TASK PLAN

Library Management System

The library management system is a software to manage the manual functions of a library. The software helps to manage the entire library operations from maintaining book records to issue a book. In addition, it allows streamlined management of fine details of books such as author name, edition, and many other important details. So, it is easier to search for books and find the right materials for students and the librarian.

Electronic management via the software is essential to track information like issue date, due date, who has borrowed any material, etc. The system is developed and designed with an aim to facilitate efficient management of the schools to manage a modern library with accurate data management.

Components of a Library Management System

In order to maintain library management software, you will have the following set of components. These components are efficient to manage library operations accurately.

Admin: the administrators can access the entire functionality of the system via this component. The admin can maintain the records and track them as necessary. Also, the admin can add or remove entries into the system respectively.

Reader: the students who want to access library materials must do registration first. The registration allows for maintaining records accurately. After registering, they can check out and check in the library material.

Book: The admin can add new books or other materials to the system with the essential details. Each book has authno, isbn number, title, edition, category, PublisherID and price.

Publisher: The publisher has PublisherId, Year of publication and name.

Report: It has UserId, Reg_no, Book_no and Issue/Return date. Admin can view the issued materials with their due date. And, if any book is overdue, the system will allow calculating fine for the same.

Relation

- Book(authno, isbn number, title, edition, category, PublisherID, price)
- Reader(UserId, Email, address, phone no, name)
- Publisher(PublisherId, Year of publication, name)
- Report(UserId, Reg_no, Book_no, Issue/Return date)
- Admin (LoginId, password)

TASK 1: Conceptual Design through FTR

CO1, S3

(Tool: Creately/ERD Plus ,ALM:Think pair share)

Using basic database design methodology and ER modeler, design Entity Relationship

Diagram by satisfying the following sub tasks:

1. a Identifying the entities.
1. b Identifying the attributes.

Sample Output

- Publisher(PublisherId, Year of publication, name)

- Admin(LoginId, password, name, staff_id)
- Report(UserId, Reg_no, Book_no, Issue/Return date)
- Book(authno, isbn number, title, edition, category, PublisherID, price)
    - Name is composite attribute of firstname and lastname.
    - Phone no is multi valued attribute.
- Reader(UserId, Email, address, phone no, name)

- c Identification of relationships, cardinality, type of relationship.

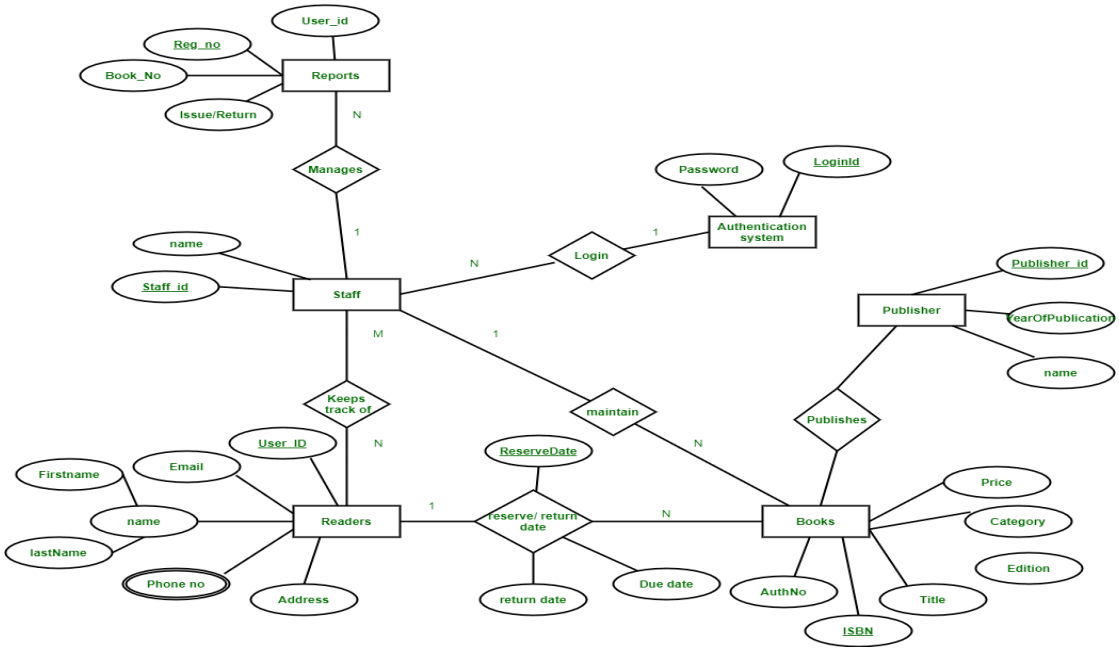
Sample Output

- A reader can reserve N books but one book can be reserved by only one reader. The relationship 1:N.
- A publisher can publish many books but a book is published by only one publisher. The relationship 1:N.
- Admin keeps track of readers. The relationship is M:N.
- Admin maintains multiple reports. The relationship 1:N.
- Admin maintains multiple Books. The relationship 1:N.
- Admin provides login to multiple staffs. The relation is 1:N.
- d Reframing the relations with keys and constraint.

