . membno .
from member right join borrowed on
borrowed . membno = member. membno;
```

SQL Right join keyword.

```
select member . name, borrowed . membno
from member = member. membno;
```

Full outer join keyword.

Tip:
full outer join and full join are the same

```
select member.name, borrowed.membno
from member full join member.membno.
```

LAB PRACTICE Assignment:

using the tables "departments" and "employees" 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 name and outer join.

Recursive Queries:

Syntax:

```
with Recuersive [cte_name] (column--]
A> ([non-recursive_term] union all
    [reursive-term])
    select ... from [cte_name];
```

Task:05 writing join queries, equivalent, and for recursive queries.

```
SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> create table vasu (member_no number,name varchar(11));

Table created.

SQL> desc naveen;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MEMBER_NO                                          NUMBER
 NAME                                               VARCHAR2(11)

SQL> insert into vasu values(1,'hari');

1 row created.

SQL> insert into vasu values(2,'syam');

1 row created.

SQL> insert into vasu values(3,'nem');

1 row created.

SQL> insert into vasu values(8,'david');
```

SQL> select*from paper;

MEMBER_NO	BOOK_ID
2	b101
3	b102
1	b103

SQL> select paper.member_no, vasu.name from Vasu inner join paper on vasu.member_no=paper.member_no;

MEMBER_NO	NAME
1	naveen
2	syam
3	nem

SQL> select paper.member_no,vasu.name from vasu right outer join paper on vasu.member_no=paper.member_no;

MEMBER_NO	NAME
1	hari
2	syam
3	nem

SQL> select paper.member_no,vasu.name from vasu left outer join paper on vasu.member_no=paper.member_no;

MEMBER_NO	NAME
2	syam

3 nem

1 hari

david

SQL> select paper.member_no,vasu.name from vasu right outer join paper on vasu.member_no=paper.member_no;

MEMBER_NO NAME

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

1 hari

2 syam

3 nem

SQL> select paper.member_no, vasu.name from vasufull outer join paperon vasu.member_no=paper.member_no;

MEMBER_NO NAME

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

1 hari

2 syam

3 nem

david

SQL

VEL TECH	
EX NO.	5
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	50
SIGN WITH DATE	25/9/25

Result:

Thus implementation of SQL commands using joins and recursion queries are executed successfully.

result.

1

2

4

8

16

32

64

128

256

loops

2

t

early

L

ds

ed

VEL TECH	
EX NO.	5
PERFORMANCE (5)	5
RESULT AND ANALYS'S (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	15/9/25

Result:

Thus implementation of SQL commands using joins and recursive queries are executed successfully.

Task [illegible]

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 language. it was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL. PL/SQL is one of three key programming languages embedded in oracle database, along with SQL its and Java.

Declarations:

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

Executable commands:

This section is enclosed between the keywords Begin and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, while may be just a null command to indicate that nothing should be executed.

Exception handling:

This section starts with the keyword EXCEPTION. This optional section contain exception(s) that handle errors in the programs.

simple program to print a sentence

syntax:

```
DECLARE
 <declaration section>
BEGIN
 < executable command s(s) >
  Exception
     <exception handling>
      END;
```

Program:

```
  DECLARE
message  varchar2(20) := " booking closed ";
  BEGIN
   dbms_output_line (message);
      END;
```

static input:

```
slow > set serveroutput on
   slow > 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.

Dynamic input;

```
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(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;
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');
    ELSIF (hid = 20) THEN
dbms_output.put_line('value of hid is 20');
    ELSIF (hid = 30) then
```

```
dbms_output.put_line ('value of hid 30');
ELSE
dbms_output.put_line ('one of the value')
ENDIF;
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.line ('hid is : ' hid ' and old
                                    is : ' old);
  end loop inner_loop;
    end loop outer_loop;
      END;
/
```

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

```
SQL> connect
Enter user-name: system
Enter password:
Connected.

SQL> set serveroutput on
SQL> declare
  2  x number(7);
  3  y number(8);
  4  z number(10);
  5  begin
  6  x:=10;
  7  y:=9;
  8  z:=x+y;
  9  dbms_output.put_line('sum is'||z);
 10  end;
 11  /
sum is19

PL/SQL procedure successfully completed.


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(var3);
 10  end;
 11  /
Enter value for var1: 30
old   6: var1:=&var1;
new   6: var1:=30;
Enter value for var2: 40
old   7: var2:=&var2;
new   7: var2:=40;
70

PL/SQL procedure successfully completed.


SQL> declare
  2  hid number(3):=100;
  3  begin
  4  if(hid=10)then
  5
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  else
 9  dbms_output.put_line('none of the values is matching');
10  end if;
11  dbms_output.put_line('exact valu of hid is:'||hid);
12  end;
13  /
none of the values is matching
exact valu of hid is:100

PL/SQL procedure successfully completed.

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:1and oid is:1
hid is:1and oid is:2
hid is:1and oid is:3
hid is:2and oid is:1
hid is:2and oid is:2
hid is:2and oid is:3
hid is:3and oid is:1
hid is:3and oid is:2
hid is:3and oid is:3

PL/SQL procedure successfully completed.

SQL>
```

simple program for only procedure:

```
sql> create or replace procedure c(information
  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 c(information (101,'ram');
PL/sql procdure successfully completed.
sql> set serveroutput on;
sql>exec c(information (101,'ram');
ID:101
name:ram
PL/sql procedure successfully completed.
```

VEL TECH	
EX NO.	6
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	25
TOTAL (20)	
SIGN WITH DATE	25/9/25

Result:-

Thus, TO implement PL/SQL procedures, functions and loop on number theory and business scenarios successfully.

Task: 7

Aim: To write PL/SQL programs using loops for printing prime number customer IDs and for demonstrating loop control in different scenarios.

Procedure:

Start a PL/SQL block or Produre.
use a cursor (if required) to fetch customer IDs from a table.
For each ID, check whether it is a prime number using a loop.
use FOR loop / WHILE loop to demonstrate prime number checking
print the results using
DBMS_OUTPUT.PUT_line.
End the block.

Example 1: using WHILE loop with cursor

prime check using WHILE loop

```
create or replace procedure print_prime_customers
cursor cust_cur is
select customer_id from customer;
    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%.NOT FOUND;
prime check using while loop
    If v_id <2 then
```

u_is_prime := FALSE;

else v_is_prime := true;
u_i := 2;
while v_i <= TRUNC (SQRT (v_id)) loop
if MOD (u_id, v_i) = 0 then
u_is_prime := false;
exist;
u_i := v_i + 1;
end loop;

end if;
if u_is prime then
DBMS_output.put line (prime customer ID'll v_id)
end if;
end loop;
close cust_cur;
END;

This procedure checks all customer Id's in the table and prints the prime ones using while loop.

Example:2

2 using FOR loop for first N prime number

create or replace procedure print_first n_prime (n number) is

u_num number := 2;
u_count number := 0;
u_is_prime BOOLEAN;

BEGIN

Task 1:7 writing PL/SQL using loops

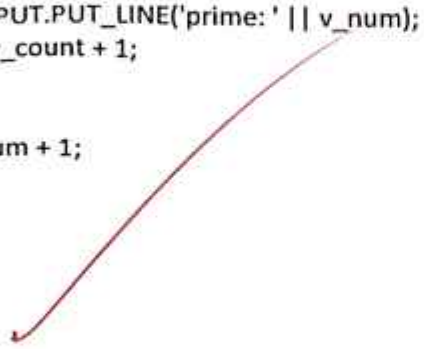
solve a map coloring problem using contains satisfaction approach.

```
SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> 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;

PL/SQL procedure successfully completed.

SQL> declare
  2   bk number(10):=&bk;
  3   s number(20):=0;
  4   r number(20);
  5   m number(20):=bk;
  6   len number(20);
  7  begin
  8   len:=trunc(log(10,bk))+1;
  9   while bk>0
 10    loop
 11     r:=mod(bk,10);
 12     s:=s+power(r,len);
 13     bk:=trunc(bk/10);
 14    end loop;
 15    if m=s
 16    then
 17     dbms_output.put_line('given number is armstrong');
 18    else
```

