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. Neha Joshika

VTU NO: 30499

REG.NO: 24UEDC0012

BRANCH: CSE – data science

YEAR/SEM: II nd year – 3rd sem

SLOT: S15L12

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

SIGNATURE OF FACULTY INCHARGE

SIGNATURE OF INCHARGE

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

EXAMINER 1

EXAMINER 2

Name : B. Neha Joshika

Class : 4.0.01 [Data science] Roll No : 30499

Subject : DBMS Teacher :

School/College : Meltech university

INDEX

S.No	Date	Particulars	Page No	sign	
1	24-07-2025	conceptual ER - model	20		24/7/25
2	31-07-2025	Implementation of DDL, DCL, DML and TCL commands	20		31/7/25
3	7-08-2025	developing queries with DML	20		7/8/25
4	17-08-2025	developing queries with Rows and functions	20		17/8/25
5	21-08-2025	implementation of different types of joins and Queries	20		21/8/25
6	28-08-2025	PL/SQL Procedure, function 2, loops (18/20)	18		28/8/25
7	4-09-2025	PL/SQL procedure for loops	20		4/9/25
8	11-09-2025	Normalization	20		11/9/25
9	18-09-25	Backing up and recovery	20		18/9/25
10	25-09-25	Crud operations	20		25/9/25
11	09-10-25	crud operations	20		9/10/25
12	23/10/25	Mini project	20		23/10/25

Completed.

DBMS – observation

1) what is Data?

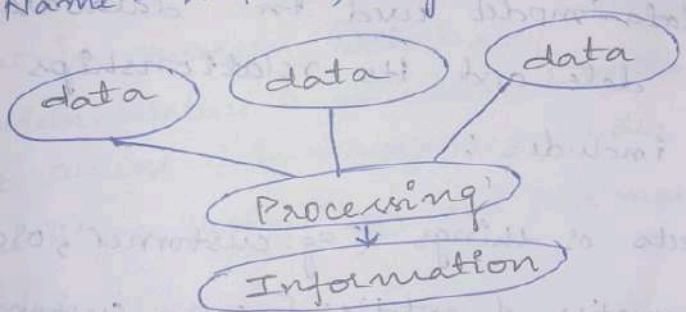
A) data is a collection of a distinct small unit of information. It can be used in a variety of forms like text, numbers, media, bytes etc.
It can be stored in pieces of paper or electronic memory etc.

Ex: Ankit, Delhi, 12, 80.

2) What is information?

A) 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?

A) 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 organizes the data about the admin, staff, students and faculty etc.

4) What is DBMS ?

A) DBMS stands for database management system, which is software for creating, managing, and manipulating databases. 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, Microsoft Access, etc. are different types of database management system

5) What is ER Model ?

A) An ER (Entity-Relationship) model is a conceptual data 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 relationships. 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 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).

→ 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) 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: combines NoSQL scalability with SQL's ACID properties, e.g., Google spanner.

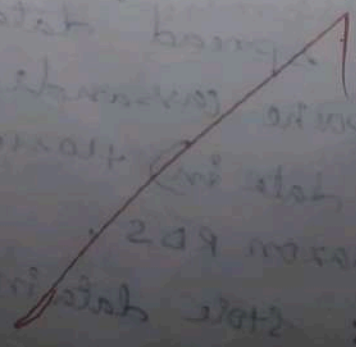
3) Distributed Databases: spread data across multiple nodes, eg., Apache cassandra

4) cloud Database: (store data in) Hosted on cloud Platforms, e.g., Amazon RDS.

5) In-Memory database: store data in RAM for fast access, e.g., Redis.

c) Multi-model Databases : support multiple data models, e.g. ArangoDB.

=> Time-series Databases : optimized for time-stamped data, e.g., Influx DB.



Task 1 : conceptual Design after FTR --

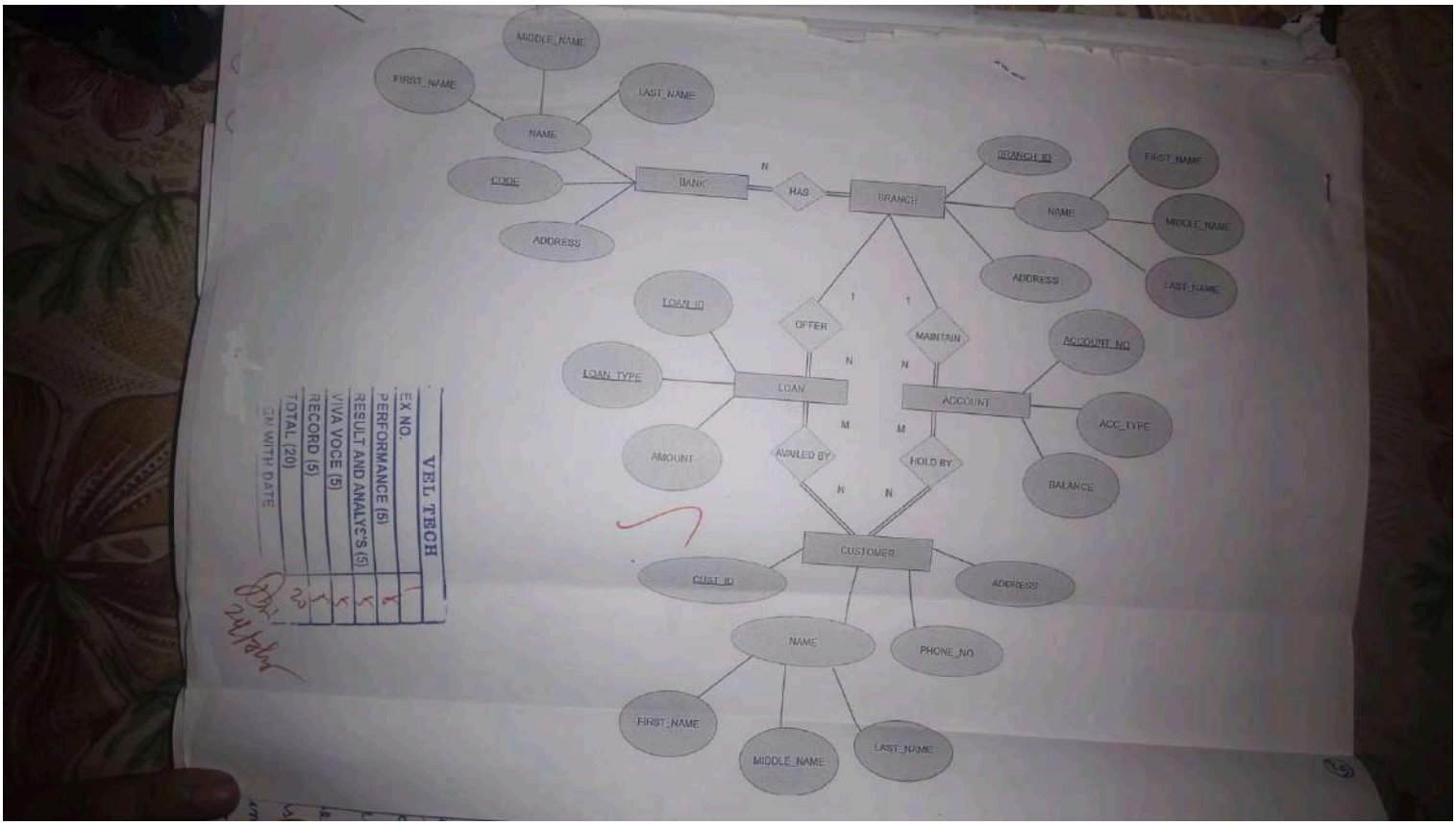
→ A bank management system is a comprehensive software solution designed to manage and streamline banking operations. It covers various aspects of banking, including customer account management, transaction processing, loan and mortgage management, and more.

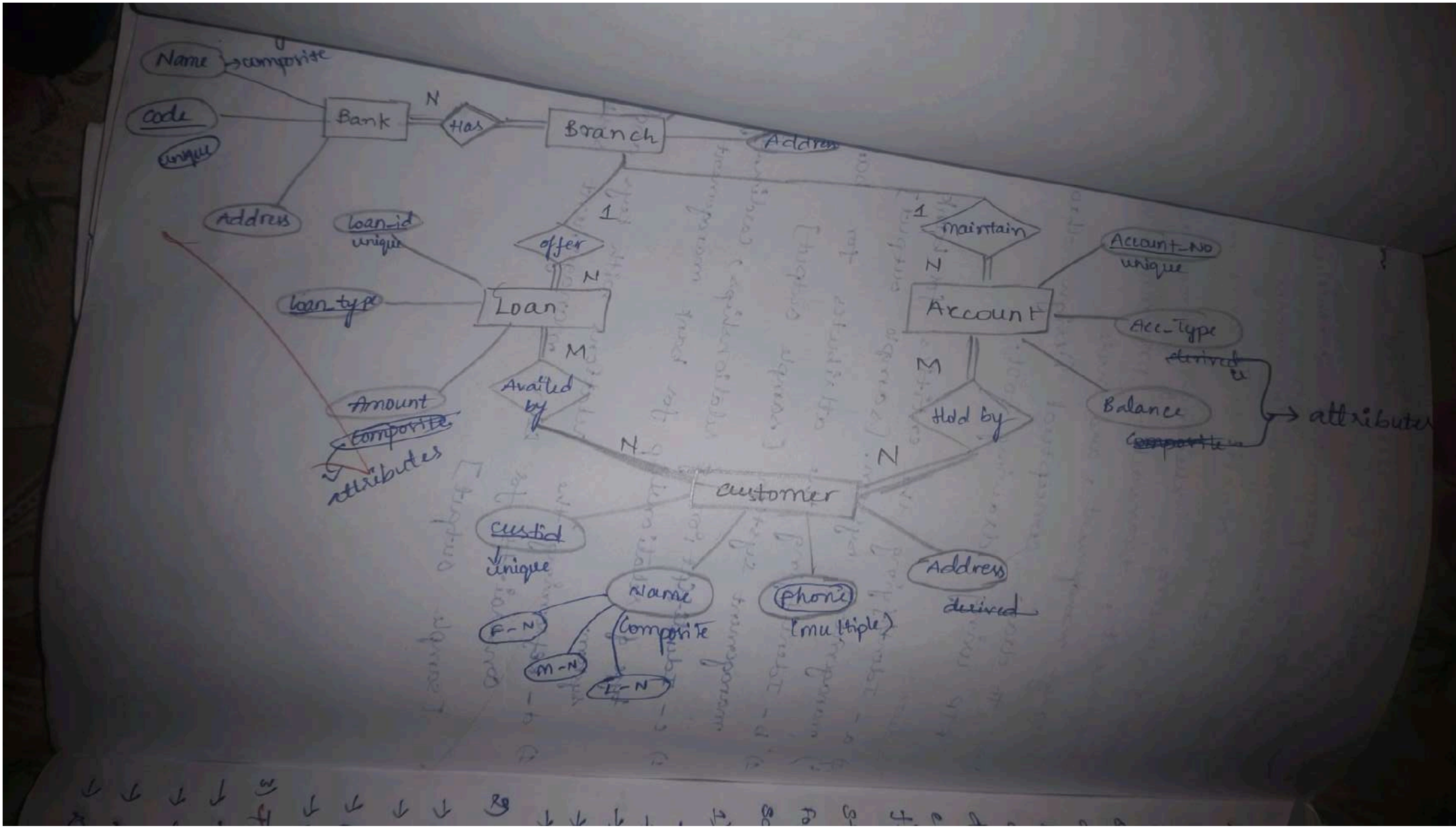
AIM : To draw - conceptual Design through FTR using drawing tool.

