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: K. ASHOK REDDY

VTU NO: 30453

REG.NO: 24UEDC0031

BRANCH: CSE Data (Science)

YEAR/SEM: 2nd / III

SLOT: S15 L12

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 ?

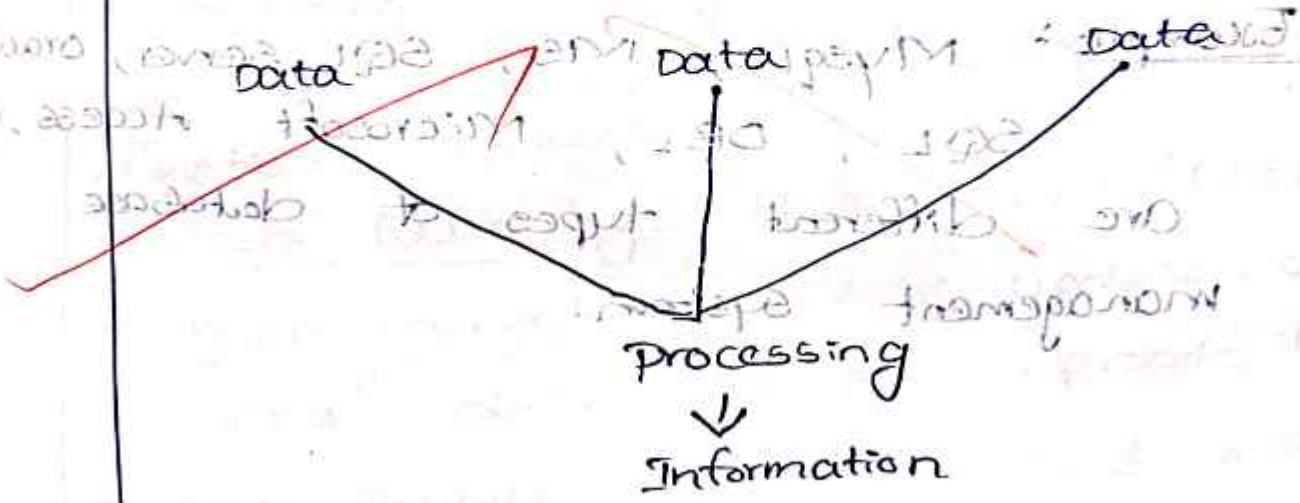
Data is a collection of a distinct small unit of information. It can be used in a variety of forms like, text numbers, media, bytes, etc. It can be stored in pieces of paper or electronic memory etc.

Examples:- Ankit, Delhi, 12, 80.

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 Ankit, City Delhi
Class 12, Marks — 80



3. What is Database ?

The data base is a collection of interrelated data which is used to retriever insert and delete the data efficiently It is also cised to organize the data in the form of a table, schema, views, and reports, etc.

Ex: The college database organizes 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 manpulating databases. It ensures data is stored seccurely, retrier efficiently data is stored accurately maintained consistently. Key feautures include data storage, integrate security, backup, recovery, and support for multiple users.

Example: Mysql, RMS, SQL Server, oracl SQL, DB2, Microsoft Access, e are different types of database management system.

5. What is ER Model?

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

* Entity: objects or things.
  [Eg. customer, order]
* Attributes: properties of entities
  [Eg. customer, ID, Name]
* Relationships: connections between entities
  [Eg. customers places order]

ER diagrams use rectangle for entities and diamonds for relationships. This model helps in organizing and structuring data for database.

6. What is SQL?

SQL (Structured Query Language) is a programming language used to manage and manipulate relational database. Key functions 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

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 Database :- Handle unstructured data. Eg: MongoDB, Redis

(2) NewSQL Database :- Combine NoSQL scalability with SQL's ACID properties Eg:- Google spanner.

(3) Distributed Database :- Spread data across multiple nodes Eg. Apache cassandra.

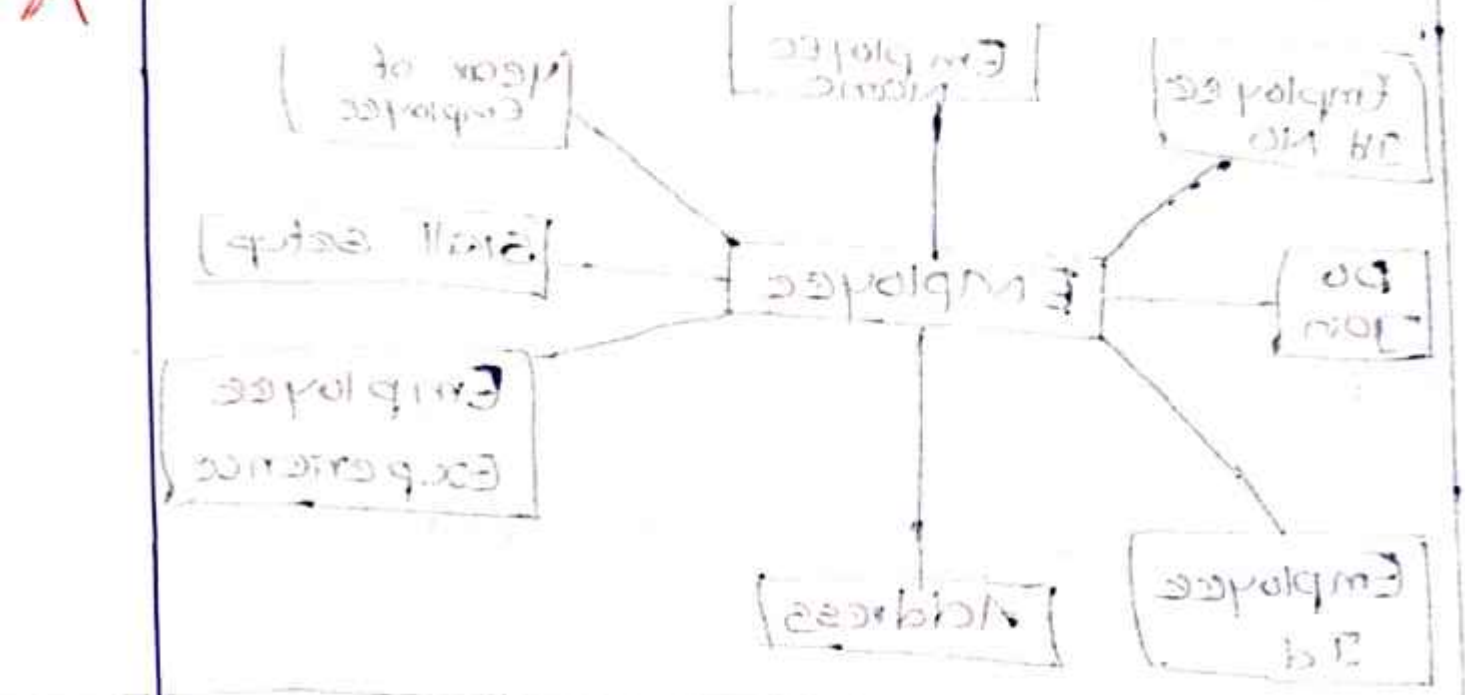
(4) Cloud Database :- Hosted on cloud platforms Eg. Amazon RDS.

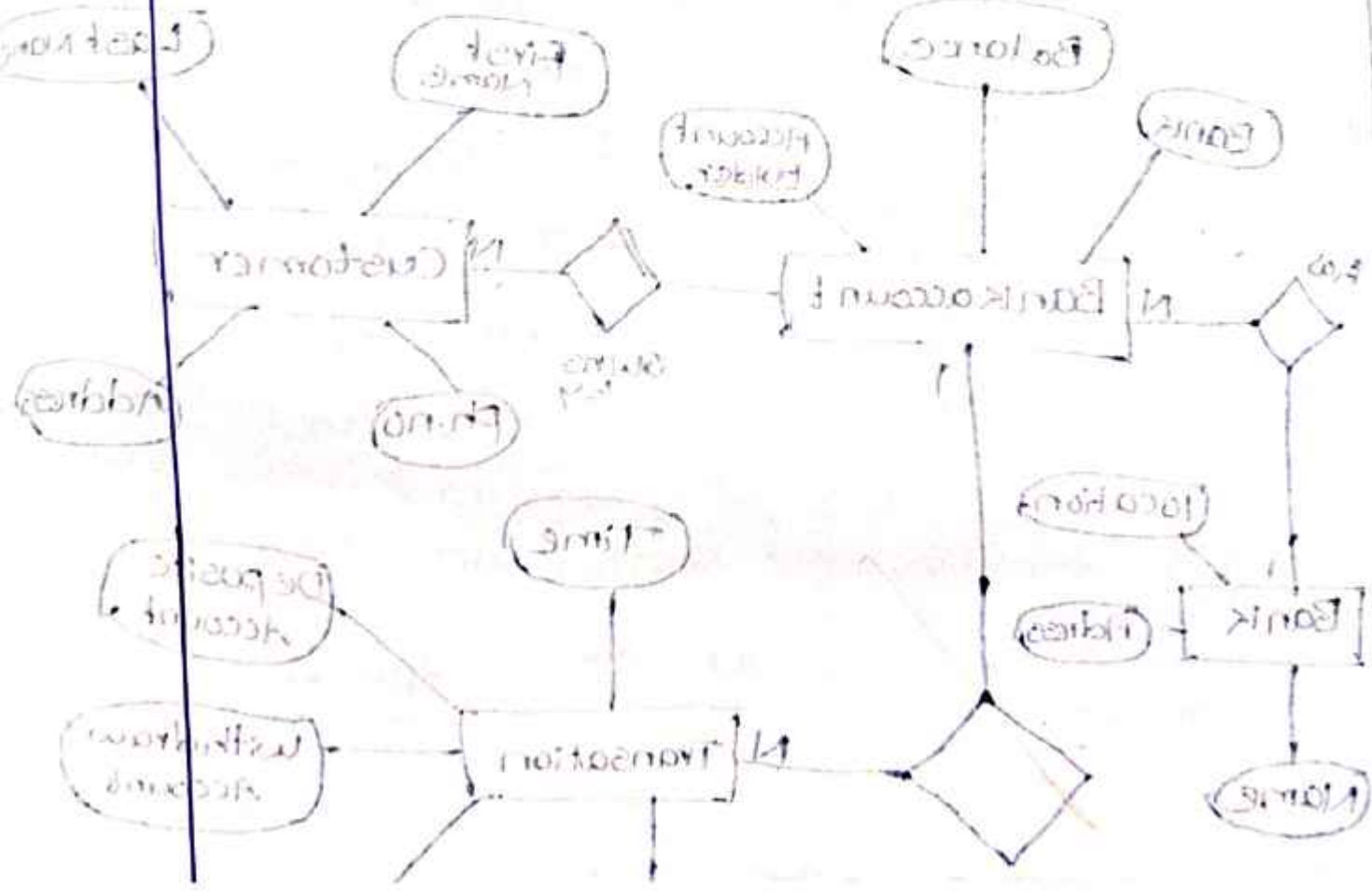
(5) In memory Database :- Store data in RAM for fast access. Eg. Redis

(6) Multi-Model Database :- Support multiple data models Eg: Arango DB.

⑦ Time series Database: optimized for time stamped data,

Eg: InfluxDB.





Task -1 :- conceptual design after FTR; Using basic database design methodology and ER modeler, design Entity Relationship Diagram by satisfying the following sub tasks:

1. a Identifying the entities
2. b Identifying the attributes
3. c Identification of relationships; Cardinality type of relationship.
4. d Reframing the relations with keys and constraints.
5. e Develop ER diagram for started case of task.