different scenarios executed succesfully

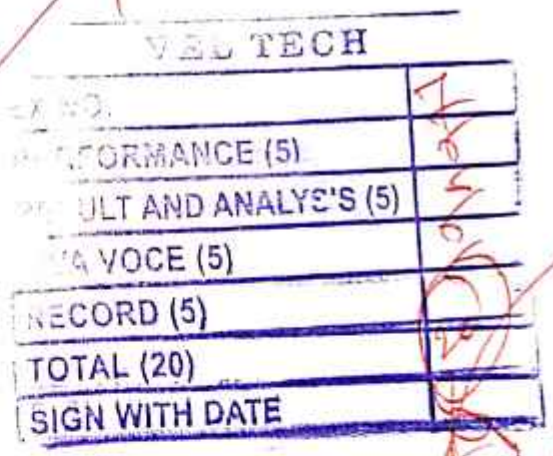
```
19    dbms_output.put_line('given number is not an armstrong');
20   end if;
21 end;
22 /
Enter value for bk: 1634
old  2:  bk number(10):=&bk;
new  2:  bk number(10):=1634;

PL/SQL procedure successfully completed.

SQL> CREATE OR REPLACE PROCEDURE print_prime_customers IS
 2  CURSOR cust_cur IS
 3    SELECT customer_id FROM customers;  -- Table with customer IDs
 4  v_id NUMBER;
 5  v_is_prime BOOLEAN;
 6  v_i NUMBER;
 7 BEGIN
 8  OPEN cust_cur;
 9  LOOP
10    FETCH cust_cur INTO v_id;
11    EXIT WHEN cust_cur%NOTFOUND;
12
13    -- Prime check using WHILE loop
14    IF v_id < 2 THEN
15      v_is_prime := FALSE;
16    ELSE
17      v_is_prime := TRUE;
18      v_i := 2;
19      WHILE v_i <= TRUNC(SQRT(v_id)) LOOP
20        IF MOD(v_id, v_i) = 0 THEN
21          v_is_prime := FALSE;
22          EXIT;
23        END IF;
24        v_i := v_i + 1;
25      END LOOP;
26    END IF;
27
28    IF v_is_prime THEN
29      DBMS_OUTPUT.PUT_LINE('Prime Customer ID: ' || v_id);
30    END IF;
31  END LOOP;
32  CLOSE cust_cur;
33 END;
34 /

Warning: Procedure created with compilation errors.

SQL> CREATE OR REPLACE PROCEDURE print_first_n_primes(n NUMBER) IS
 2    v_num    NUMBER := 2;     -- number to check for prime
 3    v_count  NUMBER := 0;     -- how many primes found so far
 4    v_is_prime BOOLEAN;
 5  BEGIN
 6    WHILE v_count < n LOOP
 7      v_is_prime := TRUE;
 8
 9      -- Prime check using FOR LOOP
10      FOR i IN 2 .. TRUNC(SQRT(v_num)) LOOP
```

```
        IF MOD(v_num, i) = 0 THEN
11        v_is_prime := FALSE;
12        EXIT;
13      END IF;
14    END LOOP;
15
16    IF v_is_prime THEN
17      DBMS_OUTPUT.PUT_LINE('prime: ' || v_num);
18      v_count := v_count + 1;
19    END IF;
20
21    v_num := v_num + 1;
22  END LOOP;
23 END;
24 /
```



lt:

Thus the solve a map coloring problem using constrains satisfaction approach is executed successfully using Python.



Results:

~~To~~ write PL/SQL programs using loops for printing prime number customers IDs and for demonstrating loop control in different scenarios executed successfully

upto BCNF

(Tool : GU / Table Normalization tool, ALM : Jigsaw).

upon relational tables created in task 2, perform normalization up to BCNF based on given dependencies as following for the assumed relations specified below.

Employee database:

1. Identify employee attributes:

Employee_ID, Name, Department, Job_Title, Manager_ID, Hire_Date, salary.

2. Define relation schema:

Employee (Employee_ID, Name, department, Job_Title, manager_ID, Hire_Date, salary).

3 Determine functional dependencies (FDs) between attributes.

Employee_ID → Name, department, Job_Title, manager_ID, Hire_Date, salary - Department → manager_ID - Manager_ID → Name.

Step:3

convert to 2NF

1. Ensure each non-key attribute depends on the entire primary key.

2. move non-key attributes to separate tables if they depend on only part of the primary key.

Task:8 Normalizing database using function 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.

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
```

VEL TECH	
EX NO.	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	

Result:

Thus the Implementation of normalizing data ba using functional dependencies upto BCNF

create depart table : Dept (Dept_ID, manger_ID name)

create employee table: employee (emp_ID, Name, dept-ID, job_title, Alize_Data, salary).

step4: convert to 3NF

ensure there one no transtive dependencies, more non-trey attributes to seperate table depends on another non_treys attributes.

create manger table: manger (manger- manger_ID),

step 5: convert to BCNF

ensure every determinate is a (ardinate trey).

check for overlapping candinate treys.

decompose relations to elimate redway. no further decomposition needed.

using Griffith Tool:

1. Input relational schema and functional dependences.
2. Griffith tool generate a dependenary graph.
3. Analayze the graph to identify issuess.
4. verify the resulting schema ~~meets~~ BCNF critiera.

VEL TECH	
EX NO.	8
PERFORMANCE (5)	5
RESULT AND ANALYS'S (5)	5
VIVA VOCE (5)	5
RECORD (5)	23
TOTAL (20)	
SIGN WITH DATE	

Result:

Thus the Implemenation of normalizing data base using functional dependencies upto BCNF.

Task NO: 9

Backing up and recovery in database.

Perform following backup and recovery the scenario.

a. Recovering a NOARCHIVELOG database with incremental backups.

b. Restoring the server parameter file.

c. performing Recovery with a Backup control file.

Scenario1;

Recovering a NOARCHIVELOG database with incremental Backups

step1; Back up database.

BACKUP DATABASE : [database_name] TO DISK = 'backup_file_back' with NO FARMAT, BACKUP Database, name = 'Full Databasesup', SKIP, REWIND, NOULOD STATS = 10,

step2; create incremental Backup.

Backup database [database_name] TO DISK = 'incremental_ backr_up. bak', with differental, NoRORMAT, NOINT, Name = 'Incremental Backup', strip Rewind, NeulAD, Stats = 10.

step3; simulate data loss

* intentionally delete or modify data.

step4; open data base.

Recover.

Scenario2:

Restoring the server parameter file (spfile)

step1: Backup SPFILE;

Backup server parameter file to file = "spfile bak"

step2: simulate spfile loss

-Delete or modify spfile.

step3: Restore spfile

STARTUP MOUNT

Restore server parameter file from file = "spfile bak"

SHUT DOWN

START UP.

Scenario 3:

Performing the sever parameter file (spfile). control.

Back file.

step2 : Backup control file

Backup control file to file = "control_back".

step3: Restore control file

START UP MOUNT

Restore. control file from file = "control.file"

ALTER control file. Reuse;

step4: Recover database

RECOVER DATABASE USING BACKUP CONTRO

LFILE;

step5 open database

ALTER DATABASE OPEN RESETLOGS;

SQL server commands:

- Backup database
- Restore database
- Recover database
- ALTER database
- Backup server 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)	5
TOTAL (20)	20
SIGNATURE	

Result:

Thus the Implemenation of backing up and recovery in database.

Task :-10

crud operation in document database.

Aim:

To perform mongoose using NPM design on mongoDB designing document database and base performing CRUD operations like creating, inserting, querying, finding and removing operations.

step1:

install mongo db using following link

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

step2:

install mongosh using the below link

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 variable:

open the control panel.

In the system and security category, click system.

click Advanced system settings.

The system properties modal displays.

click environment variables.

In the system variables section, select path and click Edit.

The Edit environment variable modal displays.

Click New and add the file path to your mongosh binary.

Click ok to confirm your changes, on each other modal.

click ok to confirm your changes.
To confirm that your PATH environment variable is correctly configured to find mongosh, open a command prompt and enter the mongosh --help command.
If your PATH is configured correctly, a list of valid commands displays.

step4:
open mongo shell i.e from
C:\program files\mongoDB\server\bin\mongod.exe

steps:
Type the CRUD (CREATE READ UPDATE DELETE) COMMANDS GIVEN IN TEXT FILE.

CRUD OPERATIONS:

```
db.createcollection("mylab")
   { "ok":1}
>
db.mylab.insertone({item:"canvas", qty:100, tags:
["cotton"], size: {h:28, w:35.5, uom:"cm"}})
{
"acknowledged": true,
   "insertedId":
ObjectId("627d13acc73990c07ue6397c")
}
> db.mylab.find({item:"canvas"})
   {"_id":
ObjectId("627d130cc73990c07ue6397c"),
   "item": "canvas", "qty": 100, "tags": ["cotton"],
"size": {"h":28, "w":35.5, "uom":"cm"}}
>
```

```
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: "mouse pad", qty:25, tags: ["gel", "blue"]
size: {h:19, w:22.85, uom: "cm"}}]) {
"acknowledged": true,
"insertedIds": [
objectId("627d1598c73990c07ue6397d"),
objectId("627d1598c73990c07ue6397e"),
objectId("627d1598c73990c07ue6397f")]}
>db.mylab.find({}, {item:1, qty:1}) {"_id":
objectId("627d13acc73990c07ue6397c"),
"item": "canvas", "qty": 100}
{"_id":
objectId("627d1598c73990c07ue6397d"),
"item": "journal", "qty":25} {"_id":
objectId("627d1598c73990c07ue6397e"),
"item": "mat", "qty":85} {"_id":
objectId("627d1598c73990c07ue6397f"),
"item": "mousepad", "qty":25}
>db.mylab.find({}, {item:1, qty:1}).pretty() {
"_id":
objectId("627d13acc73990c07ue6397c"),
"item": "canvas",
"qty": 100
}
{
"_id":
```