Procedure :

1) a - Identifying the entities for bank management system. [sample output]
2) b - Identifying the attributes for bank management system [sample output]
3) c - Identification of relationships, cardinality, type of relationship for bank management system.
4) d - Reframing the relations with keys and constraints for bank management system.

[sample output]


MIDDLE_NAME
FIRST_NAME
LAST_NAME
NAME
CODE
BANK
HAS
BRANCH
ADDRESS
BRANCH_ID
FIRST_NAME
NAME
MIDDLE_NAME
ADDRESS
LAST_NAME
LOAN_ID
OFFER
MAINTAIN
ACCOUNT_NO
LOAN_TYPE
LOAN
ACCOUNT
ACC_TYPE
AMOUNT
AVAILED BY
HOLD BY
BALANCE
CUSTOMER
CUST_ID
ADDRESS
NAME
PHONE_NO
FIRST_NAME
MIDDLE_NAME
LAST_NAME
VEL TECH
EX NO.
PERFORMANCE (5)
RESULT AND ANALYS'S (5)
VIVA VOCE (5)
RECORD (5)
TOTAL (20)
SIGN WITH DATE


Name
composite
Code
unique
Bank
N
Has
Branch
Address
Address
Loan_id
unique
1
offer
N
Loan
Loan_type
M
Availed by
Amount
Composite
attributes
N
customer
Custid
unique
Name
F-N
M-N
L-N
Composite
phone
(multiple)
Address
derived
1
maintain
N
Account
Account_NO
unique
Acc_Type
derived
Balance
Composite
attributes
M
Hold by
N

Procedure

step 1 (a): Identifying the Entities

Entities represent the major objects in the banking system.

sample output:

a) Customer
b) account
c) Transaction
d) loan
e) Branch
f) employee

step1(b): Identifying the attributes

for each entity, list out the attributes

sample output:

1) Customer
→ customer_ID (pk)
→ Name
→ Address
→ phone
→ Email

2) Account
→ Account_Number (pk)
→ Account_Type
→ Balance
→ Data_opened

3) Transaction
→ Transaction_ID (pk)
→ Transaction_Type
→ Amount
→ Date_Time

4) loan
→ loan_ID (PK)
→ Loan_Type
→ loan_Amount
→ Interest_Rate
→ start_Date

5) Branch
→ Branch_ID (pk)
→ Branch_Name
→ location

6) Employee
→ Employee_ID (pk)
→ Name
→ position
→ Contact_NO

Step 1(c) : Identification of Relationships, cardinality and Type

Relationships:

1) Customer_Account
→ A customer has one of more Accounts
→ Cardinality : 1:M
→ Type : strong relationship

2) Account - Transaction
→ An account perforns many transactions
→ cardinality: 1:M
→ Type : strong relationship

3) Custom
→ A cus
→ Cardin
→ Type

4) Branch
→ A bo
→ Cardi

5) Branc
→ A B
→ Cardi

~~Sample~~

Step 1(d):

→ Bank
→ Branch
→ loan
→ Accou

Result:

3) Customer – Loan

→ A customer takes zero or many loans

→ Cardinality : 1 : M

→ Type : strong relationship

4) Branch – Employee

→ A branch employs many employees.

→ Cardinality : 1 : M

5) Branch – Account

→ A branch manages many accounts

→ Cardinality : 1 : M

~~Sample output~~:

Step 1 (d): Reframing the Relations with keys and constraints

→ Bank (Code, Name, Address) → Code is PK

→ Branch (Branch_id, Name, Address, Code)

→ Loan (loan_id, loan_type, Amount, Branch_id)

→ Account (Account_NO, Acc_type, Balance, Branch_id)

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

Result: Thus the completion of diagram in ER-model is successful.

Task 2: Generating design of other traditional database model

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

objective:

, 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

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

Report writer: Report writer allows programmers to Prepare Innovative.

Oracle Forms: This tools allows you to create a data Entry screen along with the suitable menu objects

Oracle graphics: Some of the data can be better represented in the form of pictures.

SQL (structure Query language):

Structured Query language is a database computer language, designed for managing data in relational database management systems (RDBMS).

Data types

1) CHAR (size) : This data type is used to store character strings values of fixed length.

) VARCHAR (size) : This data type is used to store variable length alphanumeric data

) NUMBERS : The Number data types is used to store number

) DATE : This data type is used to represent date and time.

) LONG : This data type is used to store variable length character strings containing up to 2GB.

) RAW : The Raw data type is used to store binary data, such as digitized picture or image.

DATA DEFINITION LANGUAGES :

A data definition language (DDL) statement are used to define the data base structure or schema

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

ntax:

CREATE TABLE table_name
(column_name1 data_type([size]), column_name2 data_type ([size]), ...
Column_name-n data_type([size]);

reate table shoppingcustomer (SNO number(20), product no number(20), product name varchar(

Price number(20), discount number(20));

ALTER: To modify an existing database object. Alter the structure of the database

. To add a column in a table

Syntax:

ALTER TABLE table_name ADD column_name datatype(size);

. To delete a column in a Table

Syntax: ALTER TABLE table_name DROP column column_name;

. ~~alter table~~ To modify a column in a table

Syntax: ALTER TABLE table_name modify column_name date_type([new_size]);

Data Manipulation Languages:

Data manipulation languages allows the users to query and manipulate data in existing schema in object

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

Syntax: DELETE FROM <table_name> WHERE <condition>;

Insert: values can be inserted into table using insert commands

Syntax: INSERT INTO Table_name VALUES (value1, value2, value3, ...)

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

Syntax:

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

SELECT: The select statement is used to query a database.

5) SORTING: The select statement with the order by clause is used to sort the contents

Syntax: Table either in ascending or descending order

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

```
SELECT column name FROM table_name
WHERE column name LIKE "%a or _";
```

Data control languages:

1) CREATE:

```
Create user kamal identified by
        kunal;
```

user created

2) Grant:

```
Grant all privileges to kamal;
```

Grant succeeded

5) Revoke :

Revoke all privileges from kamal;

Revoke succeeded.

Transaction control language :

1) commit :

• commit;

commit complete

2) savepoint :

• savepoint k1;

save point created

3) Rollback :

• Rollback to k1;

Roll back complete

Output :

create table employee (name varchar(90), VTUNO number(20), address varchar(70));

Name	Type
NAME	VARCHAR (20)
IDNO	NUMBER (6)
ADDRESS	VARCHAR (45)

ALTER TABLE employee add phone number(10);

Name	Type
NAME	VARCHAR2(20)
IDNO	NUMBER (6)
PHONE	VARCHAR2(45)
ADDRESS	NUMBER (10)

ALTER TABLE employee rename column PHONE TO PHONE NO;

Name	Type
NAME	VARCHAR(20)
IDNO	NUMBER(5)
ADDRESS	VARCHAR(45)
PHONE NO	NUMBER(10)

Insert into employee values ('djknewd', 57780, 'jewgjwefm', 7665424669);

Name	Type
NAME	VARCHAR2(20)
IDNO	NUMBER(5)
ADDRESS	VARCHAR(45)
PHONE NO	NUMBER(10)

NAME	IDNO	ADDRESS	PHONE NO
djknewd	57780	jewgjwefm	7665424669

Result: This the execution of DDL, DML, TCL, DCL is successfully completed.

VEL TECH	
EX NO.	2
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	31/7/25

Result :- Thus the execution of DDL, DML, TCL, DCL is successfully completed.