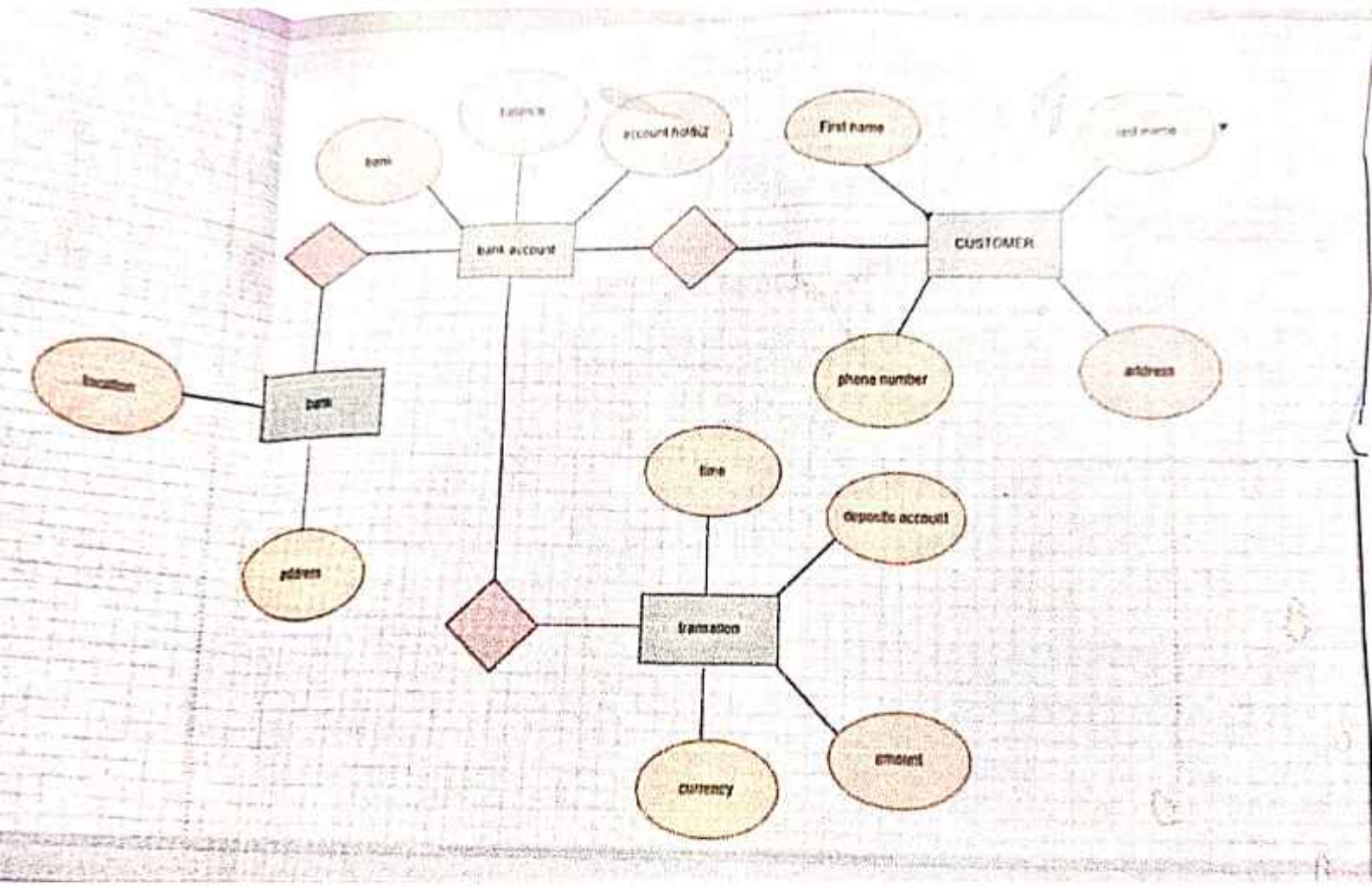
A Bank Management System is comprehensive software solution designed to manage and streamline banking operations. It covers various aspects of banking including customer account management, transaction processing, loan and mortage management and more

AIM :- To draw conceptual design through FTR using drawio tool.

Procedure :-

1. a Identifying the entities for bank management system.

Sample output


First name
bank
bank account
CUSTOMER
phone number
address
bank
address
time
deposits account
transaction
amount
currency

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 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 distinct parts. These are

* Interactive SQL: Interactive SQL is designed for create, access and manipulate data structures like tables and indexes
* PL/SQL: PL/SQL can be used to developed programs for different applications.

Oracle Forms: create a data entry screen with the suitable menu objects. Thus it is oracle forms tool that handles data gathering and data validation in a commerical application.

Report writer: Report writer allows programmers to prepare innovative reports using data form the oracle structures like tables, views etc. It is the report writer tool that handles the reporting section of commerical application.

Oracle Graphics: Some of the data can be better represented in the form of pictures. The oracle graphics tool allows programmers to prepare graphs using data from oracle structures like tables, views etc.

SQL (Structured Query language): Structured Query language is a database computer language designed for managing data in relational database management systems (RDBMS), and orignally

based upon Relational Algebra. Its scope includes data query and update, schema creation and modification, and data access control.

* IBM developed SQL in mid of 1970's
* Oracle incorporated in the year 1979
* SQL used by IBM/DB2 and DS Data base systems.
* SQL adopted as standard language for RDBS by ASNI in 1989.

DATA TYPES:

(1) CHAR(size): This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can had the maximum numbers of character is 255 characters

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

(3) NUMBER(P,S): The NUMBER data type is used to store number. Number of virtually any magnitude may be stored up to 38 digits of precision.

based upon Relational Algebra. Its scope includes data query and update, schema creation and modification, and data access control.

* IBM developed SQL in mid of 1970's
* Oracle incorporated in the year 1979
* SQL used by IBM/DB2 and DS Data base systems.
* SQL adopted as standard language for RDBS by ASNI in 1989.

DATA TYPES:

(1) CHAR(size): This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can had the maximum numbers of character is 255 characters

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

(3) NUMBER(P,S): The NUMBER data type is used to store number. Number of virtually any magnitude may be stored up to 38 digits of precision.

Number as large as 9.99 * 10^125. The precision (P) determines the number of places to the right of the decimal. If scale is omitted the default is zero. If precision is omitted, values are stored with their original precision up to the maximum of 36 digits.

④ DATE: This data type is used to represent data and time. The standard format is DD-MM-YY as in 17-SEP-2009. To enter dates other than the standard format, use the appropriate functions. Date time stores data in the 24 hours format. By default the time in a data field is 12:00:00 am, if no time portion is specified. The default data for a data field is the 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 in ASCII format.

LONG values cannot be indexed, and the normal Character functions such as SUBSTER cannot be applied.

⑥ 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 without any further conversion. RAW data type can have a maximum length of 255 bytes. LONG RAW data types can contain up to 2GB.

DATA DEFINATION LANGUAGES:

A Data Definition Language (DDL) statement are used to define the database 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 managment systems (RDBMS)

Syntax:

```
CREATE TABLE table_name
(Column_name1 data_type([Size]),
Column_name2 data_type([Size]), .....
Column_name n data_type([Size]))
```

Alter: To modify an existing database object Alter the structure of the data base

* To add a column in a table:

Syntax:

ALTER TABLE table_name ADD column_name datatype (size)

* alter table shopping customer add Review varcher(20);

* To delete column in a table

Syntax:

ALTER TABLE table_name DROP Column column_name;

* to modify a colomn in a table

Syntax:

ALTER TABLE table_name modify column 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:

TRUNCATE TABLE table_name

DATA MANIPULATION LANGUAGES

Data manipulation language 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: value can be inserted into table using insert commands. There are two types insert commands they are multiple value insert commands. Single value insert command.

Syntax:

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

(OR)

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

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

Syntax

UPDATE <table_name> SET WHERE <column1 = value>, <column2 = value>;

DELETE: This allows you to delete the particular column values using where clause condition.

Syntax:

DELETE FROM <table_name> WHERE <conditions>;

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 colums:
   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 within a range.

Syntax:
SELECT column name FROM table_name WHERE Column name BETWEEN value1 AND value2;

4. Renaming:
   The select Statement can be used to rename either a column or the entire table.

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 content of 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 substring.

_ : Matches a single character.

⑦ Select using AND, OR, NOT

We can combine one or more conditions in a SELECT statement using

the logical operators AND, OR, NOT.

Syntax:

SELECT Column name FROM table-name WHERE Condition Logical operator Condition 2.

DATA CONTROL LANGUAGES

1. create:

* create user kamal identified by kunal;

user created.

2. Grant:

* grant all privileges to kamal;

Grant succeeded

3. Revoke:

* Revoke all privideges from kamal;

Revoke succeeded.

TRANSACTION CONTROL LANGUAGES:

1. Commit:

* commit;

Commit complete.

2. Save point:

* savepoint K1;

savepoint created

3. Rollback

* rollback to K1;

Rollback complete.

SQL>
Enter user-name: system
Enter pasword:
connect.
SQL> create table ashok(name varchar(12), ph_no number(10), addhar_no number(12));

table created.

SQL> desc ashok;

Name	Null?	Type
NAME		varchar(12)
PH_NO		Number(10)
ADDHAR_NO		Number(12)

SQL> alter table student ID add gmail varchar(18);
table created.

SQL> desc student ID;

NAME	Null?	Type
NAME		varchar2(12)
PH_NO		Number(10)
ADDHAR_NO		Number(12)
Gmail		varchar(18)

SQL> alter table student ID drop column gmail;

table altered

SQL> desc student ID

NAME	Null?	Type
NAME		varchar(12)
PH_NO		Number(10)
ADDHAR_NO		Number(12)

SQL> alter table ...

Table created

SQL> desc Student10

Name	Null?	Type
Name		varchar (12)
PH_NO		Number (10)
Addhar_NO		Number (14)

SQL> alter table Student10 modify Addhar_NO number(14);

Table altered

SQL> Insert into Student10 value('ashok',77777,12345);

1 row created.

SQL> Insert into Student10 value ('guru', 66666, 54321);

1 row created.

SQL> insert into Student10 values ('sai', 55555, 67891);

1 row created.

SQL> Select * from Student 10

NAME	PH_NO	ID_NO
ashok	77777	12345
gure	66666	54321
sai	55555	67891

Result: Thus Implemention of generating design of other traditional data base modeled the program executed successfully.

VELTECH	
EX No.	
PERFORMANCE (5)	
RESULT AND ANALYSIS (5)	
VIVA VOCE (5)	
RECORD (5)	
TOTAL (20)	18
SIGN WITH DATE	

TASK: 3 :- Developing Queries with DML single-Row function and operators.

AIM: To perform the query processing on data bases for different retrieval results of queries using DML, DRL single-row operations using aggregate. data, string, ident functions, set clauses and operators.

procedure:

Create table for employee schema and insert around 10 retail-store retail store-employes data in this relation perform multi row functions -

Note: these queries assume a sample database with an "retail-store-employes" table containing columns. like "retailstore -employee-name", "Salary" and "department", "Store-phone number", "employee-phonenumber".