```
objectId ("627d1598c73990co7ue6397d"),
  "item": "journal",
    "qty": 25
  }
{"_id":
objectId ("627d1598c73990co7ue6397e"),
  "item": "mat", "qty": 85 }
  {
"_id":
objectId ("627d1598c73990co7ue6397d"),
    "item": "mousepad",
      "qty": 25
    }
  >
db.mylab.find ({item: "canvas"}).
  pretty().sort ({item: -1})
  {
"_id":
objectId ("627d13acc73990co7ue6397c"),
  "item": "canvas",
    "qty": 100,
    "tags": ["cotton"],
"size": {"h": 28, "w": 35.5, "uom": "cm"}}
}
>db.mylab.deleteone ({item: "journal"})----
>db.mylab.find ({}, {item:1, qty:1}).pretty() {
    "_id":
objectId ("627d13acc73990co7ue6397c"),
        "item": "canvas",
          "qty": 100
```

```
}
{
"_id" :
objectId ("627d1598c7399oco7ue6397d"),
  "item" : "journal",
  "qty" : 25

}
{ "_id" :
objectId ("627d1598c7399oco7ue63a7e"),
"item" : "mat", "qty" : 85}
{
"_id" :
objectId ("627d1598c7399oco7ue6397f"),
"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	

Result:

Thus implementation of CRUD operations like creating inserting, finding and removing operations using MongoDB is successfully executed.

Task 11:

CRUD OPERATIONS IN GRAPH DATABASES

Aim:

To perform CRUD operations like creating, inserting, querying, finding deleting operation on graph spaces.

- Create Node with properties

properties are the key-value pairs using which a node stores data. You can create a node with properties using the CREATE clause. You need to specify these properties separated by commas within the flower braces "{}".

Syntax:

following is the syntax to create a node 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 within the square braces "[]" depending on the direction of the relationship it is placed between hyphen "-" and allow "->" as shown in the following syntax.

syntax:

Following is the syntax to create a relationship using the CREATE clause.

CREATE
(node1)-[: Relationship type]->(node2)

◆ creating a Relationship between the existing Nodes.

You can also create a relationship between the existing nodes using.
the MATCH clause.

syntax:

Following is the syntax to create a relationship using the MATCH clause.

MATCH (a: Label of Node1),
(b : Label of Node2)
where a.name = "name of node1"
AND b.name = "name of node2"
CREATE (a)-[: Relation]->(b)RETURN a,b

◆ Deleting a Particular Node

To delete a particular node, you need to specify the details of the node in the place of "n" in the above query.

syntax:

following is the syntax to delete a particular node from Neo4j using the DELETE clause.

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

create a graph database for student course registration create student and dept node and insert values of properties.

```
create (n: student {sid: "VTU14500",
  sname: "John",
  deptname: "CSE"})
```

OUTPUT:

Added 1 label, created 1 node, set 3 properties, completed after 232 ms.

```
create (n: student {sid: "VTU14501",
  sname: "Dharsana",
  deptname: "EEE"})
```

OUTPUT:

Added 1 label, created 1 node, set 3 properties, completed after 16 ms.

```
create (n: student {sid: "VTU14502",
  sname: "vijay",
  deptname: "CSE"})
```

OUTPUT:

Added 1 label, created 1 node, set 2 properties completed after 72 ms.

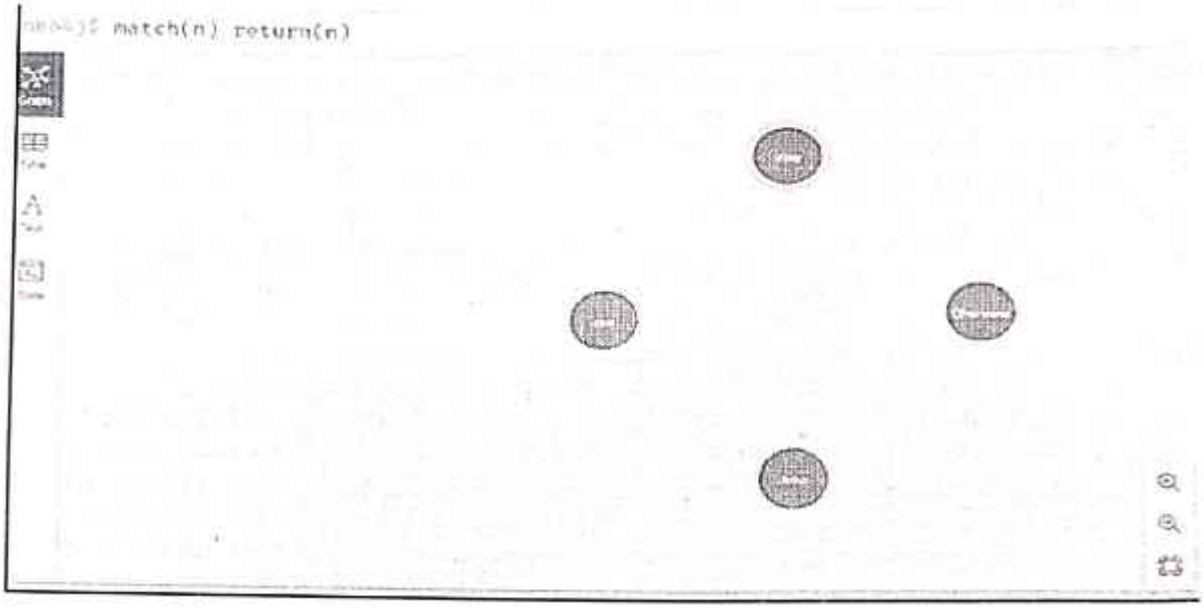
select all the nodes in your database using match command.

```
match (n) return (n)
```

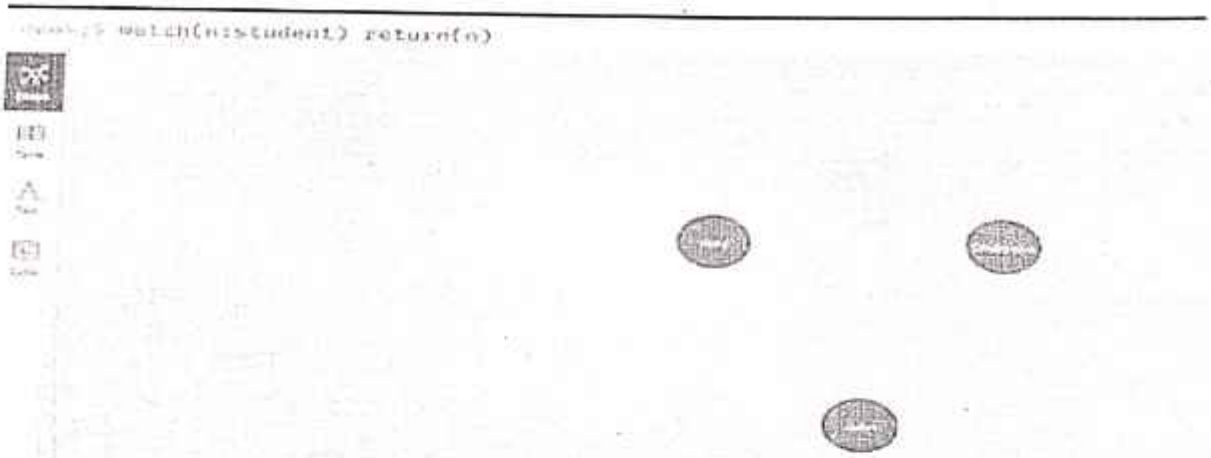
Task:11 CRUD operation In Graph database