Task 2

```
SQL> create table geetha(name varchar(25),idno number(6),address varchar(30));
Table created.
SQL> insert into geetha values('neha',638646,'guntur');
1 row created.
SQL> select *from geetha;

NAME                      IDNO ADDRESS
------------------------- ---- ------------------------------
neha                    638646 guntur
SQL> select address from geetha;

ADDRESS
------------------------------
guntur
SQL> select distinct idno from geetha;

    IDNO
--------
  638646
SQL> select idno from geetha where idno between 134526 and 638647;

    IDNO
--------
  638646
SQL> select name as beautifulname from geetha;

BEAUTIFULNAME
-------------------------
neha
SQL> delete from geetha where idno = 638646;
1 row deleted.


SQL> desc geetha

Name                      Null?    Type
------------------------- -------- ----------------------------
NAME                               VARCHAR2(25)
IDNO                               NUMBER(6)
ADDRESS                            VARCHAR2(30)
select idno from geetha where name like'_f%';
no rows selected
SQL> select name,idno from geetha order by address;
no rows selected
SQL> select idno,address from geetha order by name;
no rows selected
SQL> truncate table geetha;
Table truncated.
```

(7)

```
Connected.
SQL> create table babu (name varchar(10),vtuno number(5),address varchar(

Table created.

SQL> desc babu
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 NAME                                               VARCHAR2(10)
 VTUNO                                              NUMBER(5)
 ADDRESS                                            VARCHAR2(14)

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

Table altered.

SQL> desc babu
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 NAME                                               VARCHAR2(10)
 VTUNO                                              NUMBER(5)
 ADDRESS                                            VARCHAR2(14)
 PHONENUMBER                                        NUMBER(10)

SQL> alter table babu drop column phonenumber;

Table altered.

SQL> desc babu
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 NAME                                               VARCHAR2(10)
 VTUNO                                              NUMBER(5)
 ADDRESS                                            VARCHAR2(14)

SQL> alter table babu modify address varchar(permanent address);
alter table babu modify address varchar(permanent address)
                                        *

ERROR at line 1:
ORA-00910: specified length too long for its datatype


SQL> alter table babu modify address varchar(15);

Table altered.

SQL> desc babu
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 NAME                                               VARCHAR2(10)
 VTUNO                                              NUMBER(5)
 ADDRESS                                            VARCHAR2(15)
```

```
SQL> alter table babu rename column address to addresses;

Table altered.

SQL> desc
Usage: DESCRIBE [schema.]object[@db_link]
SQL> desc babu
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 NAME                                               VARCHAR2(10)
 VTUNO                                              NUMBER(5)
 ADDRESSES                                          VARCHAR2(15)

SQL> nsert into babu values ('jagan',29203,'kanaganapalli');
SP2-0734: unknown command beginning "nsert into..." - rest of line ignored.
SQL> desc babu
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 NAME                                               VARCHAR2(10)
 VTUNO                                              NUMBER(5)
 ADDRESSES                                          VARCHAR2(15)
```

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

31/8/22

Task3: Developing Queries with DML single Row function and operations

Aim: To perform the query processing on database for different retrieval result of queries using DML, DRL, single Row operations using aggregate, date, string, indent functions set classes and operators.

Procedure:
Create table for employee schema and insert around to retail-store_retail_store_employees data in this relation perform multi row function

1) Count: Count following by a column name returns the count of tuple in that column otherwise it will return count of all the tuples count (*) indicates all the tuples of the 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 names)

Ex: SELECT SUM (salary) FROM retail_stor-employee;

4) AVG: AVG followed by a column name returns the average value of that column values

Syntax: AVG (n1, n2 ...)

Ex: SELECT AVG (salary) FROM retail_store_employees.

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

Syntax: MAX (column name)

Ex: SELECT MAX (salary) FROM retail_store_employee;

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

DEPTNO MAX (salary)

SQL> select_retailstore_employee_name, max (salary) from retail_store_employees group by retailstore_employee_name having max (salary) < 3000;

5) MIN: MIN followed by column name returns the maximum value of that column

SQL string functions:

string functions are used to perform an operation on input string and return an output string following are the string functions defined 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_name) FROM retail_store_employees WHERE retailstore_employee_id = 1;

3) LENGTH()

Query: SELECT LENGTH(retailstore_employee_name) FROM retail_store_employees WHERE retailstore_employee_id = 1;

4) SUBSTR()

Query: SELECT SUBSTR(retail_store_employee_name 1,5) FROM retail_store_employees WHERE retail_store_employee_id = 1;

5) CONCAT()

Query: SELECT CONCAT(retail_store_employee_name, ' department') FROM retail_store

employees WHERE retail store - employee_id = 1);

SQL Data and time functions :

Date format YYYY - MM - DD

Date time format YYYY - MM - DD HH : MI : SS

TIMESTAMP format YYYY - MM - DD HH : MI : SS

YEAR format YYYY or YY

CURDATE ()

Query : SELECT CURRENT Date from dual;

CURTIME ()

Query : SELECT CURENT_TIME () from dual;

ADD DATE (DATE, DAYS)

SQL > SELECT ADD DATE ('2018-08-01', 31);

ADD TIME (exp 1, exp 2)

sql > SELECT (ADD TIME ('2018-08-01 23:59:
999999');

DAY OF MONTH (date)

sql > SELECT DAY OF MONTH ('2018-02-15');

DAY of WEEK (date)

sql > SELECT DAY of WEEK ('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

add_months

SQL > SELECT ADD_MONTHS (SYSDATE, 2) FROM

last_day :

SQL > SELECT LAST_DAY (SYSDATE) FROM DUAL;

months_between;

SQL > SELECT MONTHS_BETWEEN (SYSDATE,
FROM

least :

SQL > SELECT LEAST ('10-JAN-07', 12-OCT-07)

Greatest

SQL > SELECT GREATEST ('10-JAN-07', '12-OCT-
FROM DUAL;

String FUNCTIONS :

Lpad : LPAD returns expr 1, left padded to length n characters with the sequence of characters in expr 2.

SQL > SELECT LPAD ('ORACLE', 15, '*') FROM DUAL;

length n chara with expr 2, replicated as many times as necessary

SQL > SELECT RPAD ('ORACLE', 15, '*') FROM DUAL;

single - ROW operators:

1) IS NULL

Query: SELECT * FROM retail_store_employees
WHERE salary is NULL;

2) IS NOT NULL

Query: SELECT * FROM retail_store_employee
WHERE salary is NOT NULL;

3) LIKE

Query: SELECT * FROM retail_store_employee
WHERE retailstore_employee_name LIKE 'J%
John%';

4) NOT LIKE

Query: SELECT * FROM retail_store_employees
WHERE retailstore_employee_name NOTLIKE
'% John %';

5) BETWEEN

Query: SELET * FROM retail_store_employee
Where salaryary BETWEEN 50000 AND
100000;

VEL TECH	
EX NO	3
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	23
SIGN WITH DATE	7/8/25

Task 4 : developing queries with DML Multi-row functions and operators

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

EMPLOYEES (emp_no, emp_name, departments deptno, salary: age, order (emp_no, order id, Price, qty_ord, qty_hand) item file (itemid, itemname, qty_ord, qty_hard, item rates).

queries using UNION, INTERSECT, MINUS:

UNION:

SQL > select emp_no from empolyees;
SQL > select emp_no from orders;
SQL > select emp_no from employees
union
select emp_no from orders;

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;

developing queries with DML Task 4

SQL> create table society (name varchar(5),houseno number(3),problems varchar(10));

SQL> desc society

Name	Null?	Type
NAME		VARCHAR2(4)
HOUSENO		NUMBER(3)
PROBLEM		VARCHAR2(15)

SQL> insert into society(name,houseno,problem)values('janu',213,'watertank');

1 row created.

SQL> insert into society(name,houseno,problem)values('sri',215,'drainage');

1 row created.

SQL> insert into society(name,houseno,problem)values('ammu',230,'fansound');

1 row created.

SQL> desc society

Name	Null?	Type
NAME		VARCHAR2(4)
HOUSENO		NUMBER(3)
PROBLEM		VARCHAR2(15)

SQL> desc society

Name	Null?	Type
NAME		VARCHAR2(4)
HOUSENO		NUMBER(3)
PROBLEM		VARCHAR2(15)

SQL> select *from society;

NAME HOUSENO PROBLEM

janu 213 watertank

sri 215 drainage

ammu 230 fansound

```
SQL> create table staff(name varchar(6),idno number(7),department varchar(5));
Table created.
SQL> insert into staff(name,idno,department)values('janu',3667,'cse');
1 row created.
SQL> insert into staff(name,idno,department)values('srinu',3668,'ece');
1 row created.
SQL> insert into staff(name,idno,department)values('ammu',3669,'eee');
1 row created.
SQL> select *from staff;
NAME       IDNO DEPAR
------ ---------- -----
janu       3667 cse
srinu      3668 ece
ammu       3669 eee
SQL> desc learnerss
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 STDID                                              NUMBER(38)
 STDNAME                                            VARCHAR2(100)
 STDDEPTNAME                                        VARCHAR2(50)
 STDROLLNO                                          NUMBER(38)
 PHONENUMBER                                        NUMBER(38)

SQL> desc learners
```

```
Name                      Null?   Type
------------------------- ------- ----------------

NAME                              VARCHAR2(15)
VTUNO                             NUMBER(5)
PHONENUMBER                       NUMBER(10)

SQL> insert into learners(name,vtuno,phonenumber)values('janu',29550,'7816025446');

1 row created.

SQL> insert into learners(name,vtuno,phonenumber)values('srinu',27914,'7561938385');

1 row created.

SQL> insert into learners(name,vtuno,phonenumber)values('ammu',29157,'6752398160');

1 row created.

SQL> select *from learners;

NAME            VTUNO PHONENUMBER
--------------- ---------- -----------
janu            29550  7816025446
srinu           27914  7561938385
ammu            29157  6752398160

SQL> select houseno from society union select idno from staff;

  HOUSENO
---------
       213
       215
       230
      3667
      3668
      3669

6 rows selected.

SQL> select houseno from society intersect select idno from staff;
```

no rows selected

SQL> select name from committe intersect select name from staff;

SQL> select vtuno from learners minus select idno from staff;

```
    VTUNO
---------
    27914
    29157
    29550
```