Sample Output

- {isbn number} is the Primary Key for Book Entity.
- {authno, isbn number, title} – Super key of Book entity.
- {PublisherId} is the foreign key of Book entity.
- {authno,title} – Candidate key of Book entity.
- UserId is the Primary Key for Readers entity.
- PublisherID is the Primary Key for Publisher entity
- LoginID as Primary Key for Authentication system entity.
- Reg_no is the Primary Key of reports entity.

1. e Using creately, develop ER diagram.



TASK 2: Generating design of other traditional database model CO1, S3

(Tool: Creately, ALM:Mind map)

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

2. a Identify the specificity of each relationship, find and form surplus relations.
2.b Check is-a hierarchy/ has-a hierarchy and performs generalization and/or specialization relationship.
2. c Find the domain of the attribute and perform check constraint to the applicable.
2. d Rename the relations.
2. e Perform SQL Relations using DDL, DCL commands.

- Create all relations using DDL comments
- Alter the table Reader by adding email_id as a field.

Sample output
Syntax: Alter table reader add emailed varchar(20);
Output: Table altered

- Retrieve all the books under the category dbms.
- COMMIT the table after inserting values
- Rollback the table.

TASK 3:Using Clauses, Operators and Functions in queries:
CO2, S3

(Tool: SQL/ Oracle, ALM:Fish bowl)

Perform the query processing on databases for different retrieval results of queries using DML, DRL operations using aggregate, date, string, indent functions, set clauses and operators.

- Retrieve all the author who wrote in dbms.
- Retrieve total number of books offered in the category program core
- Retrieve all authno and name who published books after 2000
- Retrieve readers name end with letter 'a'
- Retrieve number of readers studied in each department.

Sample Output:

ECE	800
CSE	850
EEE	1000

- Retrieve all the female readers
- Retrieve all the staff who came library yesterday.

TASK 4: Writing Sub Queries and Join Queries:
CO2, S3

(Tool: SQL/ Oracle, ALM: Fish bowl)

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

- Retrieve isbnnumber ,authnumber and title of the books published under the course DBMS
- Retrieve reader name,id,dept id,department who studied in department CSE .
- Retrieve number of staff borrowed from each category of course
- Retrieve publisher id, year, name of the book published under the course OS.
- Retreive all the author name who are all published more than 30 books in the year 2022.
- Retrieve all the male readers name and their emailid

Sample input:
<refer reader, emailid)
Sample output:

name	emailid
Rahul	jeni@gmail.com

TASK 5: Design Datalog query and recursive queries(Tool: SQL/ Oracle)

CO2, S2

Make use of Datalog query designing and recursive query for student registered for a course.

Find the prerequisite of object-oriented software engineering using recursive query

Sample input:

```
WITH RECURSIVE
Coursename(pre1, pre2) AS
(
(SELECT cname, pre-sub FROM pre_requisites)
UNION
(SELECT a1.pre1, a2.pre2
FROM coursename a1, Ancestor a2
WHERE a1.pre2 = a2.pre1)
)
SELECT pre1
FROM Ancestor
WHERE pre2= 'OOSE';
```

Sample output:

PROBLEM SOLVING USING C

TASK 6: Procedures, Function and Loops:

CO3, S3

(Tool: SQL/ Oracle)

Programming using PL/SQL Procedures, Functions and loops on Number theory and business scenarios like.

- Write PL/SQL procedure using while loop, printing prime numbers in a range given.

- Write PL/SQL function recursion for factorial finding and calculate nth term.

- Write PL/SQL block without procedure/function to print all even multiples of 4,8 and not of 32 below 500.

- Write a non-recursive procedure for palindrome checking.