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
```

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

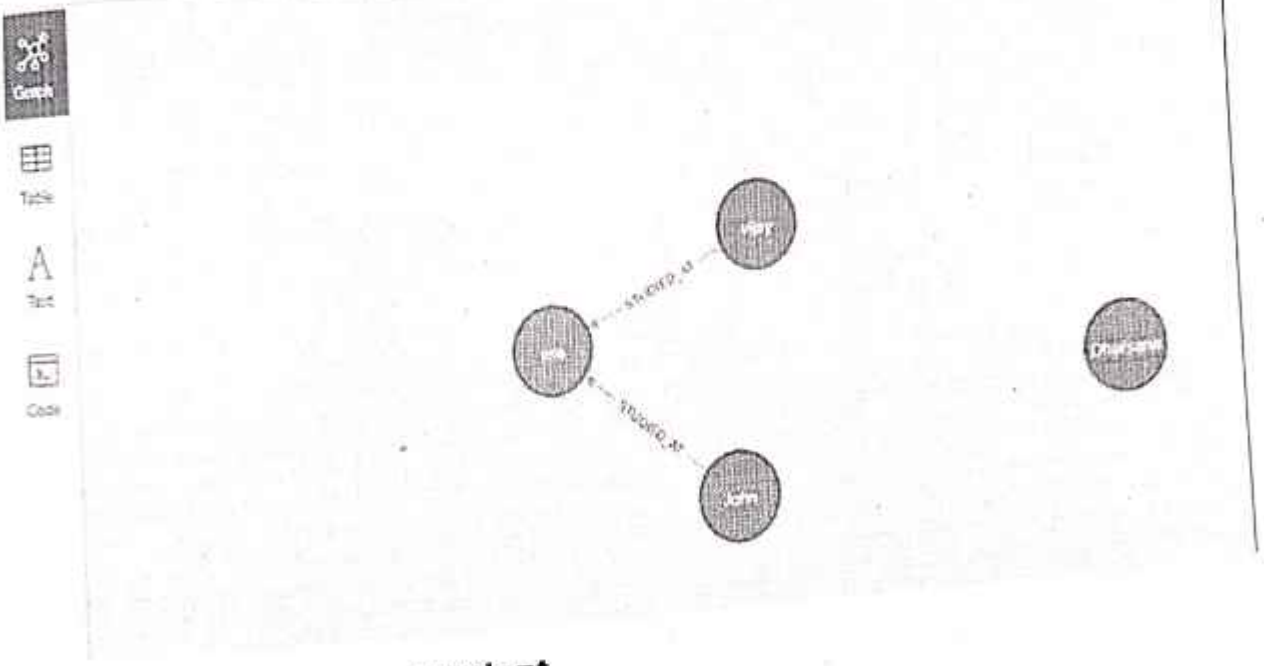
MATCH(s:student),(d:dept) WHERE s.Sname ='John' AND d.deptname='cse'

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	
X NO.	11
FORM (5)	5
S'S (5)	5
RES T AND	5
VIVA V E (5)	5
RECORD	25
TOTAL (20)	
SIGN WI	

9/10/25

Result:

Thus implementation of CRUD operations like creating, inserting finding and removing operation succe

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	10/11/25

Result:

Thus implementation of CRUD operations like creating, inserting finding and removing operations using GraphDB is successfully executed.

Date : 16/10/25

DATABASE MANAGEMENT SYSTEMS

(10211CS207)

Tourism ManagementSystem

Team Details:

Team Leader: B .Vasu

Team Members:

Name:	vtu:	Registration no:
N.m.sai Kishore	30480	24UEDC0037
P.sunil rao	30471	24UEDC0052
B.vasu	30458	24UEDC0008
P.v.satyanarayana	30486	24UEDC0056
K.ashok reddy	30453	24UEDC0031

Aim:

To develop a tourism management system

ER Diagram:

Travel and tourism booking systems

1. User: Represents individuals who interact with the system

- **UserID (Primary Key):** Unique identifier for each user.
- **Username:** Name chosen by the user for login.
- **Password:** Securely stored password for login authentication.
- **Email:** Email address associated with the user's account.
- **Name:** Full name of the user.
- **Address:** Physical address of the user.
- **Phone:** Contact the phone number of the user.

2. Booking: Records details of each reservation made by users

- **BookingID (Primary Key):** Unique identifier for each booking.
- **UserID (Foreign Key):** References the user who made the booking.
- **BookingDate:** Date when the booking was made.
- **TotalAmount:** Total amount payable for the booking.
- **Status:** Status of the booking (e.g., pending, confirmed, cancelled).

3. Flight: Stores information about available flights

- **FlightID (Primary Key):** Unique identifier for each flight.
- **Airline:** Name of the airline operating the flight.
- **DepartureAirport:** Departure airport for the flight.
- **DestinationAirport:** Destination airport for the flight.
- **DepartureDateTime:** Date and time of departure.
- **ArrivalDateTime:** Date and time of arrival.
- **Price:** Price of the flight ticket.
- **AvailableSeats:** Number of available seats on the flight.

4. Accomodation: Represents available lodging options

- **AccommodationID (Primary Key):** Unique identifier for each accommodation.
- **Name:** Name or title of the accommodation.
- **Location:** Location or address of the accommodation.
- **CheckInDate:** Date for check-in.
- **CheckOutDate:** Date for check-out.
- **PricePerNight:** Price per night for the accommodation.
- **AvailableRooms:** Number of available rooms in the accommodation.

5. Activity: Manages information about activities or tours available

- **ActivityID (Primary Key):** Unique identifier for each activity.
- **Name:** Name or title of the activity.
- **Location:** Location or address of the activity.
- **Date:** Date of the activity.
- **Time:** Time of the activity.
- **Price:** Price of the activity.
- **Capacity:** Maximum capacity or number of participants for the activity.

Relationship Between These Entities

1. User - Accommodation Relationship (Many-to-Many):

- Users can book multiple hotels, indicating a user can make bookings for different accommodations.
- Every accommodation can be booked by multiple users, meaning a hotel can have bookings from different users.

2. User - Booking Relationship (Many-to-One):

- Many bookings can be associated with one user, showing that a user can make mtultiple bookings over time.

3. User - Activity Relationship (Many-to-Many):

- Users can book multiple activities, allowing a user to participate in various activities.

- Every activity can be booked by multiple users, indicating that an activity can have participants from different users.

4. Booking - Activity Relationship (Many-to-One):

- Many activities can be associated with one booking, meaning that a booking can include multiple activities.

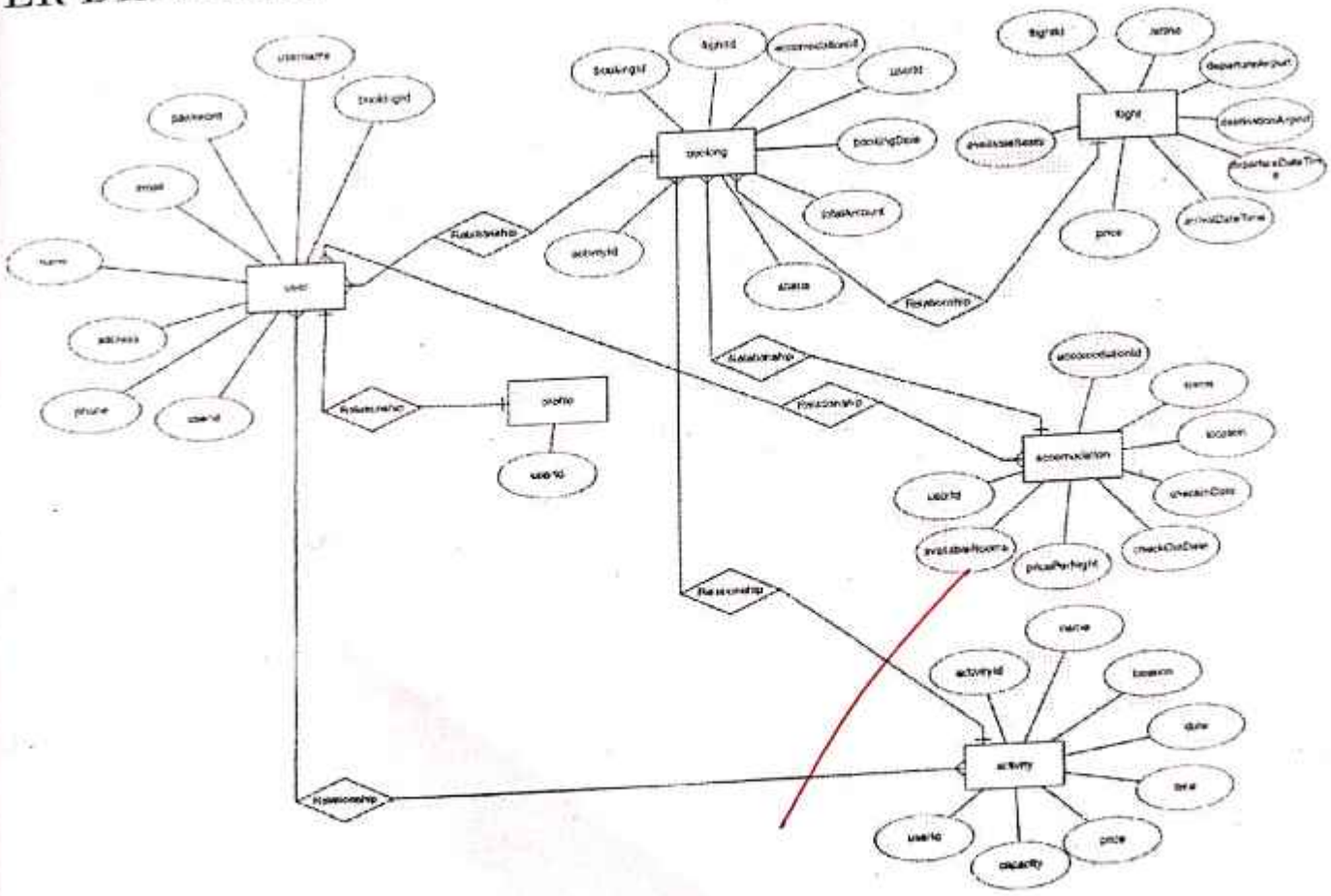
5. Booking - Accommodation Relationship (Many-to-One):

- Many hotels can be associated with one booking, showing that a booking can include stays at multiple hotels.

6. Booking - Flight Relationship (Many-to-One):

- Many bookings can be associated with one flight, indicating that a booking can include a flight reservation.

ER DIAGRAM:



2. PROJECT Operation

(Project → selecting specific columns)

SELECT Name, Email FROM Tourist;

→ This extracts only specific attributes (Name, Email).