SQL> select vtuno,count(*) from learners group by vtuno;

```
    VTUNO  COUNT(*)
---------- ----------
    29550          1
    29157          1
    27914          1
```

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

```
    VTUNO  COUNT(*)
---------- ----------
    29550          1
    29157          1
    27914          1
```

SQL> select name,idno,department from staff order by department;

```
NAME       IDNO DEPAR
------ ---------- -----
janu         3667 cse
srinu        3668 ece
ammu         3669 eee
```

SQL> select *from learners where vtuno in (27914);

```
NAME            VTUNO PHONENUMBER
```

srinu 27914 7561938385

SQL> select *from learners where phonenumber not in ('7816025446');

NAME VTUNO PHONENUMBER

srinu 27914 7561938385

ammu 29157 6752398160

SQL> select *from staff where exists(select*from learners where vtuno=learners.vtuno);

NAME IDNO DEPAR

janu 3667 cse

srinu 3668 ece

ammu 3669 eee

SQL> select *from staff where not exists(select*from learners where vtuno=learners.vtuno);

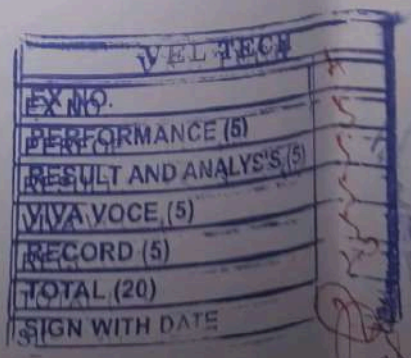
no rows selected

SQL> select *from staff where idno>all(select idno from learners where name='ammu');

no rows selected

SQL> select *from staff where idno>any(select idno from learners where name='janu');

no rows selected



Minus :

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

Queries using Group by, Having clause and order clause :

Group by :

SQL > select dept no; count(*) from employees
group by dept no;

Group by Having :

SQL > select dept no; count(*) from employees group by dept no having Deptno is not null.

order by :

select < column (s) > from < Table Name >
where [condition (s)] [order by < column name > [as (1)] desc];

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

SQL > select empno, emp_name, salary from employee order by salary desc;

SQL * plus having following operators :

SQL> select salary + comm from emp_master;

salary + comm

SQL> select salary + comm net_sal from emp_master;

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

Subqueries:

SQL> select * from employees

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

NOT IN:

Query: SELECT * FROM employees WHERE department NOT IN ('sales', 'marketing');

EXISTS:

Query: SELECT * FROM employees WHERE exists (SELECT * FROM orders WHERE orders emp_no = (link unavaliable));

NOT EXISTS:

Query: SELECT * FROM employees WHERE NOT EXISTS (SELECT * FROM orders WHERE orders. emp_no: (LINK unavailable));

Op ALL:

Verify query: SELECT * FROM employees WHERE NOT e salary > All (SELECT salary FROM employees WHERE department = 'sales');

Result: Thus the implement - ation is successfully completed.

VEL TECH	
EX NO.	4
PERFORMANCE (5)	5
RESULT AND ANALYS'S (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	21/8/25

21/8/25

Task 5: Implementation of different types of joins and recursive queries

Objective:

To implement different types of joins and recursive queries

Syntax:

SELECT column1, 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: 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

example:

select * from item, cust where item.id = custid;

Non-Equi-join:

select * from item, cust where item id < custid;

self join:

Example:

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

Task 5

```
SQL> create table member(membno int,NAME varchar(17));

Table created.

SQL> insert into member values(1,'Alice');

1 row created.
SQL> insert into member values(2,'Bob');

1 row created.
SQL> insert into member values(3,'Charlie');

1 row created.

SQL> insert into member values(4,'David');

1 row created.

SQL> create table Borrow(membno int,Book_ID int);

Table created.

SQL> insert into Borrow values(2,101);

1 row created.

SQL> insert into Borrow values(3,102);

1 row created.

SQL> insert into Borrow values(5,103);

1 row created.

SQL> select * from Borrow;

   MEMBNO    BOOK_ID
---------- ----------
         2        101
         3        102
```

5 103

SQL> select Borrow.membno,member.NAME from member inner join Borrow on member.membno=Borrow.membno;

MEMBNO	NAME
2	Bob
3	Charlie

SQL> select member.NAME,Borrow.membno from member left join Borrow on Borrow.membno=member.membno;

NAME	MEMBNO
Bob	2
Charlie	3
David	
Alice	

SQL> select member.NAME,Borrow.membno from member right join Borrow on Borrow.membno=member.membno;

NAME	MEMBNO
Bob	2
Charlie	3
	5

SQL> select member.NAME,Borrow.membno from member full join Borrow on Borrow.membno=member.membno;

NAME	MEMBNO
Alice	
Bob	2
Charlie	3
David	
	5

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	21/8/25

different types of SQL joins:

Here are the different types of the joins in SQL:

(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 records 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 = table.2.column_name;
```

Right (OUTER) JOIN: Return all records from the right table, and the match records from the left table

```
SELECT column_name(s) FROM table1 RIGHT JOIN
table2 ON table1.column_name = table2.
column_name)
```

FULL (OUTER) JOIN: Return all records when there is a match in either left or right table SELECT column_name(s) FROM table1.

Recursive Queries:

Syntax:

```
WITH Recursive [cte_name](column--)
as (
    [non-recursive_term]
    UNION ALL
```

[recursive_term])
SELECT ___ FROM [cte_name];

1: write a recursive query to create a Multiplication table by 2

```
with Recursive x 2 (result) As (
select 1
union All
SELECT result * 2 FROM x 2)
SELECT * FROM x 2 LIMIT 10;
```

output:

```
1
2
4
8
16
32
64
128
256
512
(10 rows)
```

Example 2: write a recursive query to create a fibonacci sequence

```
WITH RECURSIVE fib (f1, f2) As (
    SELECT 0, 1
    UNION ALL
SELECT f2, (f1 + f2) FROM fib)
SELECT f1 from fib . LIMIT 10;
```

```
f1
 0
 1
 1
 2
 3
 5
 8
13
21
34
(10 rows)
```

VEL TECH	
EX No.	
PERFORMANCE (5)	
RESULT AND ANALYSIS (5)	
VIVA VOCE (5)	
RD (5)	
(10)	
DATE	

Result: The implementation of SQL commands using joins and recursive queries are executed successfully.

20/8/25

Task : 6

PL/SQL Procedure, function & loops

Aim:- To implement PL/SQL procedure, function & and loops on number theory and business scenario

Procedure:

PL/SQL is a combination of SQL along with the Procedural features of programming language, It was developed by oracle corporation

→ declarations:

This sections 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

→ executive Commands:

This is enclosed b/w the keywords Begin and End and it is a mandatory section

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

PL/SQL procedure successfully completed.

dynamic input:

```
sql > declare
  2 var1 integer;
  3 var2 integer;
  5 begin
  6 var1 := &var1;
  7 var2 := &var2;
```

Enter value for var1 = 30
old 6: val1 := &n, var1;
new 6: val1 := 20;
Enter value for var2 : 30
old 7: var2 := &var2;
new 7: var2 : 30;
50
PL/SQL procedure successfully completed

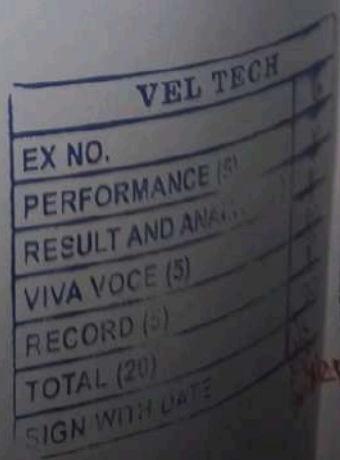
```
declare :
    hid number(3) := 100;
Begin
  if (hid = 10) Then
    dbms - output . put - line (value of hid is 10);
  else if (hid = 20) Then
    dbms_ output . put. line (value of hid is 20);
  elsif (hid = 30) Then
    dbms_ output putline(value of hid is 30);
  else
    dbms_output : put_line (name of the values
  is matching);
    End loop Inner_loop;
  End loop Outer_loop;
End;
```

1

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

```
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.
```

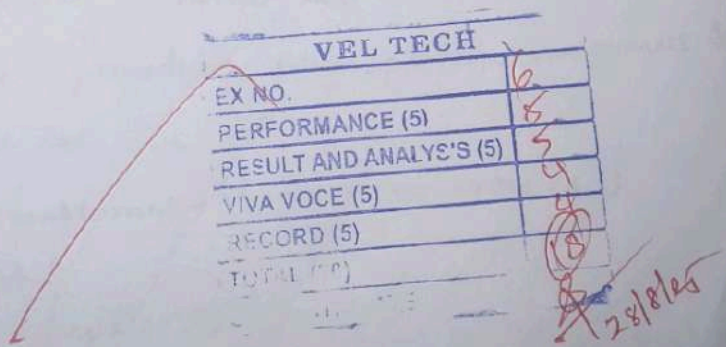


```
hid is 2 and oid is : 3
hid is 3 and oid is : 1
hid is 3 and oid is : 2
hid is 3 and oid is : 3
```

PL/SQL procedure successfully completed

simple program for only function:

```
SQL> create or replace function is_information
(h_id in number, c_name in var2)
return varchar 2
is
begin
if (-id > 200 then)
Return ('no booking available);
else
return ('booking open');
end if;
end;
/
```



Result : Thus the implementation of PL/SQL Procedure, functions, loops is successfully completed.

7/7/25

Task 7: PL/SQL procedure for loops

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

Procedure:

1) start a PL/SQL block or procedure
2) use a cursor (if required) to fetch customer ID's from a table
3) for each ID, check whether it is a prime number using a loop.
4) use for loop / while_loop to demonstrate prime number checking
5) Print the result using DBMS_output_line
6) End the block.

Eg1: using while loop with cursor

Prime check using while loop