Sample input:

Enter input: madam

Sample output:

Madam is Palindrome

- Write a simple loop program to print 1 2 3 vertically using PL/SQL loop

TASK 7: Triggers, Views and Exceptions (Tool: SQL/ Oracle)

CO3, S3

Conduct events, views, exceptions on CRUD operations for restricting phenomenon

- Create a simple trigger before insert or update or delete trigger in student table

Sample input:

<refer user schema>

Sample output:

S_id is inserted successfully

- Create a view of all readers name and emaiid who are currently in fourth semester.

Sample input:

<refer reader schema>

```
create trigger book_copies_deducts
after INSERT
on book_issue
for each row
update book_det set copies = copies - 1 where bid = new.bid;
```

Sample output:

```
mysql> insert into book_issue values(1, 100, "Java");
Query OK, 1 row affected (0.09 sec)

mysql> select * from book_det;
+-----+-------------+--------+
| bid | btitle      | copies |
+-----+-------------+--------+
|   1 | Java        |      9 |
|   2 | C++         |      5 |
|   3 | MySql       |     10 |
|   4 | Oracle DBMS |      5 |
+-----+-------------+--------+
4 rows in set (0.00 sec)
```

- Raise an exceptional handling whenever a user tries to identify publisherid that does not exists from Publisher Table.

TASK 8: CRUD operations in Document databases (Tool: MongoDB) **CO3, S3**

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

Perform the following tasks of library databases.

- Design mongoDB collection for students
- Insert single student detail at a time

Sample input/output:

- Db,student.insertOne({s_id:VTU12345",Name:"Stephen",year:"3",dept_id:"101",contact no,"9876543210" },
- 1 document inserted
- Db.book.find().pretty();
- Insert multiple students at a time
- Insert all at a time.
- Find students who enrolled Java
- Find coursename which is registered by most number of students
- Delete single student record at a time
- Delete multiple student at a time
- Delete all student at a time who are all registered more than
- Update Faculty at a time who is taken single course
- Update multiple faculty at a time who are all handled compiler design

- Update all faculty at a time

TASK 9: CRUD operations in Graph databases (Tool:Neo4j) **CO3, S3**

Perform GraphQL/Neo4j graph space design for recommendation engines. Also perform CRUD operations like creating, inserting, querying, finding, deleting operations on graph spaces.

- Create a graph database for different categories of courses offered by the department and their pre-requisites
- Create a node for prerequisites

Sample input:

Create (pr: pre-requisites
{ ccode:123, cname:"DBMS",pre_id:"12345",pre_sub: “DS”
}
)

- Insert a course details in each course category
- Delete a course enrolled by student if not satisfied the prerequesites

TASK 10: Normalizing databases using functional dependencies upto BCNF CO1, S2

(Tool: GU/ Table Normalization Tool, ALM:Learning by doing)

Upon relational tables created in task-2, perform normalization up to BCNF based on given Dependencies as following for the assumed relations specified:

Sample input

- Book(authno, isbn number, title, edition, category, PublisherID, price)
- Reader(UserId, Email, address, phone no, name)
- Publisher(PublisherId, Year of publication, name)
- Admin(LoginId, password, name, staff_id)
- Report(UserId, Reg_no, Book_no, Issue/Return date)

Isbn number authno,title,publisher id
Isbn number edition,category
Isbn number authno

useridname,contactno,
userid name

PublisherId coursename,credits,category

LoginId name, staff_id
LoginId staff_id

3. a Apply the functional dependency, normalize to 1NF
3. b Normalize the relations using FD+ and α+
3. c Find the minimal cover, canonical cover.
3. d Normalize to 2NF, add/alter constraints if necessary.
3. e Normalize to BCNF, add/alter constraints if necessary.
3. f Normalize to 3NF, add/alter constraints if necessary.
3. g Perform SQL Relational operations using simple DML queries.

Sample output

TASK 11: Menus, Forms and Reports: CO4, S3

(Tool: SQL/ Oracle 11g ,ALM:Pick the winner)

For an application, creating and debugging Menus, Forms and reports using Oracle Forms and Report Builder,make a report of students with their details.