```
SQL> SELECT train_name, fare FROM Train;

TRAIN_NAME                                          FARE
-------------------------------------------------- ----------
Shatabdi Express                                         1200
Rajdhani Express                                         2500
Duronto Express                                          2300
Tejas Express                                            1500
```

3. UNION Operation

Show destinations from both booked packages and available packages (removes duplicates).

SELECT Destination FROM Package
UNION
SELECT Destination FROM Booking

```
SQL> SELECT source AS station FROM Train
  2  UNION
  3  SELECT destination AS station FROM Train;

STATION
--------------------------------------------------
Bhopal
Chandigarh
Delhi
Kolkata
Mumbai
```

JOIN Package ON Booking.Package_ID = Package.Package_ID;

4. UNION ALL Operation

Same as UNION but **includes duplicates.**

SELECT Destination FROM Package
UNION ALL
SELECT Destination FROM Booking
JOIN Package ON Booking.Package_ID = Package.Package_ID;

```
SQL> SELECT source AS station FROM Train
  2  UNION ALL
  3  SELECT destination AS station FROM Train;

STATION
--------------------------------------------------
Delhi
Mumbai
Kolkata
Delhi
Bhopal
Delhi
Delhi
Chandigarh

8 rows selected.
```

5. INTERSECT Operation

Show destinations that **are available and already booked.**

SELECT Destination FROM Package
INTERSECT
SELECT Destination FROM Booking
JOIN Package ON Booking.Package_ID = Package.Package_ID;

```
SQL> SELECT source FROM Train
  2  INTERSECT
  3  SELECT destination FROM Train;

SOURCE
--------------------------------------------------
Delhi
```

```
SQL> SELECT SUM(fare) AS total_revenue FROM Train;

TOTAL_REVENUE
-------------
         7500
```

7. COUNT Operation

Count number of tourists who made bookings:

SELECT COUNT(DISTINCT Tourist_ID) AS Total_Tourists FROM Booking;

```
SQL> SELECT COUNT(*) AS total_bookings FROM Booking;

TOTAL_BOOKINGS
--------------
             8
```

8. AVG, MIN, MAX

Find average, minimum, and maximum package price:

SELECT
  AVG(Price) AS Average_Price,
  MIN(Price) AS Minimum_Price,
  MAX(Price) AS Maximum_Price
FROM Package;

```
AVG_FARE   MIN_FARE   MAX_FARE
------------------------------
    1875       1200       2500
```

. JOINS

a) INNER JOIN

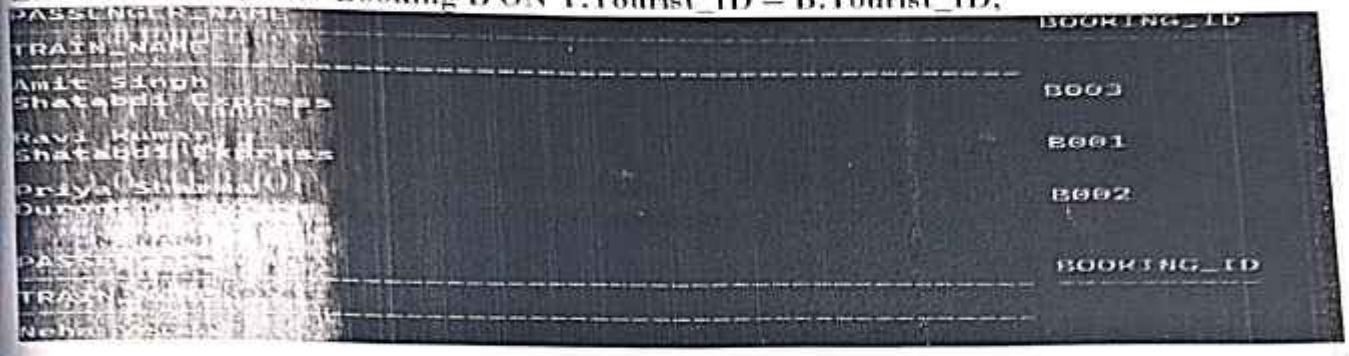
Show tourists and their booked package names:

SELECT T.Name, P.Package_Name, B.Booking_Date
FROM Tourist T
INNER JOIN Booking B ON T.Tourist_ID = B.Tourist_ID
INNER JOIN Package P ON B.Package_ID = P.Package_ID;

b) LEFT OUTER JOIN

Show all tourists (even those who haven't booked anything):

```
SELECT T.Name, B.Booking_ID
FROM Tourist T
LEFT OUTER JOIN Booking B ON T.Tourist_ID = B.Tourist_ID;
```



c) RIGHT OUTER JOIN

Show all bookings (even if some tourists' details are missing):

```
SELECT T.Name, B.Booking_ID
FROM Tourist T
RIGHT OUTER JOIN Booking B ON T.Tourist_ID = B.Tourist_ID;
```

d) FULL OUTER JOIN

Show all tourists and all bookings (matched + unmatched records):