```
create or replace procedure print_prime_customers is
    cursor cust_cur is
        select customer_id from customers;
    v_id Number;
    v_is_prime boolean;
    v_i number;
Begin
    open cust_cur;
    loop
        fetch cust_cur into v_id;
        exit when cust_cur % not fo
```

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

```
35  CLOSE cust_cur;
36 END;
37 /
SQL> CREATE OR REPLACE PROCEDURE print_first_n_primes(n NUMBER) IS
 2  v_num      NUMBER := 2;
 3  v_count    NUMBER := 0;
 4  v_is_prime  BOOLEAN;
 5 BEGIN
 6  WHILE v_count < n LOOP
 7   v_is_prime := TRUE;
 9   -- Prime check using FOR loop
10    FOR i IN 2 .. TRUNC(SQRT(v_num)) LOOP
11     IF MOD(v_num, i) = 0 THEN
12      v_is_prime := FALSE;
14     END IF;
15    END LOOP;
17    IF v_is_prime THEN
18     DBMS_OUTPUT.PUT_LINE('Prime: ' || v_num);
19     v_count := v_count + 1;
20    END IF;
22    v_num := v_num + 1;
23  END LOOP;
24 END;
25 /
Procedure created.
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
13  c:=0;
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(5);
3  s number:=0;
4  r number;
5  len number;
7  begin
8  bk:=&bk;
9  m:=bk;
10 len:=length(to_char(bk));
11 while bk>0
12 loop
13 r:=mod (bk,10);
14 s:=s+power(r,len);
15 bk:=trunc(bk/10);
16 end loop;
17 if
18 m=s
19 then
20 dbms_output.put_line('given number is armstrong');
21 else
22 dbms_output.put_line('given number is not an armstrong');
23 end if;
24 end;
Enter value for bk: 234
old  8: bk:=&bk;
new  8: bk:=234;
PL/SQL procedure successfully completed.
```

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

Result: Thus the implementation of PL/SQL is successfully completed

end end loop;
result: Thus the

```
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;
```

example 2 : using for loop

```
Create or Replace procedure print_first_n_primes
                                    (n number) Is
    v_num    Number := 2;
    v_count  Number := 0;
    v_is_prime Boolean;
Begin
    While v_count < n loop
        v_is_prime := True;
        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(v_num);
        v_count := v_count + 1;
    end if
    v_num := v_num + 1;
end loop;
end
```

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	7/9/25

Result: Thus the implementation of PL/SQL procedure for loops is successful..

11/9/25

Task 8: Normalizing databases using functional dependecies upto BCNF

upon relational tables created in task-2, Perform normalization up to BCNF based on given dependecies as following for the assumed relations specified bet

1) Identify employee attributes : Emplee_ID, Name, department, Job_title.
2) define relational schema : (Employee_ID, Name, department, Job_title, Manager_ID, Hire_Date, salary).
3) Determine functional dependencies (FDs) between attributes;

Employee_ID → Name, department, Job_title
Deparment → 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.

Step 4 : convert to 3NF

1) Ensure there are no transitive dependecies
2) Move non-key attributes to separate tables if they depend on another non-key attributes

→ Create Manager table : Manager (Manager_ID, Name).

→ update department table : department

Task 8

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.

CONVERT 2NF :

Normalize to 2NF

Attributes

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

Functional Dependencies

employee_id → name department job_title hire_date manager_id salary

Show Steps

First, find the minimal cover of the FDs, which includes the FDs
employee_id --> name
employee_id --> department
employee_id --> job_title
employee_id --> hire_date
employee_id --> manager_id
employee_id --> salary

Initially rel[1] is the original table:

Round1: checking table rel[1]

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

CONVERT 3NF :

1NF to 3NF

Attributes

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

Functional Dependencies

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

Show Steps

Table already in 3NF

CONVERT BCNF :

Normalize to BCNF

Attributes

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

Functional Dependencies

employee_id → name department job_title hire_date manager_id salary

Show Steps

Table already in BCNF, return itself.

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

Result : Thus the implementation of normalization is successfully completed

Step 5 : Convert to BCNF

1) Ensure every determinant is a candidate key
2) check for overlapping candidate keys
3) decompose relations to elimate redudancy

using Griffith Tool

1) Input relational schema and functional dependencies
2) Apply normalization rule to transmition the schema
3) verify the resulting schema meets BCNF criteria

~~using~~ Griffith Tool steps

1) create a new project in griffith
2) Define the relational schema and FD's
3) Run the "Dependency graph" 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	11/9/25

Result: Thus the implementation of normalization is succesfully completed

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 NO ARCHIVELOG Database with Incremental Backup's

Step 1: Backup database

BACKUP DATABASE [data_base name] TO DISK = 'backup_file.bak' WITH NOFORMAT,

Step 2: Create Incremental Backup

BACKUP DATABASE [data_base_name] TO DISK = 'Incremental_backup.bak' WITH DIFFERENTIAL, NOFORMAT, NOINIT, NAME = 'Incremental database Backup', SKIP, REWIND, NOUNLOAD, STATS = 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 4 : open data base

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 server parameter file from file = 'spfile.bak';

shutdown

startup

Scenario : 3 : performing Recovery with a Backup control file

Step 1 : Backup control file

BACKUP CONTROLFILE TO FILE = 'controlfile.bak';

step 2 : simulate control file loss

delete or modify control file

Step 3 : Restore control file

Step 4 : Recover Database

Step 5 : open database

SQL server commands

BACKUP DATABASE

RESTORE DATABASE

RECOVER DATABASE

AILTER DATABASE

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 : Thus the implementation of Backing up and recovery in data bases.

05/9/25

Task-10: CRUD OPERATIONS IN DOCUMENT DATABASES.

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

Steps:

Step 1: Install Mango db using following link

Step 2: Install Mongosh using the below link

Step 3: To add the mongo DB shell binary's location to your PATH environment variable:

open the control panel.

click advanced system settings. The system Properties modal displays

click environment variables

click New and add the filepath to your mongosh binary.

CRUD OPERATIONS:

```
db.createCollection("mylab")
{"ok": 1}
> db.mylab.insertOne({item: "canvas", canvas",
  qty: 100, tags: ["cotton"], size: {h: 28, w: 35.5,
      uom: "cm"}})
{
"acknowledged": true,
"insertedId": objectId("627d13acc73990c0744c6397")
}
> db.mylab.find({item: "canvas"})
{"_id": objectId("627d13acc73...
```

```
db.createCollection("mylab")

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

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

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

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

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

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

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

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

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

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

output:

Output:

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

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

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

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

```
"acknowledged": true,
"insertedIds": [
        objectId("627d1598cc73990c074e6397d"),
        objectId("627d1598c73990c074e6397c"),
        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("627d13acc73990c074e6397c"),
"item": "canvas",
"qty": 100
}
{
"_id": objectId("627d1598c73990c074e6397d"),
"item": "journal",
"qty": 25
}
> db.mylab.find({item:"canvas"}).pretty().sort
        ({item:-1})
{
"_id": objectId("627d13acc73990c074e6397c"),
"item": "canvas",
"qty": 100,
"tags": [
"cotton"
    ],
"size": {
"h": 28,
"w": 35.5,
"uom": "cm"
    }
}
> db.mylab.deleteOne({item:"journal"})
...
```

```
db.mylab.find({}, {item:1, qty:1}).pretty()
{
"_id" : objectId("627d13acc73990c074c6397c"),
"item" : "canvas",
"qty" : 100
}
{
"_id": objectId("627d1598c73990c074e6397c"),
"item" : "canvas",
"qty" : 100
}
{
"_id": objectId("627d13acc73990c074e6397c"),
"item": "canvas",
"qty :100.
}
{"_id": objectId("627d1598c73990c074e6397d"),
"item": "journal",
"qty": 25
}
{"_id": objectId("627d1598c73990c074e6397e"),
"item": "mat", "qty": 85}
{
"_id": objectId("627d1598c73990c74e6397f"),
"item": "mousepad", "qty":25}
```

VEL TECH	
EX NO.	10
PERFORMANCE (5)	5
RESULT AND ANALYS'S (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	25

25/9/25

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

9/10/25

Task11 : CRUD OPERATIONS IN GRAPH DATABASES

Aim : To perform CRUD operations like creating, inserting, querying, finding, deleting 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 like 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 arrow "→" as shown in the following syntax.