1. Count: Count following by a column name returns the count of tuple in that column if used with the Distinct keyword it will return only the count of unique tuple in the column otherwise, it will

column

Syntax: Count(column name)

Ex: 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 average value of that column value

Syntax: AVG (n1, n2)

Example: SELECT AVG (salary) from retail_store_employees;

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

Syntax: MAX (column name)

Ex: SELECT MAX (Salary) from retail_store_employees

SQL> select retail store_employee_name, max (Salary) from retail_store_employees. group by retail store_employee_name;

DEPTNO MAX(Salary):

SQL> Select retail store_employee_name, min (Salary) from retail_store_employees group by retail_employees_name

having min(salary)

SQL String Function:

String function 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 (retail store - employee - name)
From retail - store - employees WHERE
retail store - employee - id = 1;

2 LOWER()

Query:

SELECT LOWER (retail store - employee - nam
FROM retail - store - employees, WHERE retail
employee - id = 1;

3. LENGTH()

Query:

SELECT LENGTH (retail store - employee - name)
From retail - store - employee WHERE
retail store - employee - id = 1;

4. SUBSTR()

Query:

SELECT SUBSTR (retail store - employee - name, 1,
From retail - store - employees WHERE
retail store - employee - id = 1;

5. CONCAT ()

Query:

CONCAT (retail_store_employee_name, department)
From retail_store_employees WHERE retail_store
_employee_id = 1;

SQL Data and Time Functions

The data & time functions are built-in function in the SQL. These functions can be used in SQL queries to perform various date and time operations, such as altering and formatting dates for display purposes for storing a data and time value in a data base, MYSQL, offers the following date types:

DATE Format YYYY-MM-DD

DATETIME Format: YYYY-MM-DD HH:MI:SS

TIME STAMP Format: YYYY-MM-DD HH:MI:SS

YEAR Format: YYYY or YY

CURDATE ()

Query: SELECT CURRENT DATE from dual;

CURTIME ()

Query: SELECT CURRENT_TIME()
from dual;

SQL> SELECT ADDTIME (EXP1, EXP2);

SQL> select ADDTIME ('2018-08-01 23:59:59.999999', '11:11:00000002');

DAY of month (date)

SQL> select day of month ('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 LAST_DAY(SYSDATE) from dual;

months_between;

SQL> select months_between (SYSDATE, HIREDATE) from TEMP;

least;

SQL> select least ('10-Jan-09', '12-Oct-09') from dual;

Greatest;

SQL> select Greatest ('10-Jan-09', '12-Oct-09') from dual;

Trunc ();

SQL> SELECT TRUNC (SYSDATE, 'DAY') From DUAL;

Round;

SQL> SELECT ROUND (SYSDATE, 'DAY') From DUAL;

to_char

to_char (sysdate 'dd/mm/yy')

to date:

SQL> select to_date (sysdate, 'dd/mm
from dual;

CHARACTER FUNCTION

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

lower (char): select lower ('HELO') from du

upper (char): select upper ('hello') from d

trim (char, [set]): select rtrim ('cscit', 'cs
from dual;

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

replace (char, search): select replace
('jack and jue', 'j', 'bl') from dual;

STRING FUNCTIONS:

Concat: CONCAT returns char1
concatenated with char2. Both
char1 and char2 can be any
of the data types.

SQL> SELECT
CONCAT ('ORACLE', 'CORPORATION') from
dual;) ORACLE CORPORATION.

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 character with expr2, replicated as many time as neccessary

SQL> SELECT ('ORACLE',15,'*') From Dual;

Ltrim: Returns a character expression after removing leading blanks.

LTRIM ('SSMITH SS','S') From Dual;

Rtrim: Returns 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('DBMS') From Dual;

Upper: Returns a character expression with lowercase character data converted to uppercase

SQL> SELECT upper(... from dual);

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

SQL> SELECT LENGTH ('DATABASE') From Dual;

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

SQL> SELECT SUBSTR ('ABCDEFGHIJ', 3, 4) From DUAL;

SELECT SUBSTR ('retail store_employee_name', 1,4) From retail_store_employees;

Instr: the INSTR function searches string for substring the function returns an integer indicating the position of the character in string that is the first character of this occurrence.

SQL> SELECT INSTR('Corporate Floor',

From DUAL;

output:

create table (retail_store_employee_id number, store_id number,
(retail_store_employee varchar (34), salary number
department varchar (23), store_phone_no number);

Table created.

desc retail_store_employees;

Name	Null?	Type
HA		
Retail_store employee_id		Number
store_Id		Number
Retail_store employee		varchar(10)
Salary department		varchar(12)
store_phone		Number

Employee_phone Number

SQL> insert into retail_store_employee values (7777, 6666, 'ashok', 3053, 'hari', 7569267549
9676543219);

1 row created.

SQL> insert into retail_store_employee values
(9999, 6666, 'bunny', 12345, 'Sai', 1234567891
, 2345234523);

1 row created.

SQL> insert into retail_store_employee
values (2222, 3333, 'guru', 54321, 'Jagan'
, 7777777, 9876543213);

1 row created.

SQL> select * from retail_store_employee;

Retail_store_employee_ID	STORE_ID	Retail_STO	salary	Department	STORE_PHONE	Employee_PHONE
7777	8888	ashok	30453	hari		7569267549
9999	6666	bunny	12345	sai		1234567891
2222	3333	guru	54321	Jagan		7777777777

SQL> select count(*) from
retail_store_employee;

count(*)
3

SQL> select sum (salary) from r
retail_store_employees;

sum (salary)
97119

and/OR

```
SQL> select avg (salary) from
  retail_store_employee;
Avg (salary)
-----------
   32373

SQL> select min(salary) from
   retail_store_employee;
min (salary)
------------
       12345

SQL> select max (salary) from
   retail_store_employee;
Max (salary)
------------
       54321

SQL> select upper (department) from
  retail_store_employee where salary=30453;
upper (department)
------------------
HARI

SQL> select lower (department) from
   retail_store_employee where salary=30453;
------------------
hari

SQL> select length (department) from
   retail_store_employee where salary=30453;
Length (department)
-------------------
                  4
```

SQL> select sysdata from dual;

SYSDATE

28-Aug-25

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

NEXT_DAY(

04-sep-25

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

ADD_month

28-oct-25

SQL> select least('20-feb-07', '12-oct-07')
from dual;

LEAST('20-feb-07')

12-oct-07

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

Greatest

12-oct-07

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

ROUND(sysdate)

SQL> select sysdate, 'dd/mm/yy') from dual;

TO_CHAR
30/08/25

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

TO_DATE
30-Aug-25

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

CONCAT('ORACLE')
oraclecorporation

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

LPAD("ORACLE",
********oracle

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

LTRIM(
mithss

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

RTRIM(
ssmith

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

LOWER
dbms

SQL> select upper (DBMS) from dual;

UPPER (DBMS)

DBMS

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

LENGTH ('DBMS')

4

SQL> select substr ('dbms', 1, 3) from dual;

SUB

dbm

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

INSTR ('CORPORATE FLOOR', 'OR', 3, 2)

14

SQL> select * from retail_store_employee
where salary is null;

no rows selected

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

RETAIL_STORE_EMPLOYEE_ID STORE_ID
RETAIL_STO SALARY DEPARTMENT
STORE_PHONE EMPLOYEE_PHONE

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

: Thus, developing queries with DML, e row function and operation is cuted.

TASK 4: Developing queries with DML, Multi-row functions and operators.

perform the advanced query processing and test its heuristics using the designing of optimal correlated and nested subqueries eg such as finding summary statistics.

Consider the schema for

Employees (emp no, emp Name, department deptno, Salary, Salary, age)

orders (emp no, order id, price, qty ordqty, hand)

Itemfile (itemid, itemname, qty-ord, qty-hand, itemrate)

Queries using UNION, Intersect, MINUS.

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

SQL> select emp_no form employees;

output:

SQL> select emp_no from orders;

output:

SQL> select emp_no from employees union select emp_no from 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 employee 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 employee begining 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 deptno;

output:

Group By- HAVING: The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate function. The having calause must follow the Group By calause in a query and must also precede the ORDER By Clause if used.

SQL> Select deptno. Count(*) from employee 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 is 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 Empolyees order by Salary;

Output:

SQL> /select empno. emp_Name, salary from /employees order by salary desc;

Output:

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.

TASK-4 : Developing queries with DML multi-row functions and operators.

```
SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> desc ashok;
 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  ashok_course1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 S_ID                                               NUMBER
 C_ID                                               NUMBER

SQL> select*from ashok;

10 rows selected
```

) rows selected

)L> select*From course1;

ɔ rows selected

QL> select*from vasu course1;

o rows selected

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

. row created.

;QL> 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

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

ow created.

-> 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

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

ow created.

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

ow created.

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


rows selected
```

ow created.

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

ow created.

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

ow created.

> select*from course1;

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

> insert into vasu_course1 values(45,55);

ow created.

> insert into vasu_course1 values(25,67);

rows selected

created.

insert into vasu_course1 values(75,87);

created.

nsert into vasu_course1 values(85,90);

created.

sert into vasu_course1 values(5,10);

reated.

lect*from vasu_course1;

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

elected

55

67

87

90

10

select s_id from vasu union select c_id from course1;

S_ID

1

2

10

11

12

112

122

152

192

232

selected.

lect s_id from vasu union all select c_id from course1;

selected

S_ID

12

10

11

1

2

122

152

112

192

232

s selected.

elect s_id from vasu intersect select c_id from course1;

s selected

lect s_id from vasu minus select c_id from course1;

S_ID

1

2

selected

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

> 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

elect ph_no+s_id from vasu;

l+S_ID

3007203
3456798
123469
245659
8876555

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

s created.

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

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

selected

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

elected.

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

selected

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

_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

selected

VEL TECH	
EX NO.	4
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20

two

on

rform

sp

developing queries with DML Multi-law

tions and operations done sucessfully

using Run SQL command line

VEL TECH	
EX NO.	4
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	
TOTAL (20)	20
IGN WITH DATE	

and /OR Recursive Queries. equivalent Join queries.

title: Implementation of different types of Joins and recursive queries.

* A SQL Join combines records from two tables.
* A Join locate 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 optinal.

theory:

the SQL JOINS clause issued to combine records from two or more tables in a data base. a Join means for combining fields from two tables by using values common to each the join is actualy performed where clause which combines specified of tables

Syntax:

SELECT column 1, column2, column 3. —
from table - name 1 , table - name 2
where table - name 1 column name = table name 2 Column name)

Types of Joins :-

1. Simple Join

2. self Join
3. outer Join

Simple Join:
It is most common type of Join. it retrives 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

Example:
Select * from item cust where item-id = cust-id;
in the above statement item-id = cust.id perform Join statement retrived raws from both the tables provided they both have the same id as specified by the where clause since the where the comparision operator (=) to perform Join, it is said to be it combined the matched raw s of tables 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.

Select * from item, cust where item.id < cust id;

Table aliases

Table aliases are used to make multiple table queries shorted and more readable we given an alias name in the from class and use it instead of the name through the query.

Self Join:

Joining of a table to itself is known as self Join 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 emp x, empy where x.salary >= select avg(salary) from y. emp where x. deptno = y.deptno);

Outer Join:

it extends the result of simple join An outer join return all the rows returned by sample Join will those raws from one table do not match any row from the table the symbol (+) represent outer Join

here are the different types of Joins in SQL: (Inner) Join: returns record that having match values.

in both tables.

Select column-name(s) from table 1 Inn
Join table2 ON table column-name = table
2 column-name;

LEFT (alter) Join : return all records from
left and the matched records from the
right table.

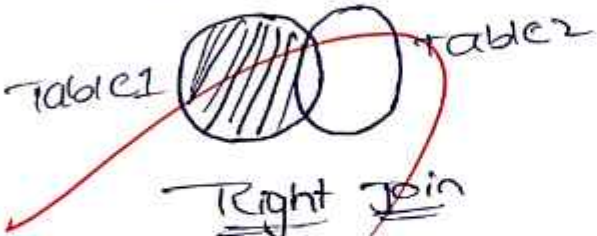
SELECT column name(s) from table left join
table 2 on table 1. column-name = table 2
column-name;

Right (Outer) Join : return all return
from right table. and the matched
records the left table.

SELECT column-name(s) FROM table 1 right
Join table 2 on table 1 column-name = table 2
column-name;

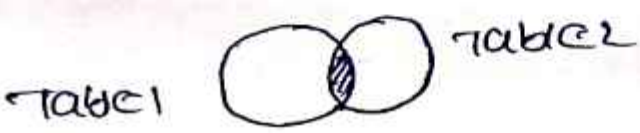
FULL outer Join 1. column-name = table
2 column-name;

LEFT Join



Right Join

Inner Join



Task-5: writing Join Queries, equivlen Recursive Queries.