```
SELECT T.Name, B.Booking_ID
FROM Tourist T
FULL OUTER JOIN Booking B ON T.Tourist_ID = B.Tourist_ID;
```

3. Normalization:

Normalization in the context of databases refers to the process of redundancy and dependency by organizing fields and table of a organizing data in a database efficiently. The goal is to reduce data database. This helps in minimizing the anomalies that can arise when modifying the data.

There are several normal forms (NF) that define the levels of normalization, with each normal form addressing different types of issues:

First Normal Form (INF):

Eliminate duplicate columns from the same table.

Create a separate table for each group of related data and identify each row with a unique column or set of columns.

Second Normal Form (2NF):

Meet all the requirements of INF.

Remove partial dependencies-ensure that non-prime attributes are fully functionally dependent on the primary key.

Boyce-Codd Normal Form (BCNF):

A more stringent form of 3NF.

For a table to be in BCNF, it must satisfy an additional requirement compared to 3NF, dealing specifically with certain types of functional dependencies.

In this database we perform normalisation using Griffith university normalisation tool

Check normal from:

Check Normal Form

2NF
The table is in 2NF

3NF
The table is in 3NF

BCNF
The table is in BCNF

ormalize to 2NF

Normalize to 2NF

Attributes

user_id | traveler_name | phone_no | location | email | hostel | hostel_id | room_no

Functional Dependencies

location → hostel
hostel → traveler_name hostel_id room_no
traveler_name → phone_no user_id email

how Steps

First, find the minimal cover of the FDs, which includes the FDs
location → hostel
hostel → traveler_name
hostel → hostel_id
hostel → room_no
traveler_name → phone_no
traveler_name → user_id
traveler_name → email

ormalize to 3NF

1NF to 3NF

Attributes

location hostel

Functional Dependencies

hostel

Attributes

traveler_name phone_no user_id email

Functional Dependencies

traveler_name phone_no user_id email

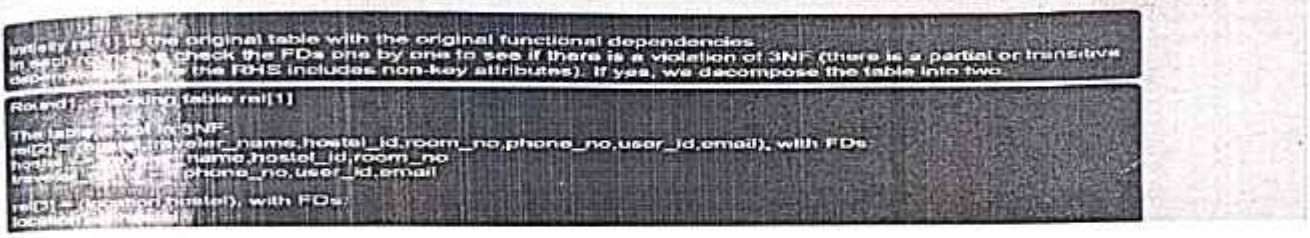
Attributes

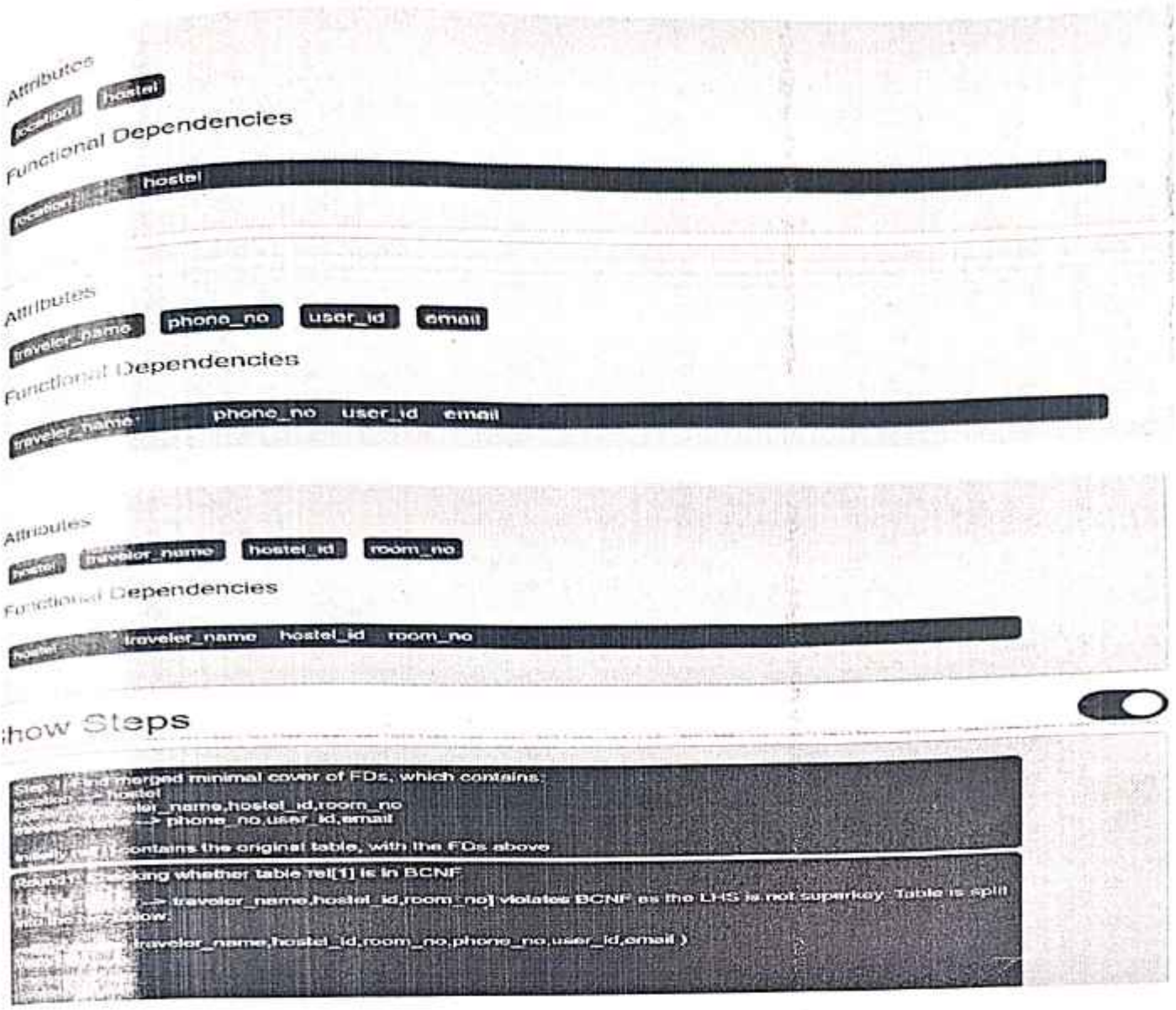
hostel traveler_name hostel_id room_no

Functional Dependencies

hostel traveler_name hostel_id room_no

Show Steps





xplanation:

lormalization ensures:

- No data duplication (Tourist details appear once)
- Easy updates (change one field, affects all references)
- Reliable relationships via foreign key

Using Griffith Tool:

If you're using the **Griffith Normalization Tool (Online/Software)**:

1. Open **Griffith Normalization Tool.**
2. Input the **unnormalized table** (like the first one).
3. Click **"Normalize" → 1NF → 2NF → 3NF.**

Thus, the normalization to 1NF,2NF,3NF,BCNF is completed successfully.

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.

```
use travel_tourism;

db.tourists.insertMany([
 { _id: 1, name: "Kishore", email: "kishore@email.com", phone: "9876543210", country: "India" },
 { _id: 2, name: "Meena", email: "meena@email.com", phone: "9988776655", country: "India" }
]);
db.packages.insertMany([
 { _id: 101, package_name: "Goa Delight", destination: "Goa", duration_days: 5, price: 25000, description: "Enjoy beaches, water sports, and luxury stay." },
 { _id: 102, package_name: "Royal Rajasthan", destination: "Jaipur", duration_days: 6, price: 22000, description: "Desert safari, forts, and palace tours." }
]);
db.bookings.insertMany([
 { booking_id: "B001", tourist_id: 1, package_id: 101, booking_date: "2025-10-10", total_amount: 25000, payment_status: "Paid" },
 { booking_id: "B002", tourist_id: 2, package_id: 102, booking_date: "2025-10-12", total_amount: 22000, payment_status: "Pending" }