Syntax :

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

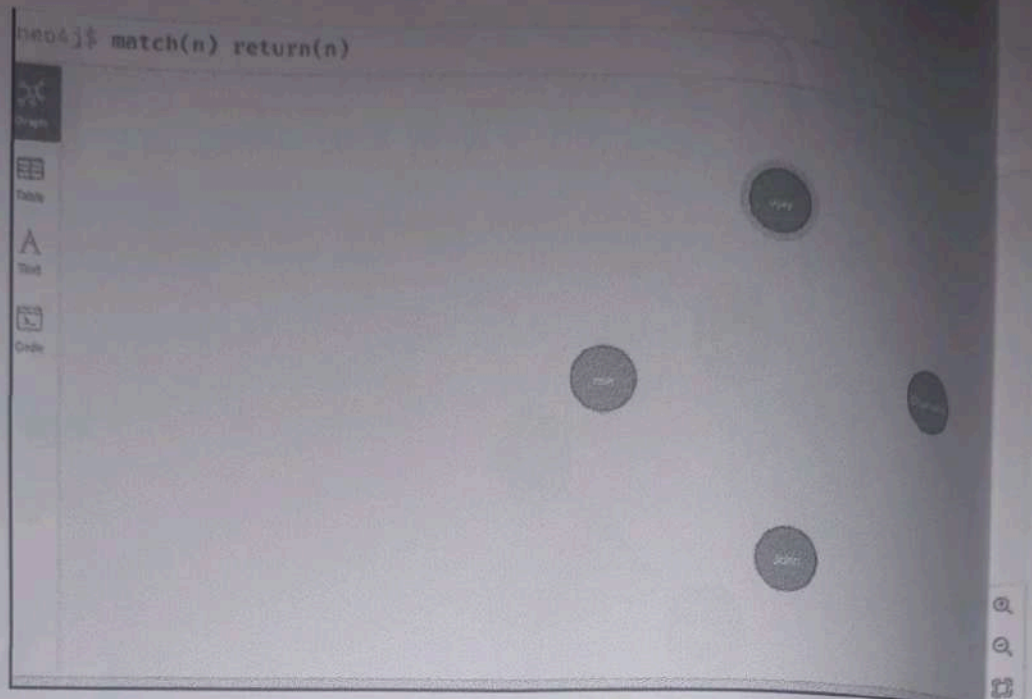
CREATE (node1)-[:RelationshipType]->(node2)

Task 11

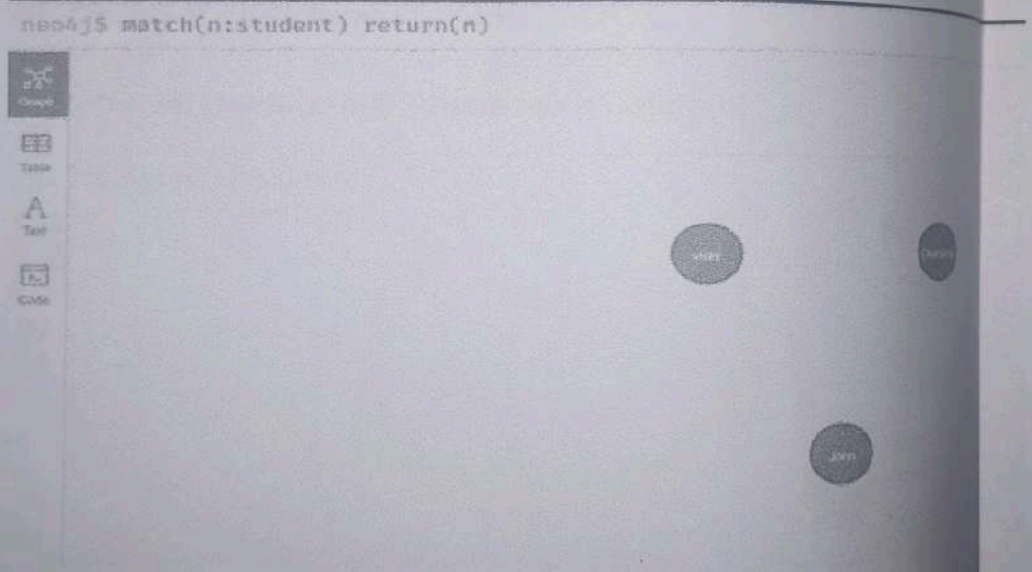
Creed operations in Graph database

Select all the nodes in your database using match command.

match(n) return(n)



match(n:student) return(n)

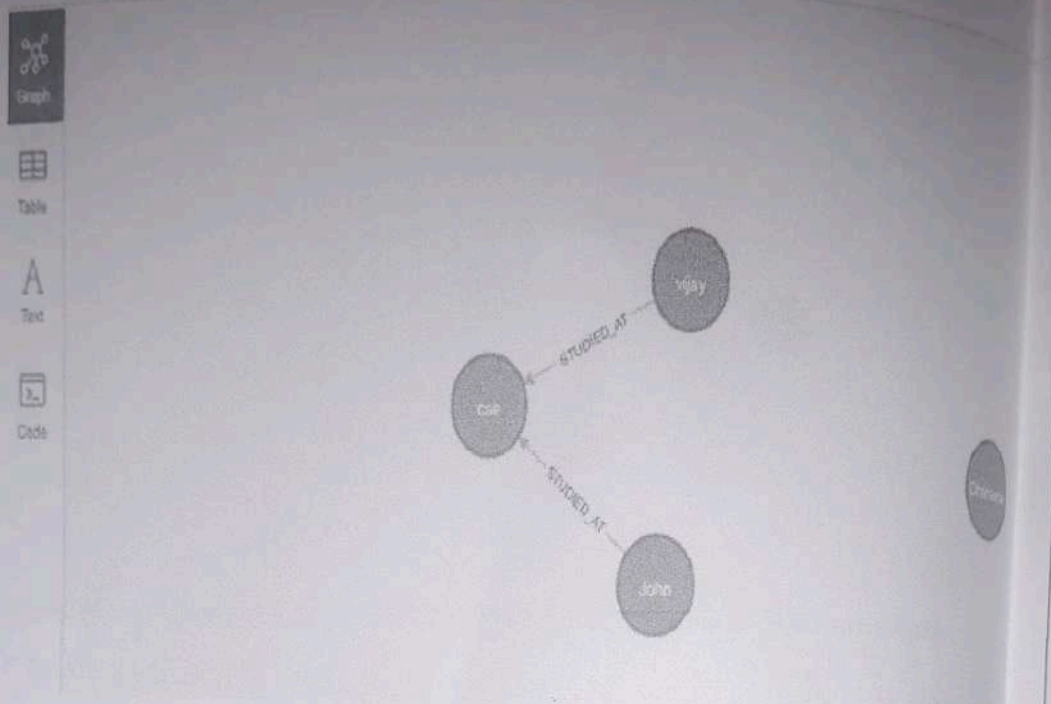


a) Create relationship between student and cse .

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

match(n) return(n)

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

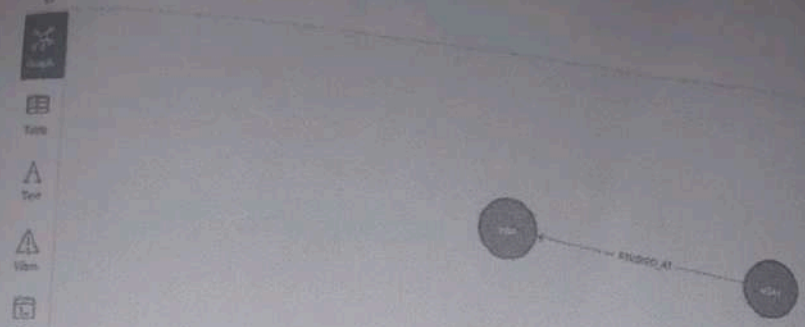


b) Delete a node from student

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

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

```
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
4
5
6
7
8
```



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

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

return s,d

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	

Result : Thus the implementation of crud operations in graph databases is successfully completed :

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

Syntax

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

syntax

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

MATCH (a: Label of Node 1), (b: Label of Node 2)
WHERE a.name = "name of node 1" AND b.name = "name of node 2"
CREATE (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 delete a particular node from NEO4j using the DELETE clause.

MATCH (node: Label {properties... })
DEMATCH DELETE node

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

Output :

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

Create (n: student { sid: "VTU14502",

Sname : "vijay",

deptname : "CSE"

})

output

Added 1 label, created 1 node, set 3 properties, completed after 12ms

Create (n: dept { deptname : "CSE", deptid : "d001" })

output :

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

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

Result : Thus the implementation of CRUD operations in graph database is successfully completed.

DATABASE MANAGEMENT SYSTEMS

(10211DS207)

TASK-12

HOTEL ROOM BOKING MANAGEMENT SYSTEM

Team Details:

Team Leader:B.Neha joshika(VTU30499)

Reg No:24UEDC0012

Team Members:

Names	VTU	Registration no
A .Prudhvi raj	30505	24UEDC0001
M .Javeed khan	30527	24UEDC0046
L.Ranjeeth kumar	30529	24UEDC0033
B.Lenin	30531	24UEDC0015

Aim: To develop a microproject on hotel management system

1 ER Diagram Definition :

An **Entity–Relationship (ER) Diagram** is a graphical representation of entities and the relationships between them in a database system.

In the **Hotel Room Booking Management System**, the ER diagram helps in designing the database by identifying key entities like *Hotel, Room, Customer, Booking,* and *Payment*, and showing how they interact with each other.

This system manages room availability, customer details, bookings, and payments efficiently.

2. Entities and Their Attributes

a) Guests

- **Primary Key:** Customer_ID
- **Attributes:**
  - Customer_Name
  - Contact_Number
  - Email
  - Address
  - ID_Proof

b) Room

- **Primary Key:** Room_ID
- **Attributes:**
  - Room_Type (Single/Double/Deluxe)