```
SQL> create table dinesh(member_no number,name varchar(7));

Table created.

SQL> desc dinesh;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MEMBER_NO                                          NUMBER
 NAME                                               VARCHAR2(7)

SQL> insert into dinesh values(7,'bunny');

1 row created.

SQL> insert into dinesh values(8,'hari');

1 row created.

SQL> insert into dinesh values(9,'sai');

1 row created.

SQL> insert into dinesh values(0,'vasu');

1 row created.

SQL> select*from dinesh;

 MEMBER_NO NAME
---------- -------
         7 bunny
         8 hari
         9 sai
         0 vasu

SQL> create table jagan(member_no number,jagan_id varchar(7));

Table created.

SQL> desc jagan;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MEMBER_NO                                          NUMBER
 JAGAN_ID                                           VARCHAR2(7)

SQL> insert into jagan values(1,'xyz');

1 row created.

SQL> insert into jagan values(2,'zyx');

1 row created.

SQL> insert into jagan values(3,'zyz');

1 row created.
```

ert into jagan values(7,'zyy');

eated.

ect*from jagan;

ER_NO JAGAN_I
------
yz
yx
yz
yy

elect jagan.member_no,dinesh.name from dinesh inner join jagan on
member_no=jagan.member_no;

ER_NO NAME
------
bunny

elect jagan.member_no,dinesh.name from dinesh right outer join jagan on
.member_no=jagan.member_no;

ER_NO NAME
------
bunny

elect jagan.member_no,dinesh.name from dinesh left outer join jagan on
.member_no=jagan.member_no;

BER_NO NAME
------
bunny
vasu
hari
sai

select jagan.member_no,dinesh.name from dinesh full outer join jagan on
h.member_no=jagan.member_no;

BER_NO NAME

1
2
3
7 bunny
vasu
hari
sai

vs selected.
create table bunny(emp_id number,emp_name varchar(77));

ed.

into bunny values(7,'ashok');

ted.

t into bunny values(8,'charan');

ted.

t into bunny values(9,'siddu');

ted.

rt into bunny values(1,'siva');

ated.

ct*from bunny;

D

ME

D

ME

eate

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	

Thus, the Implementation of Set commands Joins and recursive are executed successfully

executed successfully

Consider the following two tables - number and borrowed.

inner join query :-

SELECT borrowed . membno . number

. Name . From member

Inner join borrowed.

Left Join Query :-

SELECT member name borrowed member

LEFT Join borrowed on borrowed

member = member ; mem bno ;

SQL RIGHT Join key word :-

SELECT member . Name borrowed membno

From member

RIGHT join borrowed ON borrowed

no = member membno ;

SQL FULL outer Join key word :

Tip : full outer join and full join are the same SELECT member . name borrowed membno From = member . membno ;

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

Result : Thus, the ~~implementation~~ of SQL comm using Joins and recursion queries are executed successfully :-

TASK-6: PL/SQL procedures, functions, and loops using Number theory and business scenarios.

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 oracle corporation in the early 90's to enhance the capabilities of SQL. PL/SQL is one of three key programming languages embedded in the oracle database, along with SQL itself and Java.

S.NO	Sections & Description
1	Declarations This section starts with the keyword DECLARE. It is an optional section and defines all variables, subprograms, and other elements to be used in the program.
2-	Executable commands this section is enclosed between the keywords BEGIN and END. It should have at least 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 keyword Exception. This optional section contains exception(s) that errors in the program.

Simple program to print a sentence:

Syntax

```
DECLARE
  <declarations section>
 BEGIN
  <executable command(s)>
 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
```

```
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
```

```
DECLARE
    hid number(5) := 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')
    ELSE
        dbms_output.put_line ('None of the values
        is matching');
        dbms_output.put_line ('Exact value of
            hid is: ' ||hid);
    END;
    /
```

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

successfully completed.

```
DECLARE
  hid number(1);
  oid number(1);
BEGIN
  <<outer_loop>>
  For hid IN 1..3 Loop
      <<inner_loop>>
      For oid IN 1..3 loop
        dbms_output.put_line('hid is: ' || hid || ' and
            oid is: ' || oid);
  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
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

TASK – 6: PL/SQL procedure, function

```
SQL*Plus: Release 11.2.0.2.0 Production on Thu Sep 25 14:41:38 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.

SQL> connect
Enter user-name: system
Enter password:
ERROR:
ORA-01017: invalid username/password; logon denied


SQL> connect
Enter user-name: system
Enter password:
Connected.
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 is22

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: 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.

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');
ID:101
name:raam

PL/SQL procedure successfully completed.

SQL> set serveroutput on;
SQL> exec csinformation(101,'raam');
ID:101
name:raam

PL/SQL procedure successfully completed.
```

procedure.

```
create or replace procedure cs information
(c_id in number, c_name in varchar2)
is
begin
dbms_output.put_line('ID:' ||:c_id);
dbms_output.put_line('Name:' ||:c_name);
```

procedure created

```
SQL> exec cs information (101, 'ram');
PL/SQL procedure successfully completed.

SQL> set serveroutput on;
SQL> exec cs information ('101, 'ram');
ID : 101
Name : ram
PL/SQL procedure successfully completed
```

Sample program for only function:

```
SQL> create or replace function cs information
(h_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;
```

Result) PL/SQL procedures / functions executed successfully.

ATTENDANCE	6
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	4
TOTAL (20)	19
SIGN WITH DATE	

Task-7 PL/SQL procedure for loops.

Aim: to write PL/SQL programs using loops for printing prime number customer ID's and for demonstrating loop control in diffrent scenarios.

procedure :-

1. Start a PL/SQL block or procedure
2. use a cursor (if required) to fech customer ID from a table
3. for each ID, check whether it is prime number using a loop.
4. use for loop (while loop to demonstrate prime checking
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 procedurt print_prime
customers.
cursor cust_cur IS
selecte customer_id from customer;
v_id number)
v_is prime BOOLEAN;
v_ : Number ;
begin open cust_cur;
```

```
FETCH cust_cur INTO v_id;
Exit when cust_cur%NOT FOUND;
-- prime check using while loop
   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;
     v_i := v_i + 1;
     End Loop;
  End If;
  If v_is prime then
  DBMS_output.put_line (prime Customer ID)
  : v_id
  End If;
  End loop;
  Close (cust_cor)
  END;
```

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

Example: 2 using for loop for first N prime numbers

```
create or replace procedure print_first n_prime
                                (n number) is
    v_num number := 2;
    v_count number := 0;
    v_is_prime BOOLEAN;
```

TASK - 7: PL/SQL procedure for loo

```
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
```

```
 dbms_output.put_line('given number is not an armstrong');
end if;
nd;

value for bk: 1634
2:  bk number(10):=&bk;
 2:  bk number(10):=1634;

QL procedure successfully completed.

> CREATE OR REPLACE PROCEDURE print_prime_customers IS
 CURSOR cust_cur IS
  SELECT customer_id FROM customers;  -- Table with customer IDs
 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;

    -- Prime check using WHILE loop
    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 IF;
  END LOOP;
  CLOSE cust_cur;
END;
/

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
      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;
/
```

VEL TECH	
EX NO.	7
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	

4/9/25

Loops

Aim: To write PL/SQL programs using for printing prime number Customer IDs and for demostrating loop control in different Scenarios executed successfully

```
v_is_prime := true
prime check using for loop ---
i in 2 .. trunc (SQRT (v_num)) loop
If MOD (v_num, i) = 0 then
  v_is_prime := false;
Exit;
nd If;
d Loop;
  v_is_prime then
BMS_output_line ('prime:' || v_num);
  v_count := v_count + 1;
End If;
  v_num := v_num + 1;
End Loop;
End;
EGIN
  print_first_n_prime (10); --print first
                              10 prime numbe
END;
```

EX NO.	7
PERF...	5
RESU... S (5)	5
VIVA ...	5
RECO...	5
TOTA...	20
SIGN ...	

Result)

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

11/9/25

TASK 8: Normalizing database using functional dependencies upto BCNF

(Tool: GU / TABLE Normalization Tool, MLM. 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 relational 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, Manger_ID, Hire_Date, Salary
- Department → Manager_ID
- Manager_ID → Name.

Step 2: Convert to 1NF

1. Eliminate repeating groups or arrays
2. Create separate tables for each repeating group

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.

- Create Department table: Department (department_ID, Manager_ID, Name)
- Create Employee table: Employee (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 (Manger_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

Using Griffith Tool:

1. Input relation schema and functional
2. Griffith tool generates a dependency graph
3. Analyze the graph to identify normalization issues.

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 candidates 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 candidates 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.

butes

loyee_id | name | department | job_title | manager_id | hire_date | salary

ctional Dependencies

loyee_id -> name department job_title hire_date manager_id salary

ow Steps

st, find the minimal cover of the FDs, which includes the FDs
ployee_id -> name
ployee_id -> department
ployee_id -> job_title
ployee_id -> hire_date
ployee_id -> manager_id
ployee_id -> salary
ally rel[1] is the original table:

und1: checking table rel[1]
-- The table is in 2NF already, send it to output ------

NVERT 3NF :

F to 3NF

tributes

mployee_id | name | department | job_title | manager_id | hire_date | salary

unctional Dependencies

mployee_id -> name
mployee_id -> department
mployee_id -> job_title
mployee_id -> hire_date
mployee_id -> manager_id
mployee_id -> salary

now Steps

VERT BCNF :

nalize to BCNF

utes

yee_id | name | department | job_title | manager_id | hire_date | salary

ional Dependencies

yee_id → name department job_title hire_date manager_id salary

v Steps

already in BCNF, return itself.

Griffith tool steps:

1. create a new project in Griffith
2. Define the relational schema. and FDS.
3. Run the "Dependency Graph" tool.
4. Apply transformations using the "Normalize" 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).

VEL TECH	
EX NO.	8
PERFORMANCE (5)	5
RESULT AND ANALYS'S (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	14/9/25

14/9/25

18/9/24

TASK 9: Backing up and recovery in databases

perform following backup and recovery scenario

(a) Recovering a NOARCHIVELOG database with Incremental Backups.

(b) Restoring the server parameter FILE

(c) performing Recovery with a Backup control file.

Scenario 1: Recovering a NOARCHIVELOG Database with Incremental Backups.

Step 1: Backup database.

BACKUP DATA BASE (database_name) TO DISK = incremental_backup.bak' with 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; STAT = 10

Step 3: Simulate Data Loss

Intentionally 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 [database_name] set ONLINE

Scenario 2: Restoring the server parameter File (SPFILE)

Step 1: Backup SPFILE

BACKUP SERVER parameter FILE TO FILE =
'Spfile.bak';

Step 2: Simulate SPFILE Loss

Delete or modify SPFILE.

Step 3: Restore SPFILE.

Startup MOUNT
Restore CONTROLFILE From FILE = 'controlfile.bak';
ALTER CONTROLFILE REUSE;

Step 4: Recover database.

RECOVERY DATABASE USING BACKUP CONTROLFILE

Steps: open database.

Alter Database open RESETLOGS:

SQL Server Commands

- BACK up database
- RESTORE Data base
- Recover Data base
- Alter Data base
- Backup CONTROLFILE
- Restore CONTROL FILE

VEL TECH	
EX NO.	9
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	(20)
SIGN WITH DATE	18/9/25

Result: Backing up and recover in data base perform following backup and recovery executed successfully

20/9/24

TASK 10 - CRUD OPERATIONS IN DOCUMENT DATA BASES.

AIM: To perform Mongoose using NPM design on MongoDB designing document database and performing CRUD operations like creating, inserting, querying, finding and removing operations.

Steps:

Step 1) install Mongo db using folkwing link.

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

Step 2) install Mongosh using the below link

https://www.mongodb.com/docs/Mongodb-shell/

↓ download - and install - mongosh.

Step 3) 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 Environment variables.

In the system variable section, select path and click, Edit. The Edit environment variable modal displays.

Click new and file newpath to your mongosh binary

If your PATH is configured correctly a list of valid commands displays.

Step 4) open mongo shell 4.0 from c:/program file/mongo DB/server/bin/mongod.exe.

Step 5) 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("627d13acc73990c074e397c")
}
> db.mylab.find({item:"canvas"})
{ "_id" : ObjectId("627d13acc73990c074e397c"),
"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:"mousepad", qty
:25, tags:["gel", "blue"], size:{h:19, w:22.85
cm:"cm"}}])
```

```
{
"acknowledge" : true
"insertedIds" : [
        ObjectId ('627d1598c73990c074e6397d')
        objectId ("627d1598c73990c074e6397e")
        objectId ("627d1598c73990c074e6397f")
    ]
}
> db.mylab.find ({}, item:1, qty:1)
{"_id" : objectId ("627d13acc73990c074e6397c"),
    "item" : "canvas", "qty" : 100}
{ "_id" : objectId ("627d1598c73990c074e6397d"),
    "item" : "journal", "qty" : 25 }
{ "_id" : objectId ("627d1598c73990c074e6397f"), "item
    "mousepad", "qty" : 25 }
> db.mylab.find({}, {item:1, qty:1}).pretty()
{
"_id" : objectId ("627d13acc73990c074e6397c")
"item" : "canvas",
"qty": 100
}
{
"_id" : objectId ("627d1598c73990c074e6397d"),
"item" : "journal",
"qty" : 25
}
{
"_id" : objectId ("627d1598c73990c074e6397f"),
"item" : "mousepad",
"qty" : 25
}
```

```
> db.mylab.find({item:"canvas"}).pretty().sort({item:-1})
{
"_id": ObjectId("627d13acc73990c074e6597c"),
"item": "journal",
qty:25
}
{"_id": ObjectId("627d1998c73996c74e6391");
{
"id": ObjectId("627d159ec730990c074e6391f")
"item": mousepad",
"qty": 25
"_id" ObjectId("627d13acc73990c074e697c"),
"item": "canvas",
"qty": 100,
"tags": [
  "cotton"
  ],
"size": {
"h": 28
"m": 35.5
  }
}
> db.mylab.delete one {item, journal}
= - =
>db.mylab.find({}, item, 1, qty.pretty()
  {
  "_id": ObjectId("627d13acc73990c74e637c")
  "item" = canvas
  "qty" = 25
  }
```

```
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" :
{
        "_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}
```