]);
```

```
db.payments.insertMany([
 { payment_id: "P001", booking_id: "B001", amount: 25000, method: "UPI",
payment_date: "2025-10-10" },
 { payment_id: "P002", booking_id: "B002", amount: 22000, method: "Card",
payment_date: "2025-10-12" }
]);
db.tourists.find().pretty();
db.packages.find().pretty();
db.bookings.find({ payment_status: "Paid" });
db.bookings.aggregate([
  { $lookup: { from: "tourists", localField: "tourist_id", foreignField: "_id", as:
"TouristDetails" } }
]).pretty();
db.bookings.updateOne({ booking_id: "B002" }, { $set: { payment_status: "Paid" } });
db.packages.updateOne({ _id: 102 }, { $set: { price: 23000 } });
db.tourists.deleteOne({ name: "Meena" });
db.packages.deleteOne({ destination: "Jaipur" });
db.bookings.aggregate([{ $group: { _id: null, total_revenue: { $sum: "$total_amount" } }
}]);
db.tourists.aggregate([{ $count: "Total_Tourists" }]);
db.packages.aggregate([{ $group: { _id: null, avg_price: { $avg: "$price" } } }]);
db.bookings.aggregate([
  { $lookup: { from: "tourists", localField: "tourist_id", foreignField: "_id", as:
"TouristDetails" } },
  { $lookup: { from: "packages", localField: "package_id", foreignField: "_id", as:
"PackageDetails" } },
  { $lookup: { from: "payments", localField: "booking_id", foreignField: "booking_id", as:
"PaymentDetails" } }
]).pretty();
```

OUTPUT:

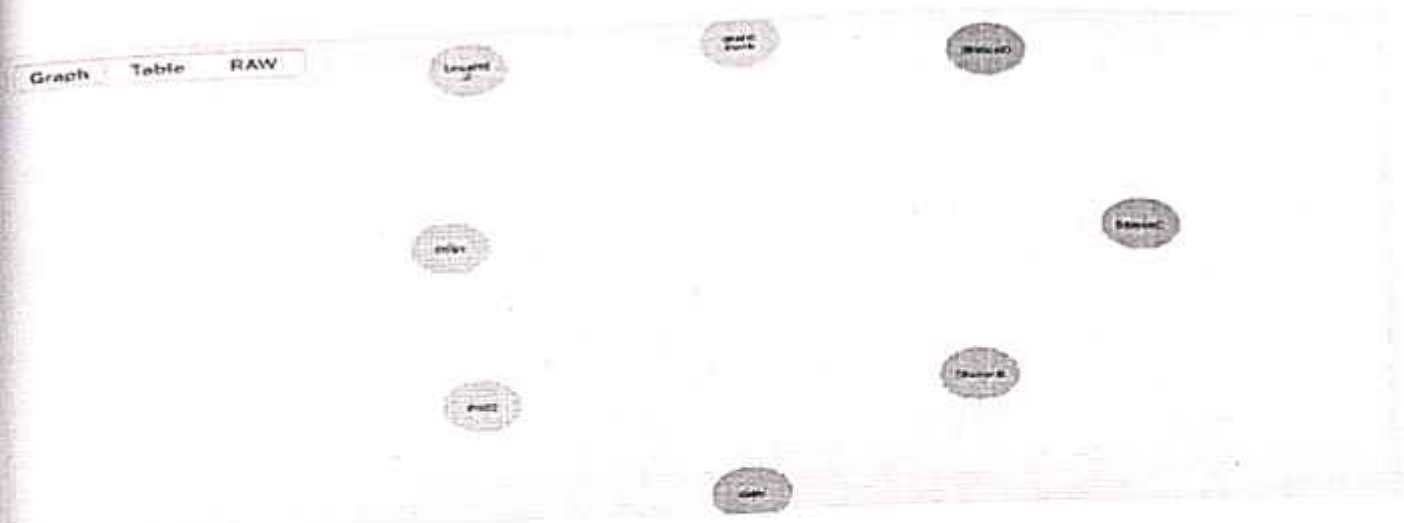
```
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("68f9cc70f219c70249bbb9ca"),
                ObjectId("68f9cc70f219c70249bbb9cb"),
                ObjectId("68f9cc70f219c70249bbb9cc")
        ]
}
uncaught exception: SyntaxError: unexpected token: identifier :
@(shell):1:39
{ "acknowledged" : true, "insertedIds" : [ 1, 2 ] }
{ "acknowledged" : true, "insertedIds" : [ 101, 102 ] }
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("68f9cc70f219c70249bbb9cd"),
                ObjectId("68f9cc70f219c70249bbb9ce")
        ]
}
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("68f9cc70f219c70249bbb9cf"),
                ObjectId("68f9cc70f219c70249bbb9d0")
        ]
}
{
        "_id" : 1,
        "name" : "Kishore",
        "email" : "kishore@email.com",
        "phone" : "9876543210",
        "country" : "India"
}
{
        "_id" : 2,
        "name" : "Meena",
        "email" : "meena@email.com",
        "phone" : "9988776655",
        "country" : "India"
}
{
        "_id" : 101,
        "package_name" : "Goa Delight",
        "destination" : "Goa",
        "duration_days" : 5,
        "price" : 25000,
        "description" : "Enjoy beaches, water sports, and luxury stay."
}
{
        "_id" : 102,
        "package_name" : "Royal Rajasthan",
        "destination" : "Jaipur",
        "duration_days" : 6,
        "price" : 22000,
        "description" : "Desert safari, forts, and palace tours."
}
{ "_id" : ObjectId("68f9cc70f219c70249bbb9cd"), "booking_id" : "B001", "tourist_id" : 1, "package_id" :
101, "booking_date" : "2025-10-10", "total_amount" : 25000, "payment_status" : "Paid" }
{
        "_id" : ObjectId("68f9cc70f219c70249bbb9cd"),
        "booking_id" : "B001",
        "tourist_id" : 1,
        "package_id" : 101,
        "booking_date" : "2025-10-10",
        "total_amount" : 25000,
        "payment_status" : "Paid",
        "TouristDetails" : [
                {
                        "_id" : 1,
                        "name" : "Kishore",
                        "email" : "kishore@email.com",
                        "phone" : "9876543210",
                        "country" : "India"
                }
        ]
}
{
        "_id" : ObjectId("68f9cc70f219c70249bbb9ce"),
        "booking_id" : "B002",
        "tourist_id" : 2,
```