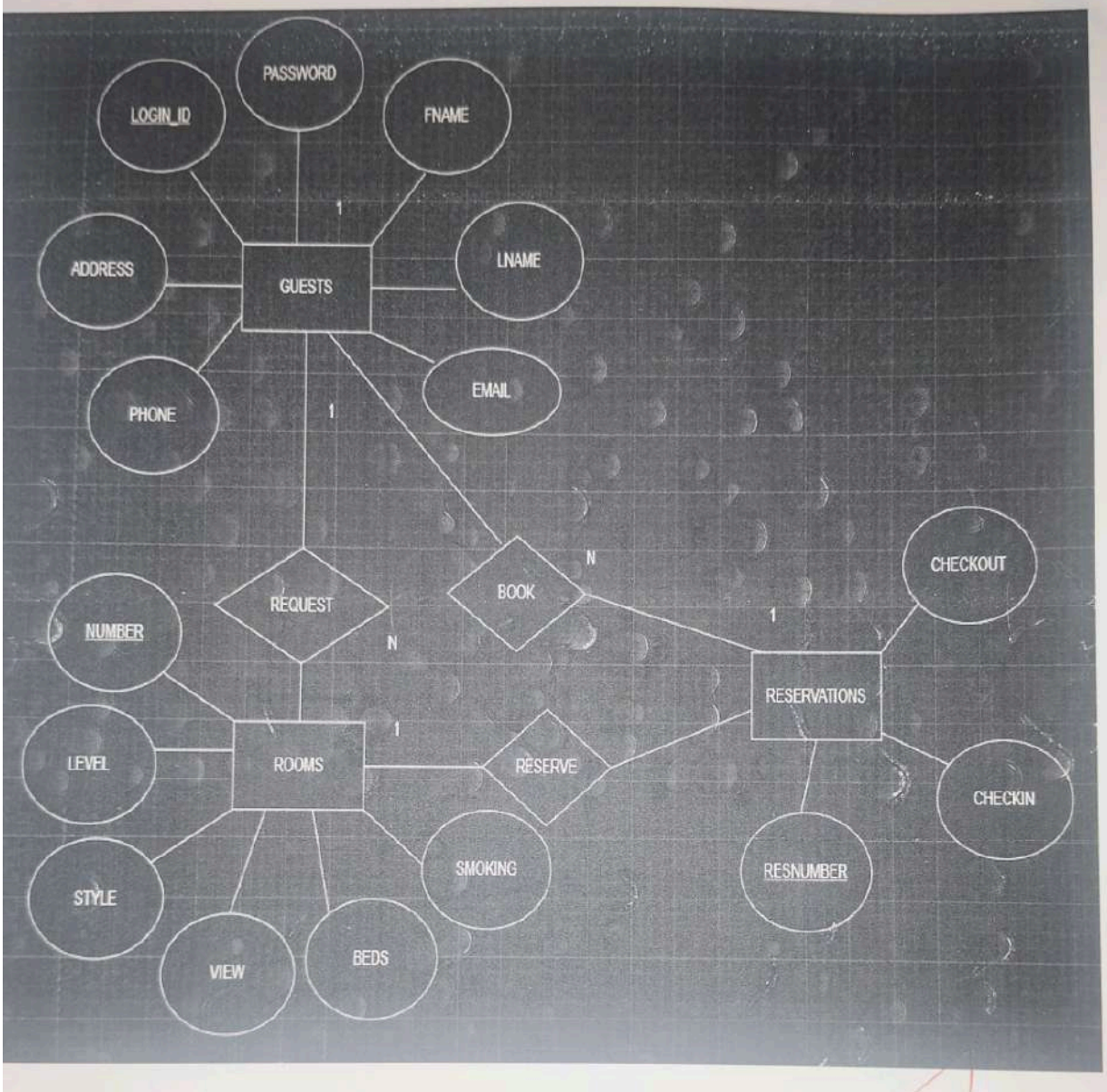
- Room_Status (Available/Booked)
- Level
- Style
- View,beds,smoking
- Room_no

c) Reservation

- **Primary Key:** Booking_ID
- **Attributes:**
  - Booking_Date
  - CheckIn_Date
  - CheckOut_Date
  - Payment_Status

Relationship	Description	Cardinality
Customer – Booking	One customer can make many bookings. Each booking belongs to one customer.	1 : M
Booking – Room	Each booking can include one or more rooms. Each room can be booked multiple times (over time).	M : N
Booking – Payment	Each booking has one payment record.	1 : 1
Hotel – Room	A hotel contains many rooms, but each room belongs to one hotel.	1 : M

ER DIAGRAM


PASSWORD
LOGIN_ID
FNAME
1
ADDRESS
GUESTS
LNAME
PHONE
1
EMAIL
N
CHECKOUT
REQUEST
BOOK
NUMBER
N
1
RESERVATIONS
1
LEVEL
ROOMS
RESERVE
CHECKIN
SMOKING
RESNUMBER
STYLE
VIEW
BEDS

2.SQL Queries & Relational operations:

➢ Select operation:

SELECT * FROM Customer;

Output:

Customer_ID	Customer_Name	Contact_Number	Email	Address	ID_Proo
101	Riya Sharma	9876000001	riya@gmail.com	Delhi	A12345
102	Arjun Mehta	9876000002	arjun@gmail.com	Mumbai	B67890

➢ Projection operation:

SELECT Customer_Name, Contact_Number FROM Customer;

Output:

Customer_Name	Contact_Number
Riya Sharma	9876000001
Arjun Mehta	9876000002

➢ Union operation:

SELECT City FROM Customer
UNION
SELECT City FROM Guest;

Output:

City
Delhi
Mumbai
Chennai

➢ Union All:

```
SELECT City FROM Customer
UNION ALL
SELECT City FROM Guest;
```

Output (with duplicates):

City
Delhi
Mumbai
Chennai
Mumbai
Delhi
Mumbai

➢ Intersection :

```
SELECT City FROM Customer
WHERE City IN (SELECT City FROM Guest);
```

Output

City
Delhi
Mumbai

➢ SUM :

```
SELECT SUM(Amount_Paid) AS Total_Revenue
FROM Payment;
```

Output

Total_Revenue
27000.00

➢ Count :

```
SELECT COUNT(*) AS Total_Customers
FROM Customer;
```

Output
City
Delhi
Mumbai

➢ AVG:

Table example

Payment_ID	Amount_Paid
401	15000
402	5000
403	7000
404	10000

syntax

```
SELECT AVG(Amount_Paid) AS Average_Payment
FROM Payment;
```

Output
Average_Payment
9250.00

➢ MAX:

```
SELECT MAX(Amount_Paid) AS Maximum_Payment
FROM Payment;
```

Output
Maximum_Payment
15000.00

➢ MIN:

```
SELECT MIN(Amount_Paid) AS Minimum_Payment
FROM Payment;
```

Output

Minimum_Payment
5000.00

➢ NESTED QUERIES:

Example table:

Customer table

Payment_ID	Amount_Paid
401	15000
402	5000
403	7000
404	10000

Booking table

Booking_ID	Customer_ID	Total_Amount	Payment_Status
301	101	15000.00	Paid
302	102	5000.00	Pending
303	103	7000.00	Paid

Payment table

Payment_ID	Booking_ID	Amount_Paid
401	**301**	**15000.00**
402	**303**	**7000.00**

Syntax :

```
SELECT Customer_Name
FROM Customer
WHERE Customer_ID IN (
  SELECT Customer_ID
  FROM Booking
  WHERE Booking_ID IN (
    SELECT Booking_ID
    FROM Payment
    WHERE Amount_Paid > 10000
  )
);
```

Output

Customer_Name
Riya Sharma

3. JOINS

Customer table

Customer_ID	Customer_Name	City
101	Riya Sharma	Delhi
102	Arjun Mehta	Mumbai
103	Neha Singh	Chennai

Booking table

Booking_ID	Customer_ID	Room_ID	Total_Amount
301	101	201	15000.00
302	102	202	5000.00
303	104	203	7000.00

➢ Inner Join

```
SELECT Customer.Customer_ID, Customer.Customer_Name, Booking.Booking_ID,
Booking.Total_Amount
FROM Customer
INNER JOIN Booking
ON Customer.Customer_ID = Booking.Customer_ID;
```

Output

Customer_ID	Customer_Name	Booking_ID	Total_Amount
101	Riya Sharma	301	15000.00
102	Arjun Mehta	302	5000.00

➢ Left outer join

```
SELECT Customer.Customer_ID, Customer.Customer_Name, Booking.Booking_ID,
Booking.Total_Amount
FROM Customer
LEFT JOIN Booking
ON Customer.Customer_ID = Booking.Customer_ID;
```

Output

Customer_ID	Customer_Name	Booking_ID	Total_Amount
101	Riya Sharma	301	15000.00
102	Arjun Mehta	302	5000.00
103	Neha Singh	NULL	NULL

➢ Right outer join

```
SELECT Customer.Customer_ID, Customer.Customer_Name,
Booking.Booking_ID, Booking.Total_Amount
FROM Customer
RIGHT JOIN Booking
ON Customer.Customer_ID = Booking.Customer_ID;
```

Output

Customer_ID	Customer_Name	Booking_ID	Total_Amount
101	Riya Sharma	301	15000.00
102	Arjun Mehta	302	5000.00
NULL	NULL	303	7000.00

Full outer join

```
SELECT Customer.Customer_ID, Customer.Customer_Name, Booking.Booking_ID,
Booking.Total_Amount
FROM Customer
LEFT JOIN Booking
ON Customer.Customer_ID = Booking.Customer_ID
UNION
SELECT Customer.Customer_ID, Customer.Customer_Name, Booking.Booking_ID,
Booking.Total_Amount
FROM Customer
RIGHT JOIN Booking
ON Customer.Customer_ID = Booking.Customer_ID;
```

Output

Customer_ID	Customer_Name	Booking_ID	Total_Amount
101	Riya Sharma	301	15000.00
102	Arjun Mehta	302	5000.00
103	Neha Singh	NULL	NULL
NULL	NULL	303	7000.00

4. Normalization

Normalization in databases is the process of organizing data to reduce redundancy and improve data integrity. It breaks a large, disorganized table into smaller, more manageable ones and links them using defined relationships. This makes the database more efficient, accurate, and reliable.

The normal forms

The process of normalization is guided by a series of rules called "normal forms." A database must satisfy the rules of the previous normal form before moving to the next. Most databases only require normalization up to the Third Normal Form (3NF).

- First Normal Form (1NF): Each column must contain only a single, indivisible (atomic) value, and the table must have no repeating groups of data. Any multi-valued attributes are placed into a new table.
- Second Normal Form (2NF): The table must be in 1NF, and every non-key attribute must depend on the *entire* primary key. If a table has a composite key, any attributes that depend on only part of that key are moved to a new table.
- Third Normal Form (3NF): The table must be in 2NF, and all attributes must depend directly on the primary key. Any transitive dependencies, where a non-key attribute depends on another non-key attribute, are moved to a separate table.