1

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	25/2/25

3)

1.

Result: The implementation of crud operation like creating, inserting, finding and removes operations using mongoDB is successfully executed.

9/10/24

TASK-11 : CRUD OPERATIONS IN GRAPH DATA BASES.

Aim:- to perform crud operations like creating inserting, querying, finding, delete operations 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 specific these properties suprated by commans "flower braces '{}'

Syntax :-

Following is the syntax to create node with properties.

CREATE (node label {key : value1, key2 value --})

* returning the creating node :-

To verify the creation of the node type of execute the following query in the cmd prompt match (n) Return.

* Creating relationships.

We can create a relationship using the CREATE clause. we will specify relationship with the square braces '[]' depending on the direction of the relationship it is placed

as shown the following syntax.

Syntax:-

following is the syntax to create a relation ship using the CREATE Clause.

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

* Delete a particular node:-

To Delete a particular node you need to specify the details of the node 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 (node. label. {properties ..})

DETACH DELETE node.

Create a graph database for student Course registration, create student and dept node insert value of properties.

Create (n: studat {Srd: "VTU 30455"

Sname: "Ashok"

Department name: "CSE"}

)

Output:

Added 1 label, created 1 node, Set properties. Completed after 232 ms.

output:-

added 1 label

create (n: Student {Sid : VTU3045(9)

Sname : "John"

dept name "CSE"}

Output:

added 1 label, created, node, set 2 properties completed after 72MS.

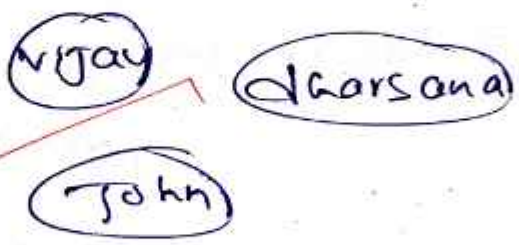
Select all nodes in your database using command

* Match(n) return(n)

output:

* match (n: student) return (n)

output:



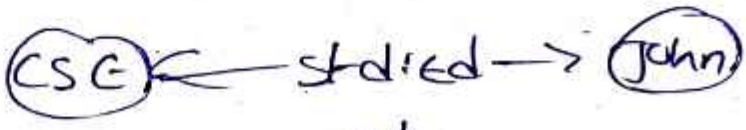
(a) create relationship b/w student and CSE match (s:student),(d: dept) where s. name vijay AND d dept = 'CSE'

Create (s) -[st: STUDENT-AT] -> (d)

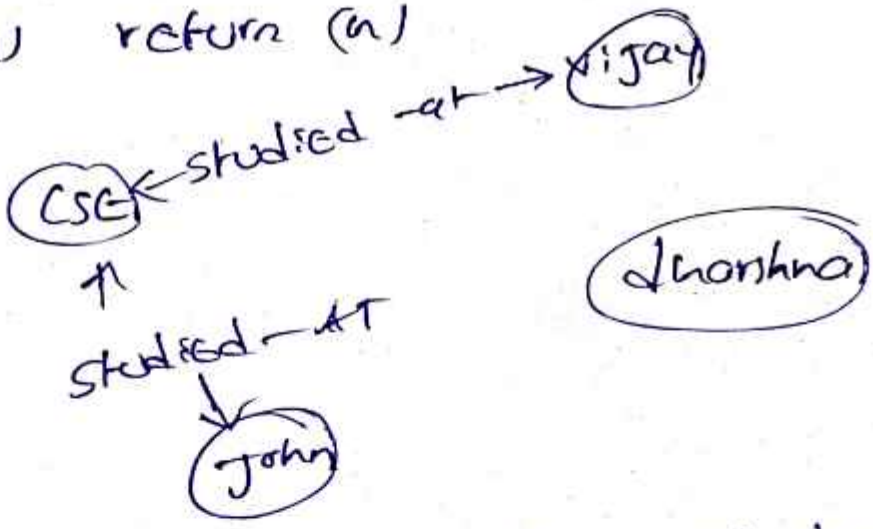
Output:

Match (s: student) (d: dept) . where s. name = 'Vijay' and d. deptname = 'CSE' CREate (s) - [st: studied - AT] -> (d).

output:



match (n) return (n)



(b) delete a node from student

match (n: student { sname: 'Dhanshna' }

DELETE (n)

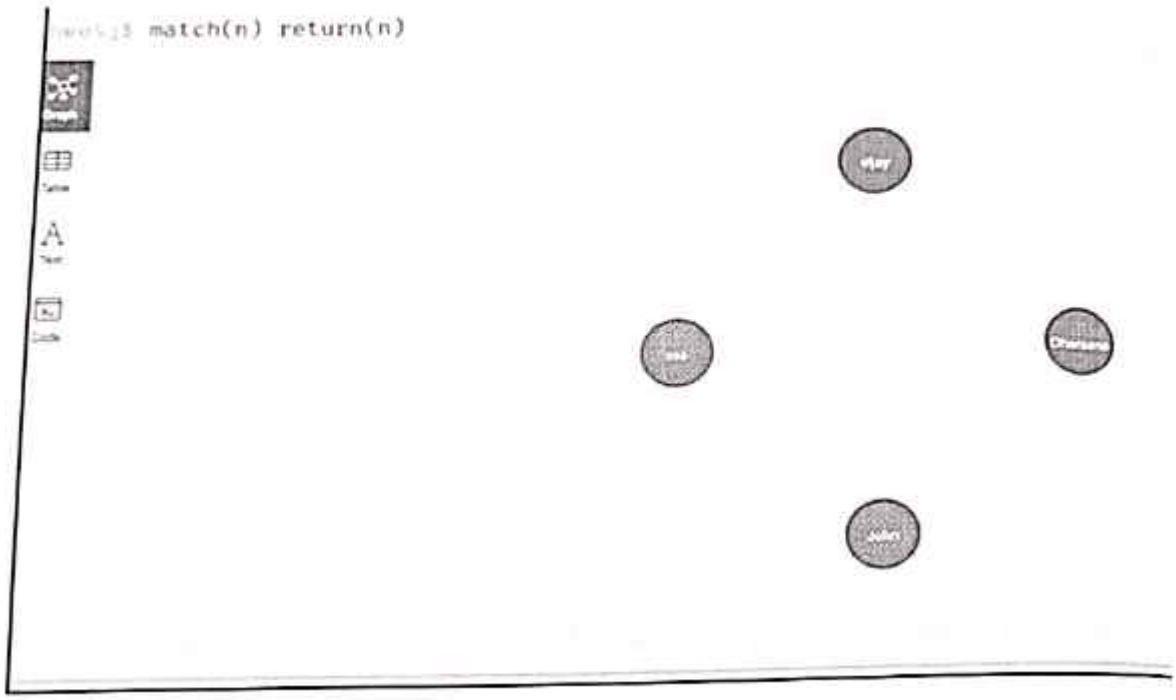
output:

Deleted 1 node, completed after

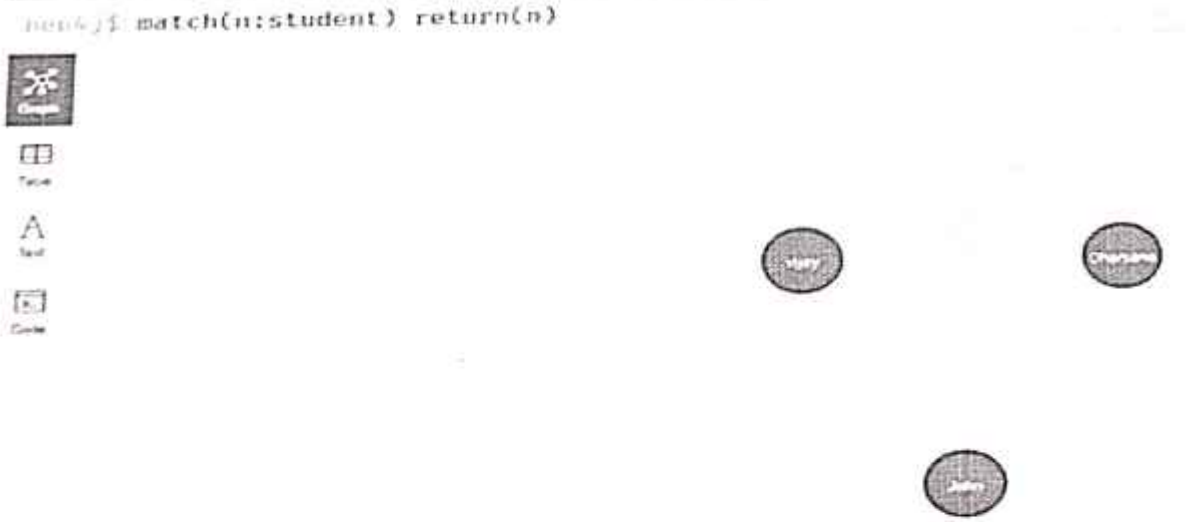
10834 ms.

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 .