```
    "package_id" : 102,
    "booking_date" : "2025-10-12",
    "total_amount" : 22000,
    "payment_status" : "Pending",
    "TouristDetails" : [
            {
                "_id" : 2,
                "name" : "Meena",
                "email" : "meena@email.com",
                "phone" : "9980776655",
                "country" : "India"
            }
    ]
}
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
{ "acknowledged" : true, "deletedCount" : 1 }
{ "acknowledged" : true, "deletedCount" : 1 }
{ "_id" : null, "total_revenue" : 47000 }
{ "total_tourists" : 1 }
{ "_id" : null, "avg_price" : 25000 }
{
    "_id" : ObjectId("68f9cc70f219c70249bbb9cd"),
    "booking_id" : "B001",
    "tourist_id" : 1,
    "package_id" : 101,
    "booking_date" : "2025-10-10",
    "total_amount" : 25000,
    "payment_status" : "Paid",
    "TouristDetails" : [
            {
                "_id" : 1,
                "name" : "Kishore",
                "email" : "kishore@email.com",
                "phone" : "9876543210",
                "country" : "India"
            }
    ],
    "PackageDetails" : [
            {
                "_id" : 101,
                "package_name" : "Goa Delight",
                "destination" : "Goa",
                "duration_days" : 5,
                "price" : 25000,
                "description" : "Enjoy beaches, water sports, and luxury stay."
            }
    ],
    "PaymentDetails" : [
            {
                "_id" : ObjectId("68f9cc70f219c70249bbb9cf"),
                "payment_id" : "P001",
                "booking_id" : "B001",
                "amount" : 25000,
                "method" : "UPI",
                "payment_date" : "2025-10-10"
            }
    ]
}
{
    "_id" : ObjectId("68f9cc70f219c70249bbb9ce"),
    "booking_id" : "B002",
    "tourist_id" : 2,
    "package_id" : 102,
    "booking_date" : "2025-10-12",
    "total_amount" : 22000,
    "payment_status" : "Paid",
    "TouristDetails" : [ ],
    "PackageDetails" : [ ],
    "PaymentDetails" : [
            {
                "_id" : ObjectId("68f9cc70f219c70249bbb9d0"),
                "payment_id" : "P002",
                "booking_id" : "B002",
                "amount" : 22000,
                "method" : "Card",
                "payment_date" : "2025-10-12"
            }
    ]
}
```


6.Graph Database using MONGODB:

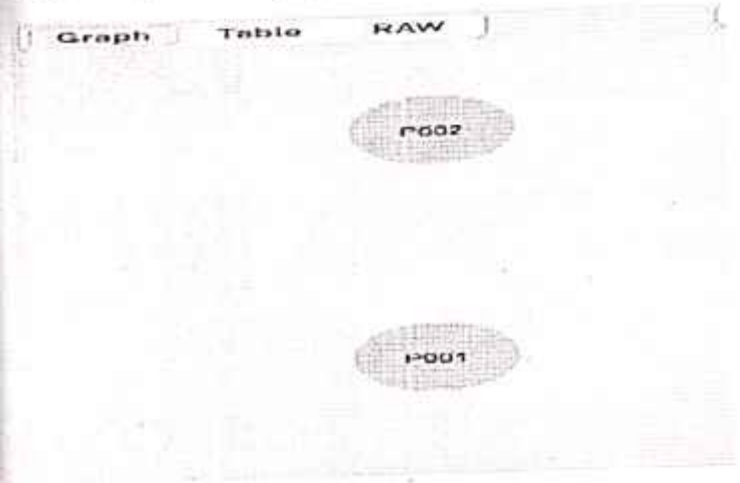
ALL NODES:

MATCH (n) RETURN n LIMIT 25;



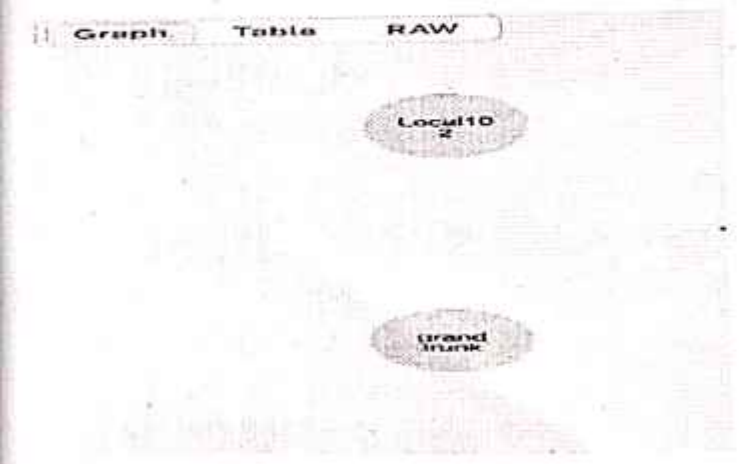
TRAVELER NODE:

MATCH (n:Passenger) RETURN n LIMIT 25;



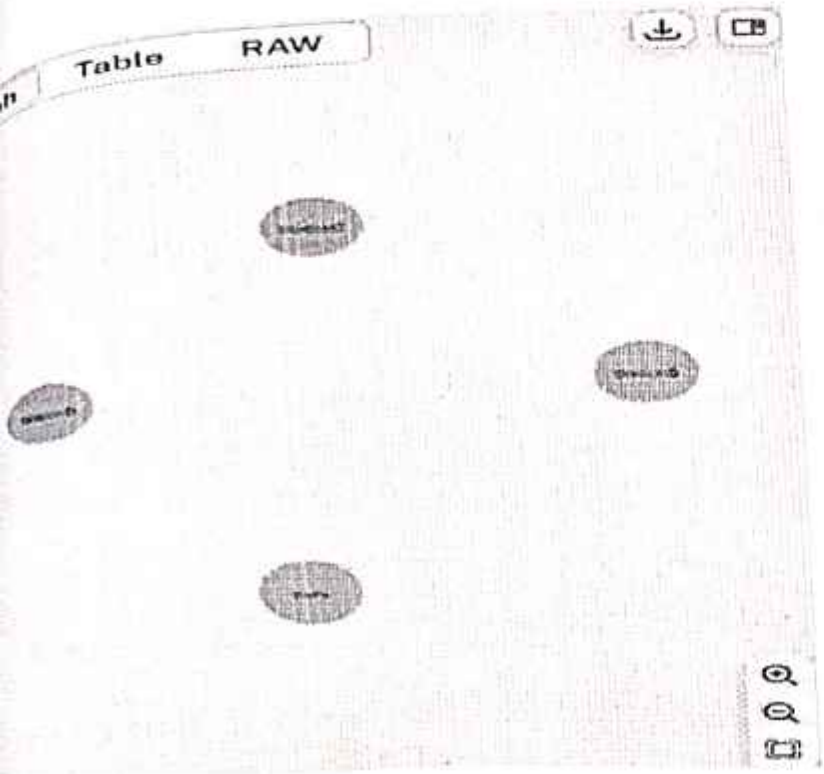
PLACE NODE:

MATCH (n:PLACE L) RETURN n LIMIT 25;



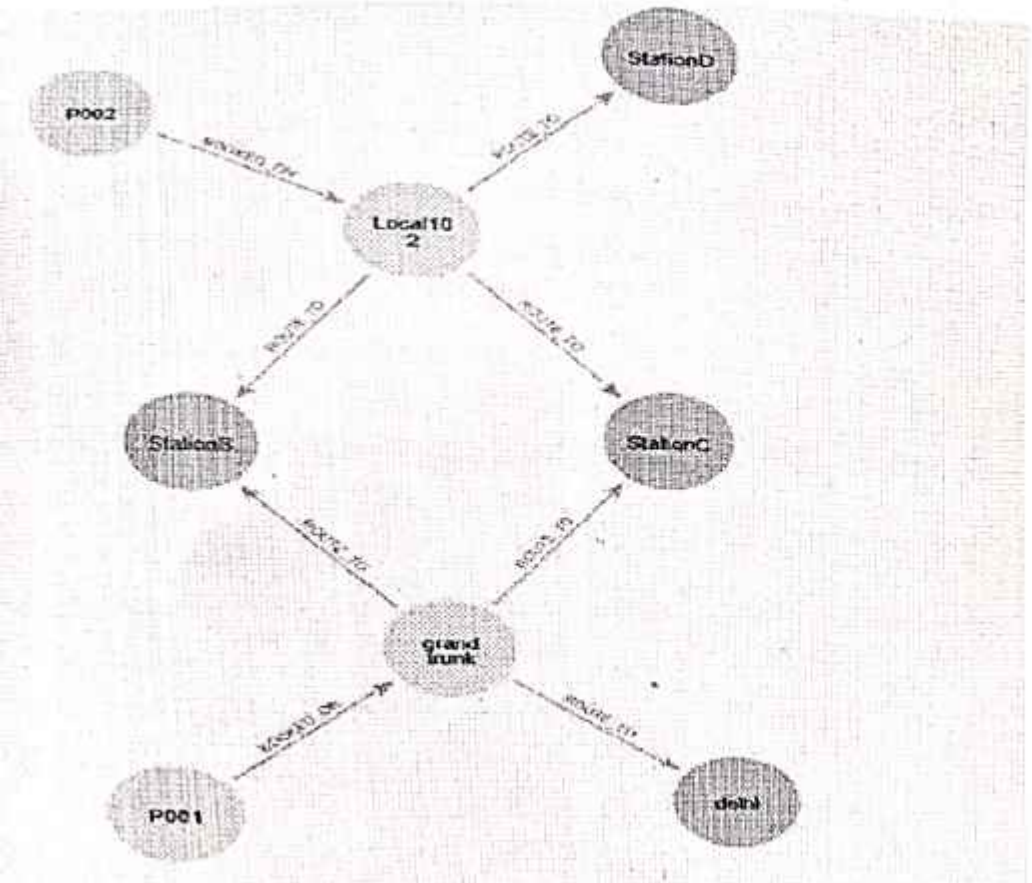
STATION NODE:

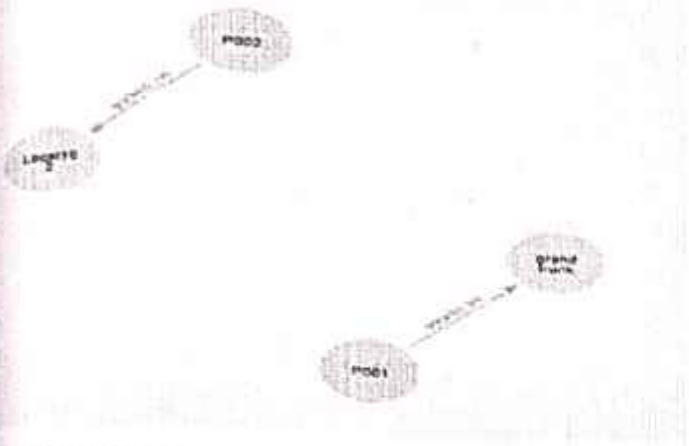
MATCH (n:Station) RETURN n LIMIT 25;



OUTES NODE:

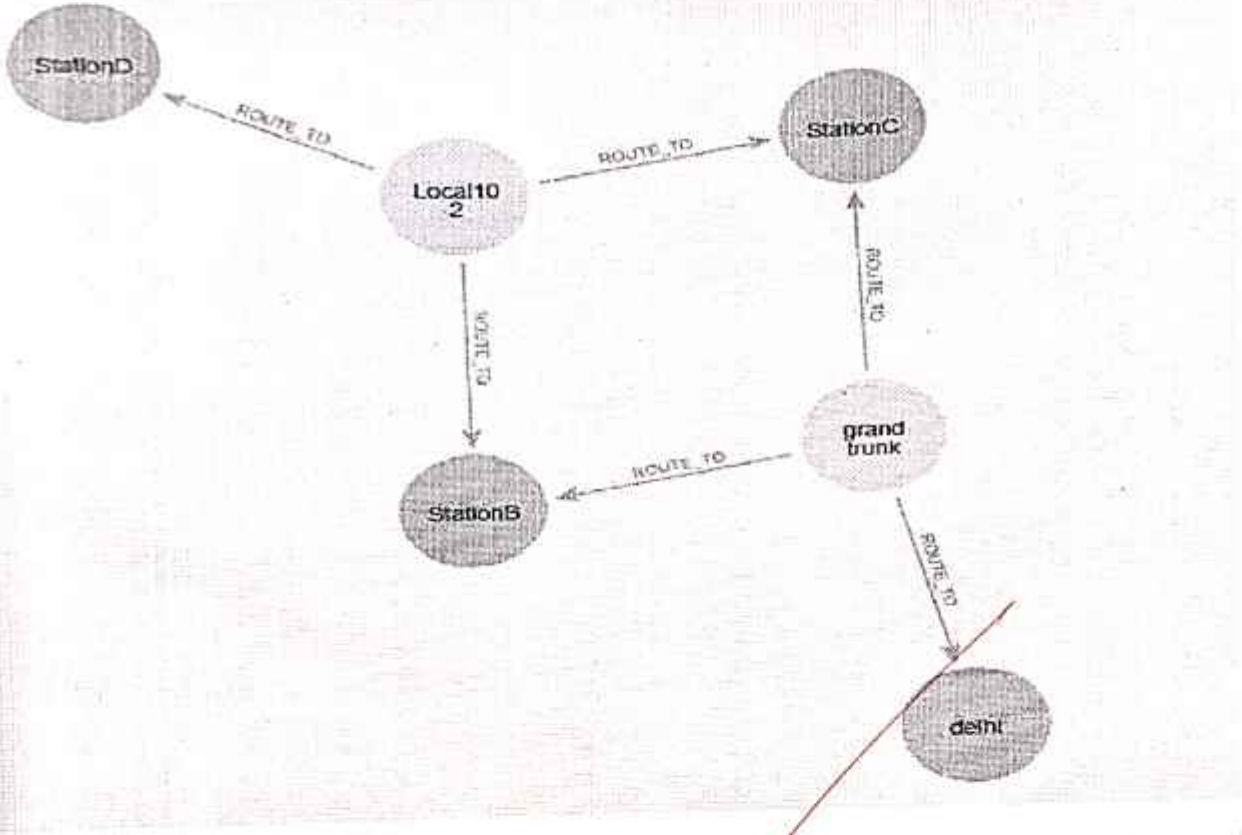
H p=()-[]->() RETURN p LIMIT 25;





ROUTE NODE:

MATCH p=()-[:ROUTE_TO]->() RETURN p LIMIT 25;



Thus,the document database and graph database by using mongodb is implemented .

Result: thus the develop a tourism management system is implemention is successfully completed

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