First Normalization form:

Attributes in Table

Separate attributes using a comma (,)

Customer_ID,Name,Contact_No,Address,Email

Functional Dependencies

Customer_ID ×	→	Name × Contact_No × Address × Email ×	Delete
Email ×	→	Contact_No × Name × Customer_ID ×	Delete
Contact_No ×	→	Customer_ID × Email × Name ×	Delete

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 { Customer_ID}, { Contact_No}, { Email}, The set of key attributes are: { Customer_ID,Contact_No,Email }
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: Customer_ID --> Name,Contact_No,Address,Email
checking FD: Email --> Contact_No,Name,Customer_ID
checking FD: Contact_No --> Customer_ID,Email,Name
```

3NF

```
find all candidate keys. The candiates keys are { Customer_ID}, { Contact_No}, { Email}, The set of key attributes are: { Customer_ID,Contact_No,Email }
for each FD, check whether the LHS is superkey or the RHS are all key attributes
checking functional dependency Customer_ID --> Name,Contact_No,Address,Email
checking functional dependency Email --> Contact_No,Name,Customer_ID
checking functional dependency Contact_No --> Customer_ID,Email,Name
```

BCNF

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

Normalize to 2NF

Normalize to 2NF

Attributes

`Customer_ID` `Name` `Contact_No` `Address` `Email`

Functional Dependencies

```
Customer_ID → Address Email
Email → Contact_No
Contact_No → Customer_ID Name
```

Show Steps

```
First, find the minimal cover of the FDs, which includes the FDs
Customer_ID --> Address
Customer_ID --> Email
Email --> Contact_No
Contact_No --> Customer_ID
Contact_No --> Name

Initially rel[1] is the original table:
Round1: checking table rel[1]
***** The table is in 2NF already, send it to output *****
```

1 NF To 3 NF

1NF to 3NF

Attributes

Customer_ID | Name | Contact_No | Address | Email

Functional Dependencies

Customer_ID → Address
Customer_ID → Email
Email → Contact_No
Contact_No → Customer_ID
Contact_No → Name

Show Steps

Table already in 3NF

Normalize to BCNF

Normalize to BCNF

Attributes

Customer_ID | Name | Contact_No | Address | Email

Functional Dependencies

Customer_ID → Address Email
Email → Contact_No
Contact_No → Customer_ID Name

Show Steps

Table already in BCNF, return itself.

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.

Create:

```
db.createCollection("customers");
db.customers.insertMany([
{ Customer_ID: 101, Name: "RiyaSharma", Address: "Delhi", Room_ID: 201, Payment_ID: 501, Amount: 4000 },
{ Customer_ID: 102, Name: "AmanGupta", Address: "Mumbai", Room_ID: 202, Payment_ID: 502, Amount: 5000 },
{ Customer_ID: 103, Name: "NehaJoshika", Address: "Chennai", Room_ID: 203, Payment_ID: 503, Amount: 3000 }]);
```

Output:

Output:

```
{ "ok" : 1 }
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("68f0f5ec04c948845a1604da"),
                ObjectId("68f0f5ec04c948845a1604db"),
                ObjectId("68f0f5ec04c948845a1604dc")
        ]
}
```

Read:

```
db.customers.find({ Address: "Delhi" });
```

Output:

```
{ "_id" : ObjectId("68f0f5ec04c948845a1604da"), "Customer_ID" : 101, "Name" : "RiyaSharma", "Address" : "Delhi", "Room_ID" : 201, "Payment_ID" : 501, "Amount" : 4000 }
```

Update:

```
db.customers.updateOne(
  { Customer_ID: 103 },
  { $set: { Address: "Bangalore", Amount: 3500 } }
);
```

Output:

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Delete:

```
db.customers.deleteOne({ Customer_ID: 102 });
```

Output:

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

6.Graph Database using MONGODB

a)create a graph database:

```
CREATE(c1:Customer {Customer_ID: 101, Name: 'Riya Sharma', Address: 'Delhi'});
```

Output:

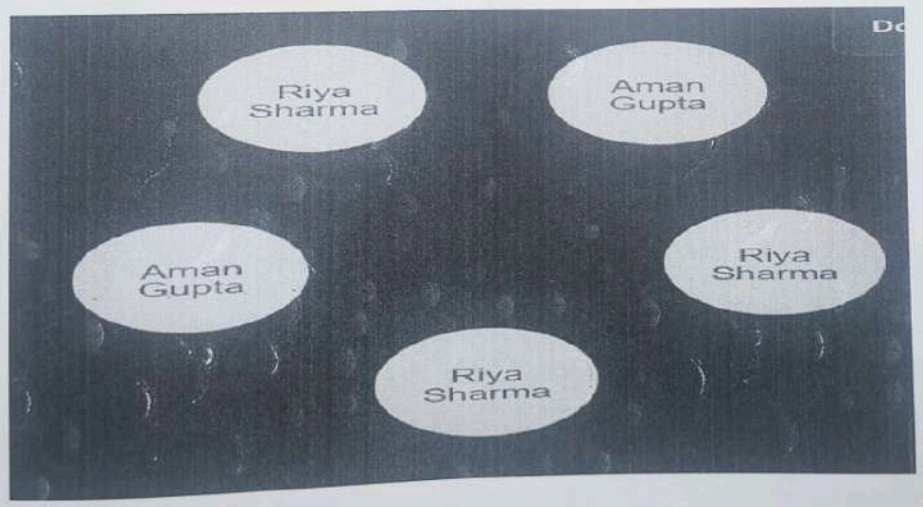
```
Created 1 node, set 3 properties, added 1 label
```

```
CREATE (c2:Customer {Customer_ID: 102, Name: 'Aman Gupta', Address: 'Mumbai'});
```

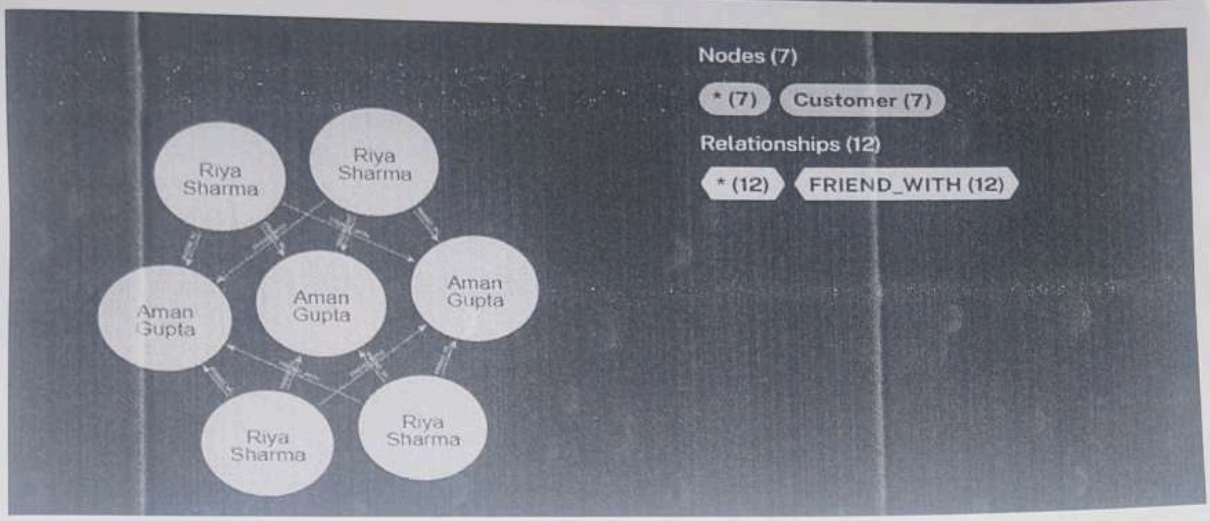
Output:

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

```
MATCH (n) RETURN (n);
```



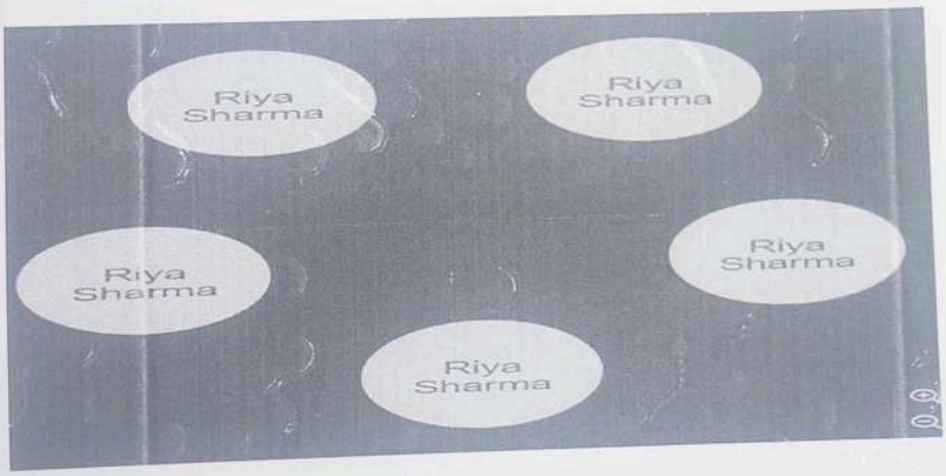
```
MATCH (c1:Customer {Customer_ID: 101}), (c2:Customer {Customer_ID: 102})
CREATE (c1)-[:FRIEND_WITH]->(c2);
```



Delete:

```
MATCH (c:Customer {Customer_ID: 102})
DETACH DELETE c;
```

Output:



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

Result : Thus,Micro Project for Dairy Farming System was developed and implemented successfully.

VEL TECH

EX NO.	
PERFORMANCE (5)	
RESULT AND ANALYSIS (5)	
VIVA VOCE (5)	
RECORD (5)	
	(20)