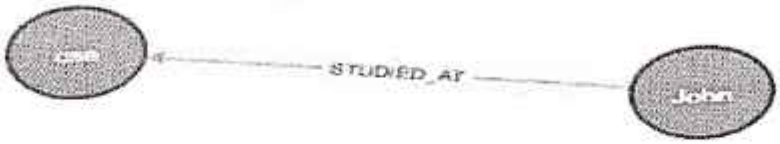
MATCH(s:student),(d:dept) WHERE s.Sname ='vijay' AND d.deptname='cse'

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

return s,d

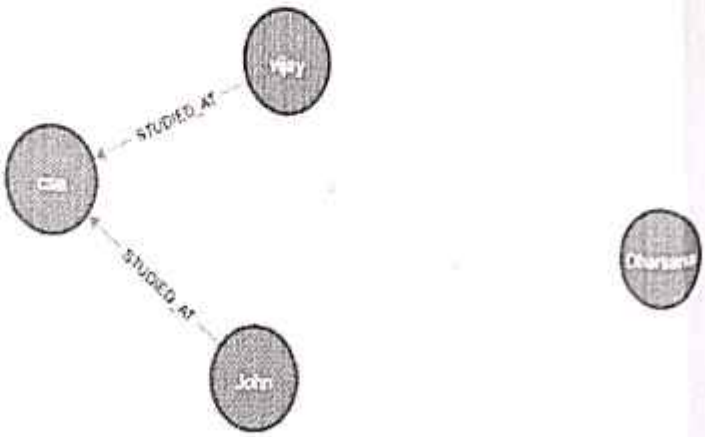
```
(s:student),(d:dept) WHERE s.Sname ='vijay' AND d.deptname='cse'
E(s)-[st:STUDIED_AT]->(d)
n s,d
```

```
(s:student),(d:dept) WHERE s.Sname ='John' AND d.deptname='cse'
E(s)-[st:STUDIED_AT]->(d)
s,d
```



) return(n)

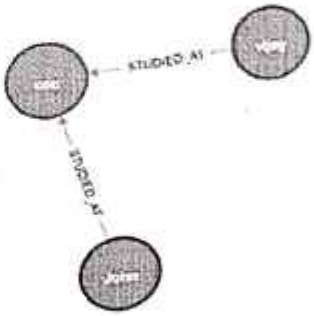
match(n) return(n)



lete a node from student

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

match(n)return(n)



VEL TECH	
EX NO.	11
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	9/10/25

Result: The, Implementation of crud operations like creating, inserting, finding and remove operations using graph DB, is successfully executed

DATABASE MANAGEMENT SYSTEMS

(10211CS207)

Tourism Management System

Team Details:

Team Leader: B.Vasu

Team Members:

Name:	vtu:	Registration no:
KASHOKREDDY	30453	24UEDC0031
P.sunil rao	30471	24UEDC0052
B.vasu	30458	24UEDC0008
P.v.satyanarayana	[illegible]	24UEDC0056
N.m.sai kishore	30480	24UEDC0037

K. ASHOK REDDY.

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 multiple 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.

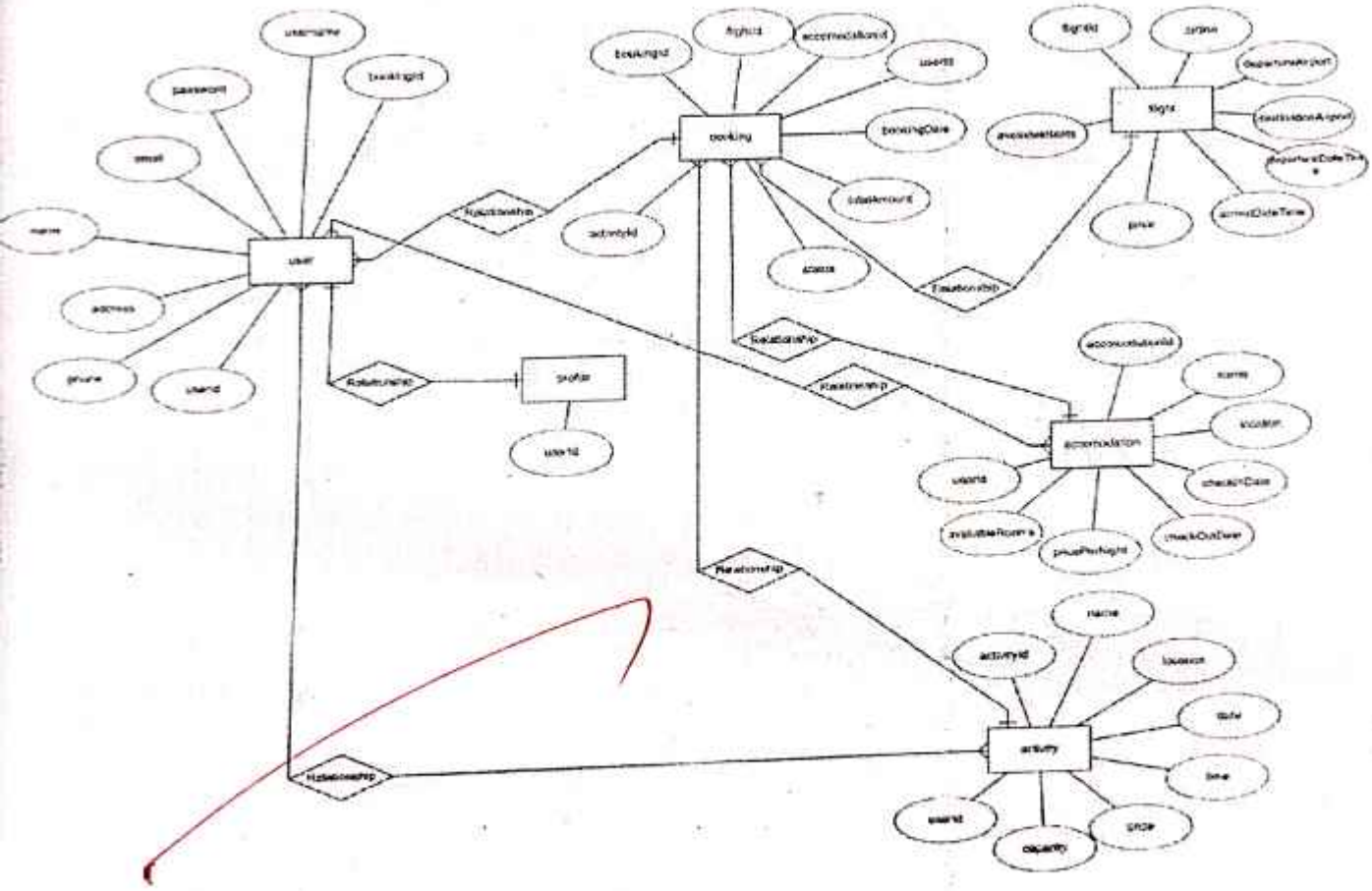
5. Booking - Accommodation Relationship (Many-to-One):

- 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:



. SQL Queries & Relational operations:

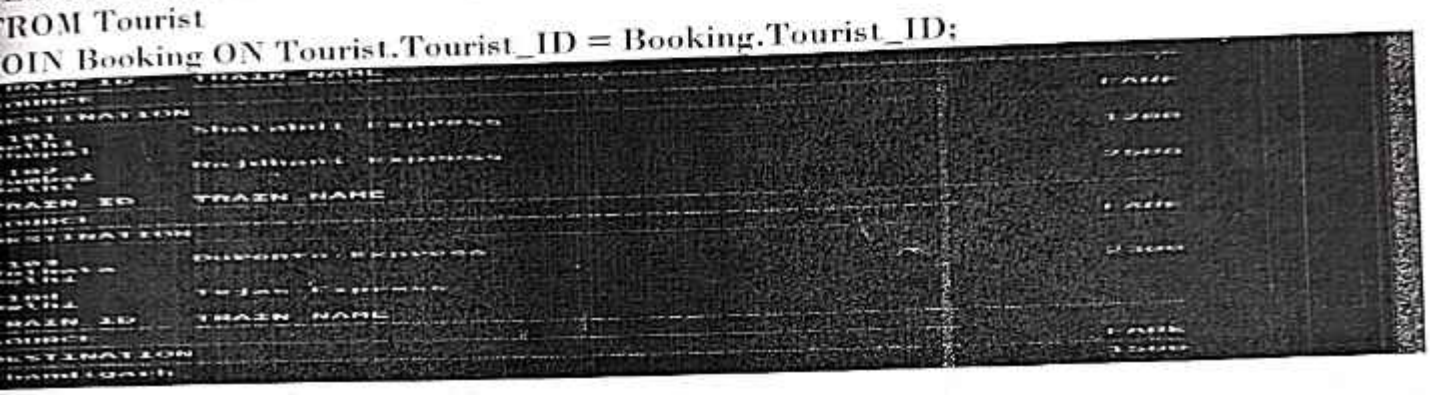
. SELECT Operation

etch all tourists who booked a package.

ELECT * FROM Tourist;

etch tourists and their booking details:

ELECT Name, Email, Booking_ID, Booking_Date
ROM Tourist
OIN Booking ON Tourist.Tourist_ID = Booking.Tourist_ID;



2. PROJECT Operation

Project → selecting specific columns)

ELECT 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
```

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
```

6. SUM Operation

Find total revenue (sum of booking amounts):

```
SELECT SUM(Amount) AS Total_Revenue FROM Booking;
```

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

OTAL_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
--------------
             3
```

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;
```

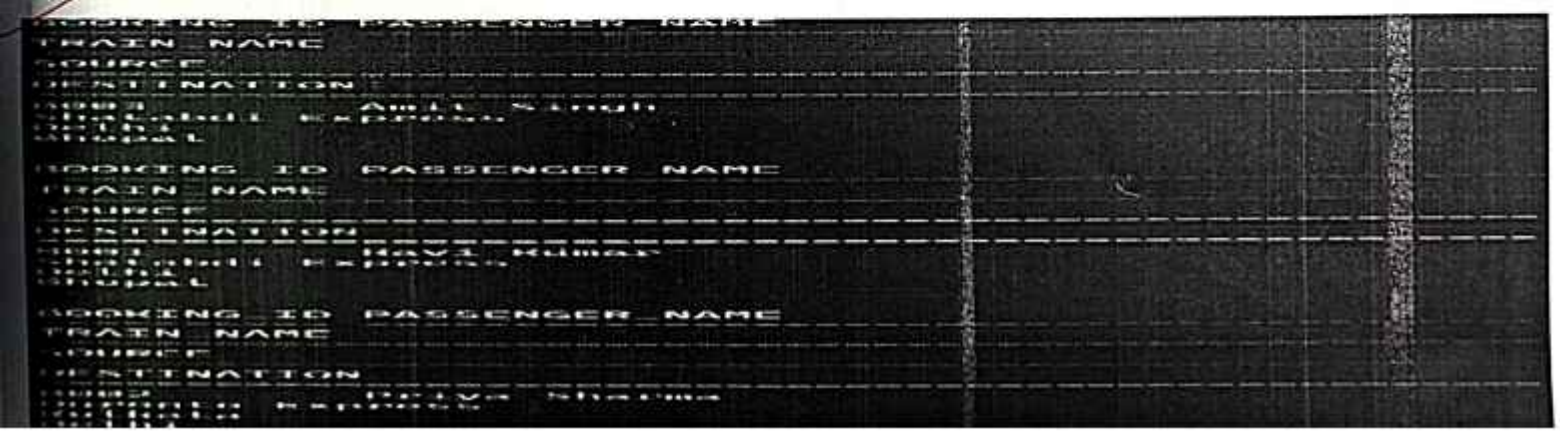
```
 AVG_FARE   MIN_FARE   MAX_FARE
---------- ---------- ----------
     1875       1200       2500
```

. JOINS

a) INNER JOIN

Show tourists and their booked package names:

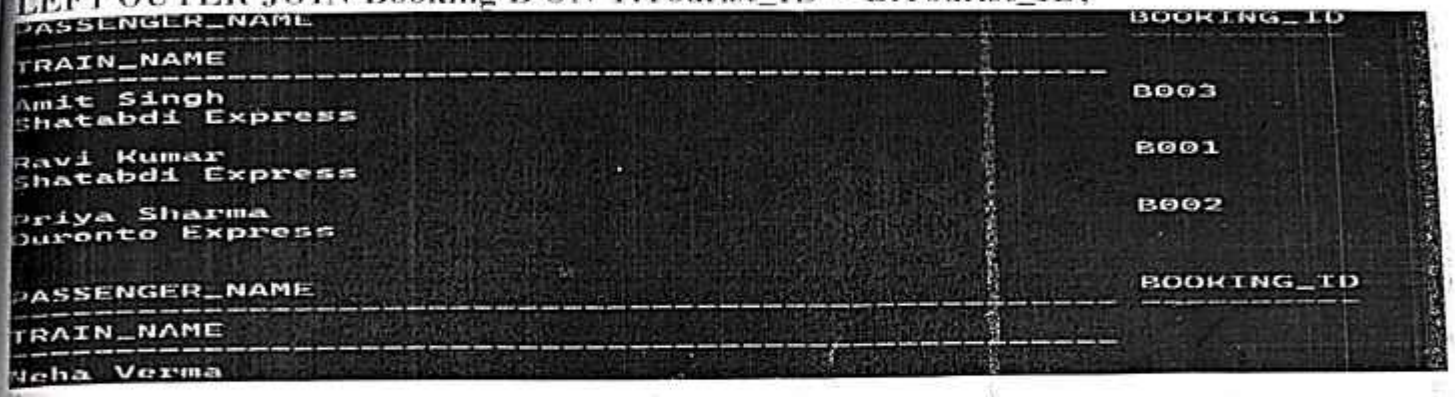
```
SELECT T.Name, P.Package_Name, B.Booking_Date
FROM Tourist T
INNER JOIN Booking B ON T.Tourist_ID = B.Tourist_ID
INNER JOIN Package P ON B.Package_ID = P.Package_ID;
```



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

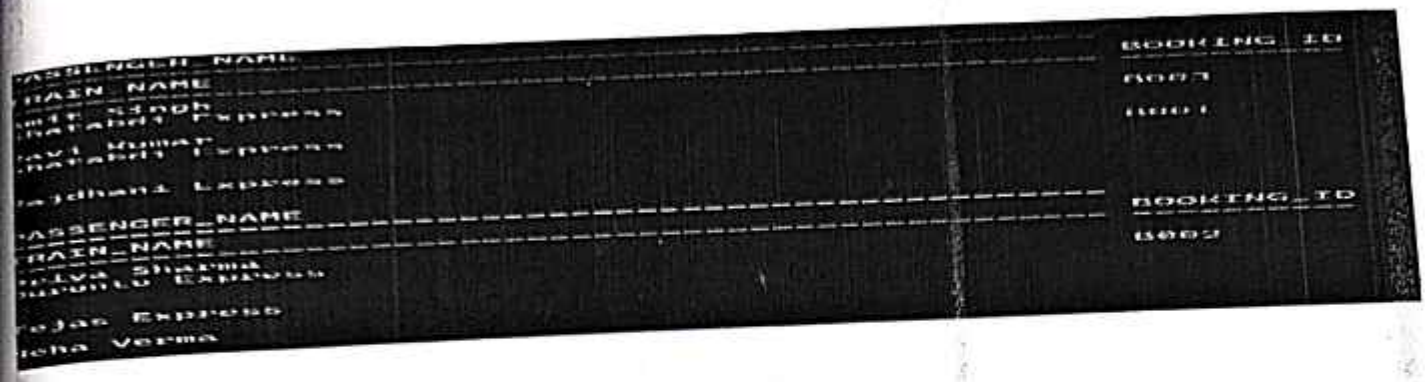
Show all bookings (even if some tourists' details are missing):

```
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

lormalize 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

NF to 3NF

Attributes

location hostel

Functional Dependencies

location → 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

Initially rel[1] is the original table with the original functional dependencies.
In each round we check the FDs one by one to see if there is a violation of 3NF (there is a partial or transitive dependency where the RHS includes non-key attributes). If yes, we decompose the table into two.

Round1: checking table rel[1]

The table is not in 3NF.
rel[2] = (hostel,traveler_name,hostel_id,room_no,phone_no,user_id,email), with FDs:
hostel --> traveler_name,hostel_id,room_no
traveler_name --> phone_no,user_id,email

rel[3] = (location,hostel), with FDs:
location --> hostel

Round2: checking table rel[2]

The table is not in 3NF.
rel[4] = (traveler_name,phone_no,user_id,email), with FDs:
traveler_name --> phone_no,user_id,email

rel[5] = (hostel,traveler_name,hostel_id,room_no), with FDs:
hostel --> traveler_name,hostel_id,room_no

Round3: checking table rel[3]

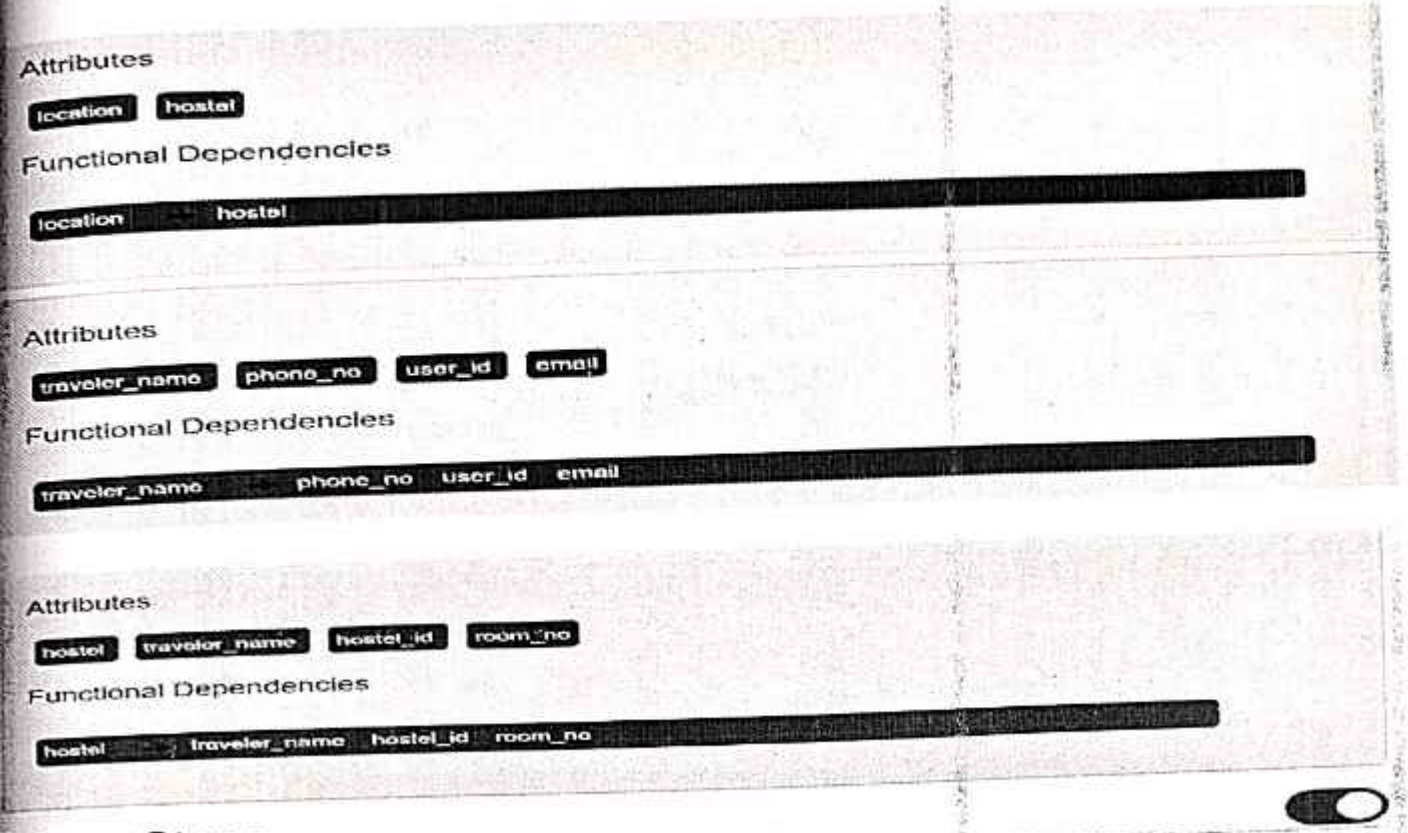
***** The table is in 3NF already, send it to output *****

Round4: checking table rel[4]

***** The table is in 3NF already, send it to output *****

Round5: checking table rel[5]

***** The table is in 3NF already, send it to output *****



Explanation:

Normalization 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.**

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

Document database using MONGODB :

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

se travel_tourism;

b.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" }

]);

```
b.payments.insertMany([
{ payment_id: "P001", booking_id: "B001", amount: 25000, method: "UPI",
ayment_date: "2025-10-10" },
{ payment_id: "P002", booking_id: "B002", amount: 22000, method: "Card",
ayment_date: "2025-10-12" }
;
b.tourists.find().pretty();
b.packages.find().pretty();
b.bookings.find({ payment_status: "Paid" });
b.bookings.aggregate([
{ $lookup: { from: "tourists", localField: "tourist_id", foreignField: "_id", as:
TouristDetails" } }
.pretty();
lb.bookings.updateOne({ booking_id: "B002" }, { $set: { payment_status: "Paid" } });
lb.packages.updateOne({ _id: 102 }, { $set: { price: 23000 } });
lb.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:

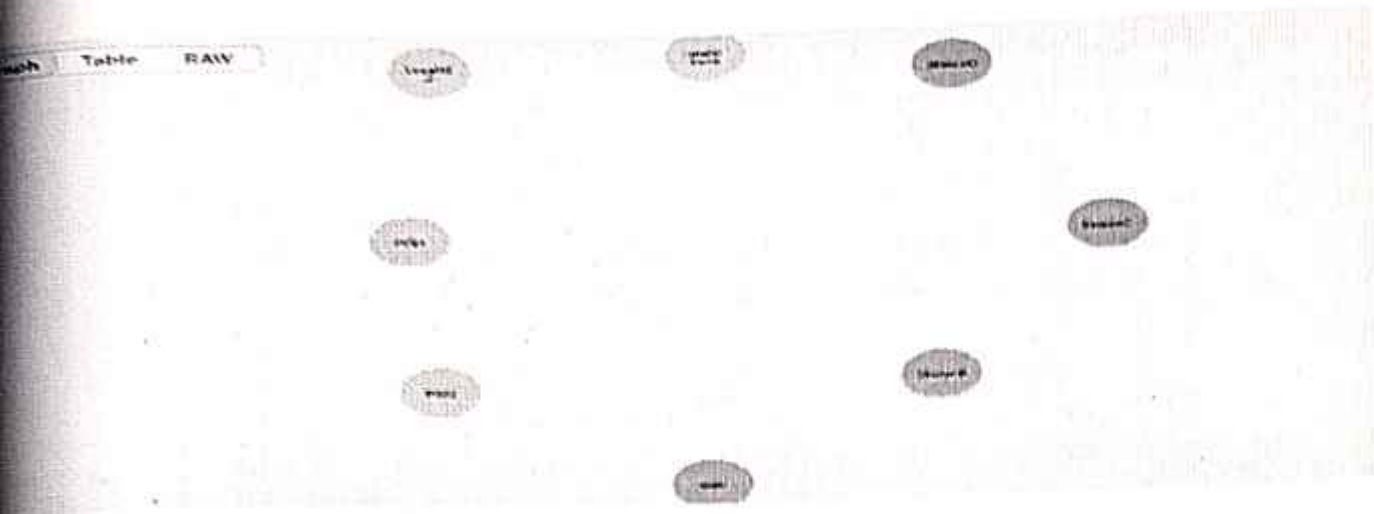
```
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("68f9cc70f219c70249bbb9ca"),
                ObjectId("68f9cc70f219c70249bbb9cb"),
                ObjectId("68f9cc70f219c70249bbb9cc")

        ]
ncaught 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,
```

```
"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:

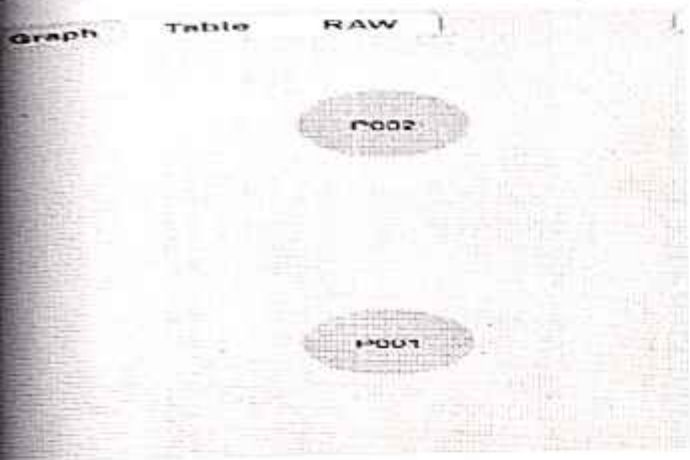
ALL NODES:

MATCH (n) RETURN n LIMIT 25;



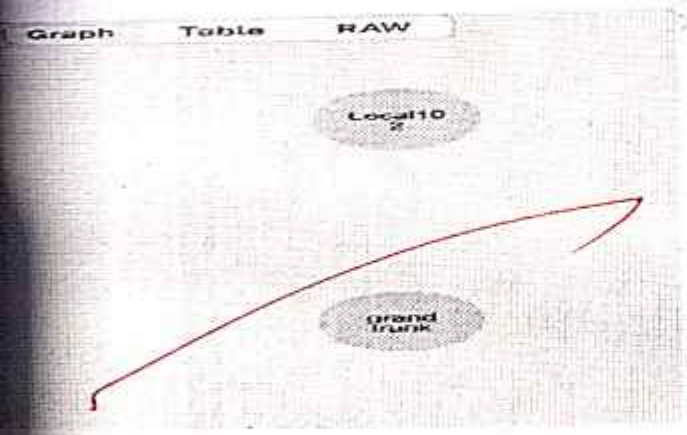
AVELER NODE:

ATCH (n:Passenger) RETURN n LIMIT 25;



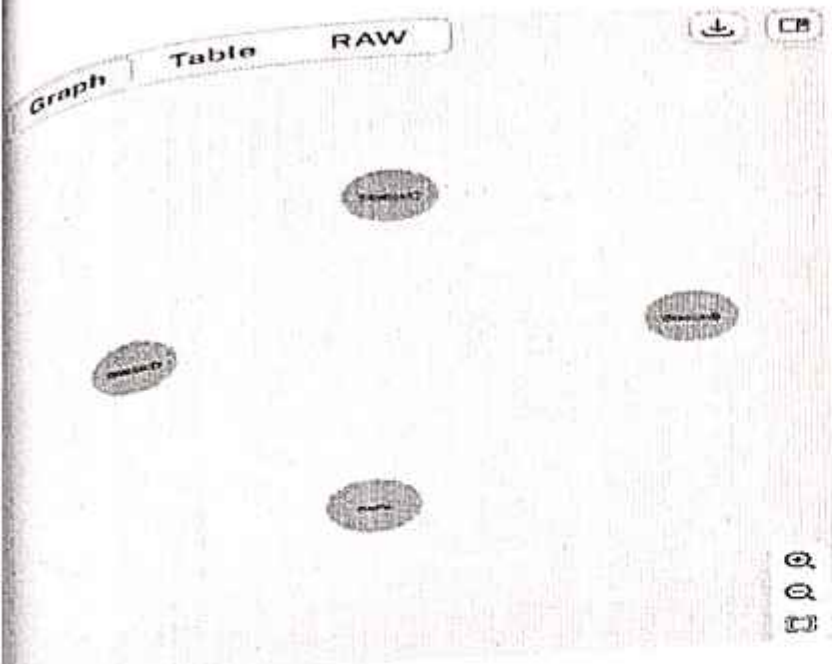
LACE NODE:

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



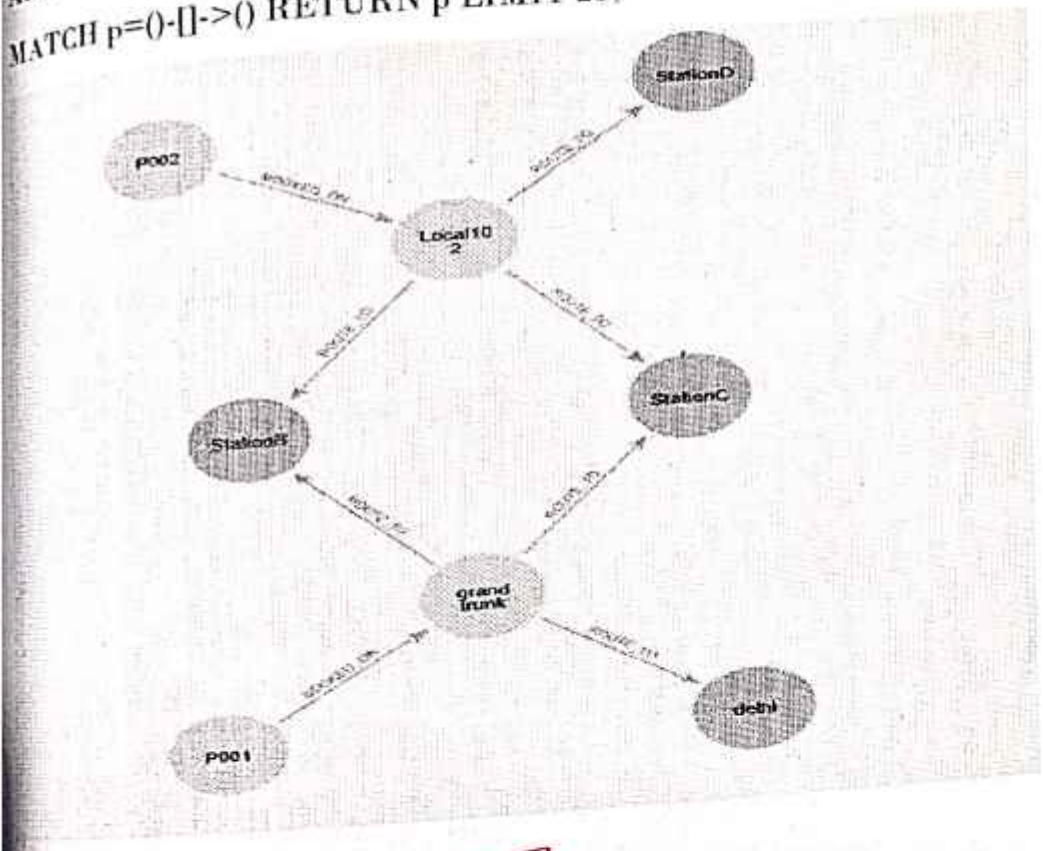
STATION NODE:

MATCH (n:Station) RETURN n LIMIT 25;



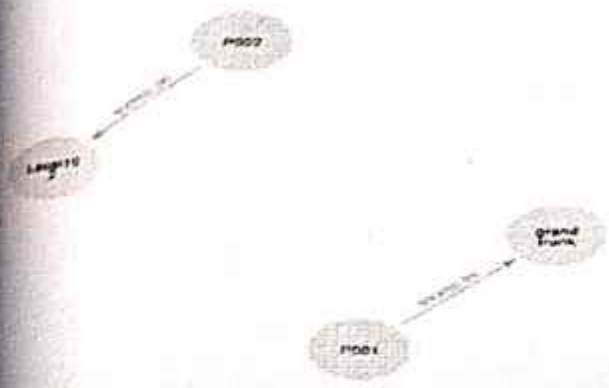
ALL ROUTES NODE:

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



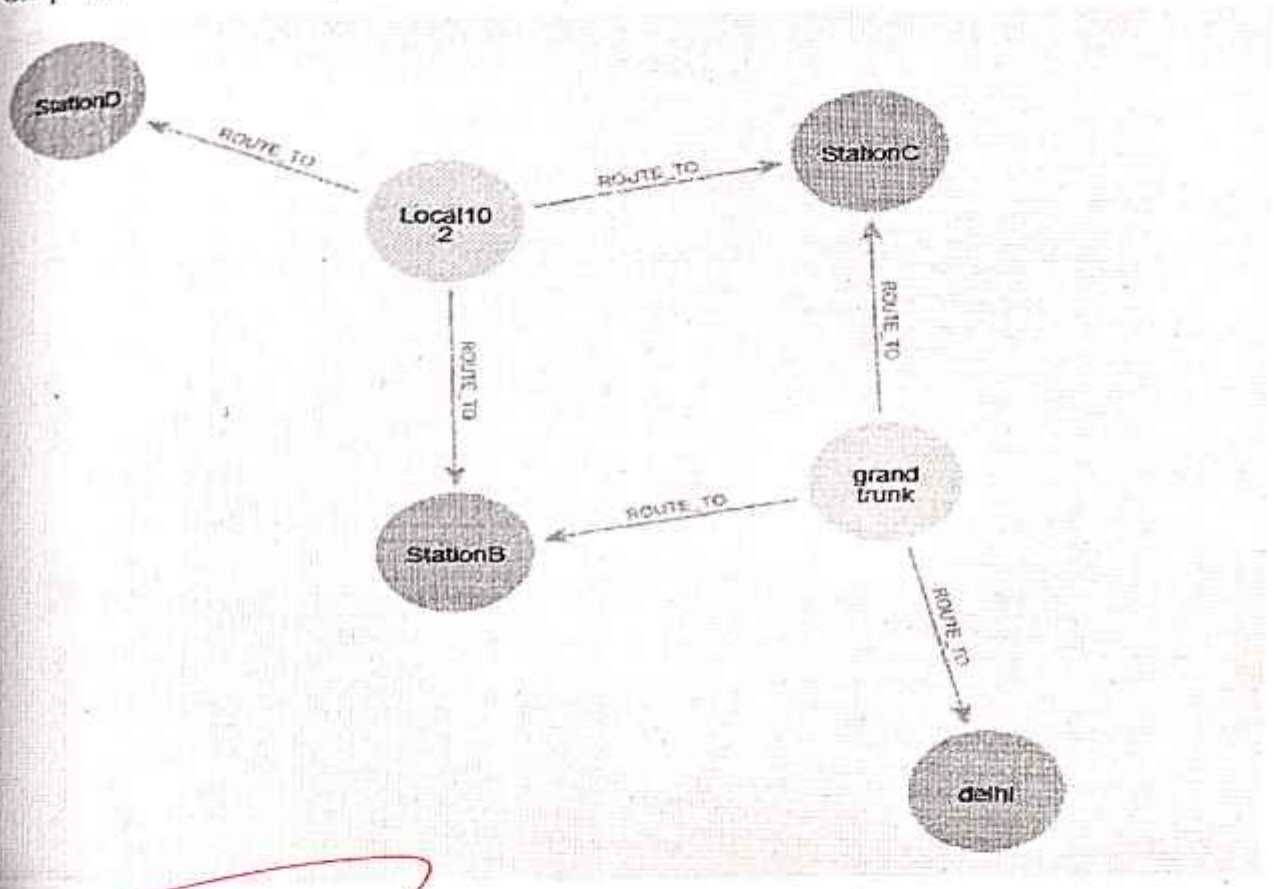
BOOKED NODE:

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



OUTE NODE:

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



us,the document database and graph database by using mongodb is plemented .

sult: thus the develop a tourism management system is plemention is successfully completed

VEL TECH	
EX NO.	12
PERFORMANCE (5)	5
RESULT AND ANALYS'S (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	23/10/24