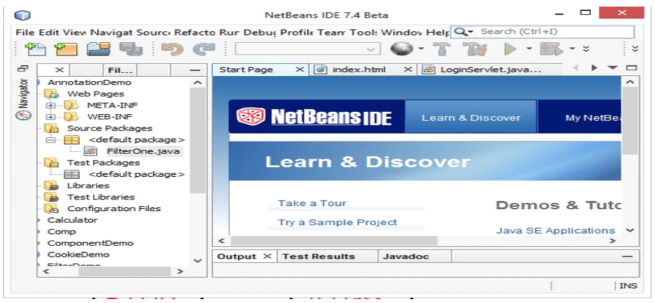
Sample input:

Make a report of students with their details

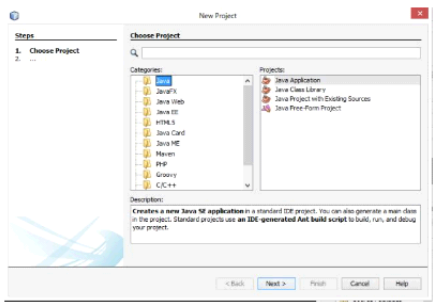
Sample output:

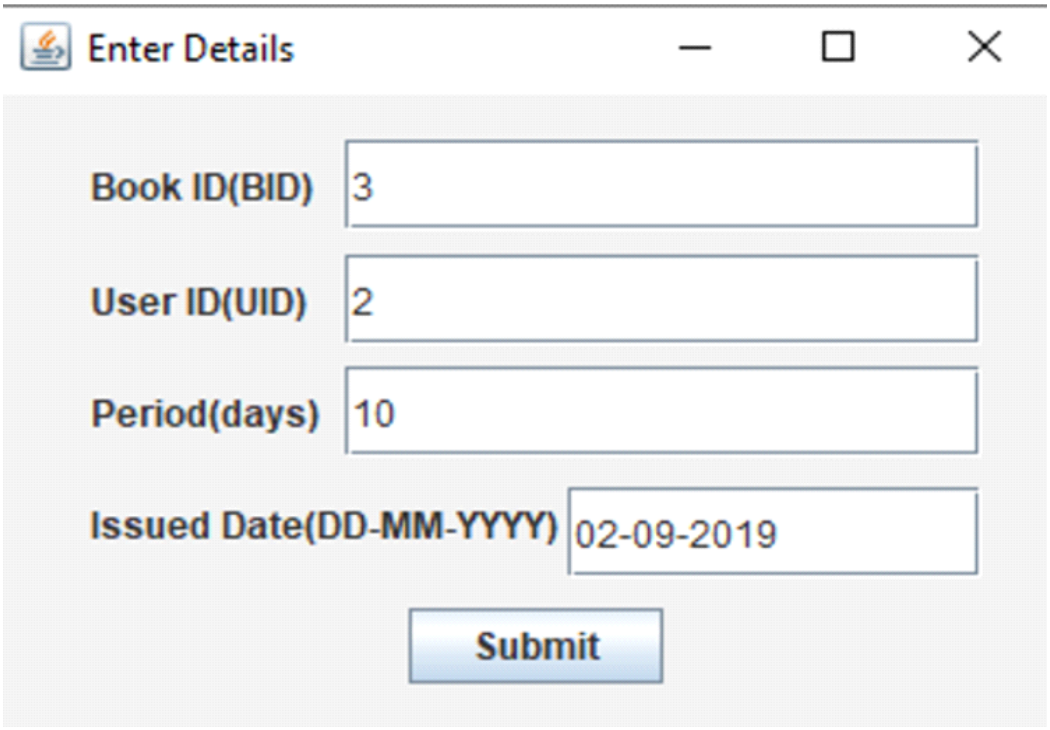
s_id	s_name	year	dept_id	dob	gender	place	phone_no	email_id
1234	TEJA	3	301	17/08/2001	M	Chennai	9876543210	xyz@gmail.com

Open the NetBeans IDE.



Choose "Java" -> "Java application" as shown below





TASK 12: Micro Project **CO5, S3**

(Tool: Oracle SQL/ SQL Developer/MySQL/MongoDB/NetBeans)

Develop Micro project based on business case scenario on use cases specified in Part-II.

TASK 1 - CONCEPTUAL DESIGN THROUGH FTR

Aim

To identify entities attributes and relationship from the usecase given and to draw the ERDiagram for the same.

Tasks and sample output

Using basic database design methodology and ER modeler, design Entity Relationship
Diagram of library management system.

1. a. Identifying the entities.

1. b. Identifying the attributes.

Sample Output

- Publisher(PublisherId, Year of publication, name)
- Admin(LoginId, password, name, staff_id)
- Report(UserId, Reg_no, Book_no, Issue/Return date)
- Book(authno, isbn number, title, edition, category, PublisherID, price)
    - Name is composite attribute of firstname and lastname.
    - Phone no is multi valued attribute.
- Reader(UserId, Email, address, phone no, name)

2. Identification of relationships, cardinality and type of relationship.

Sample Output

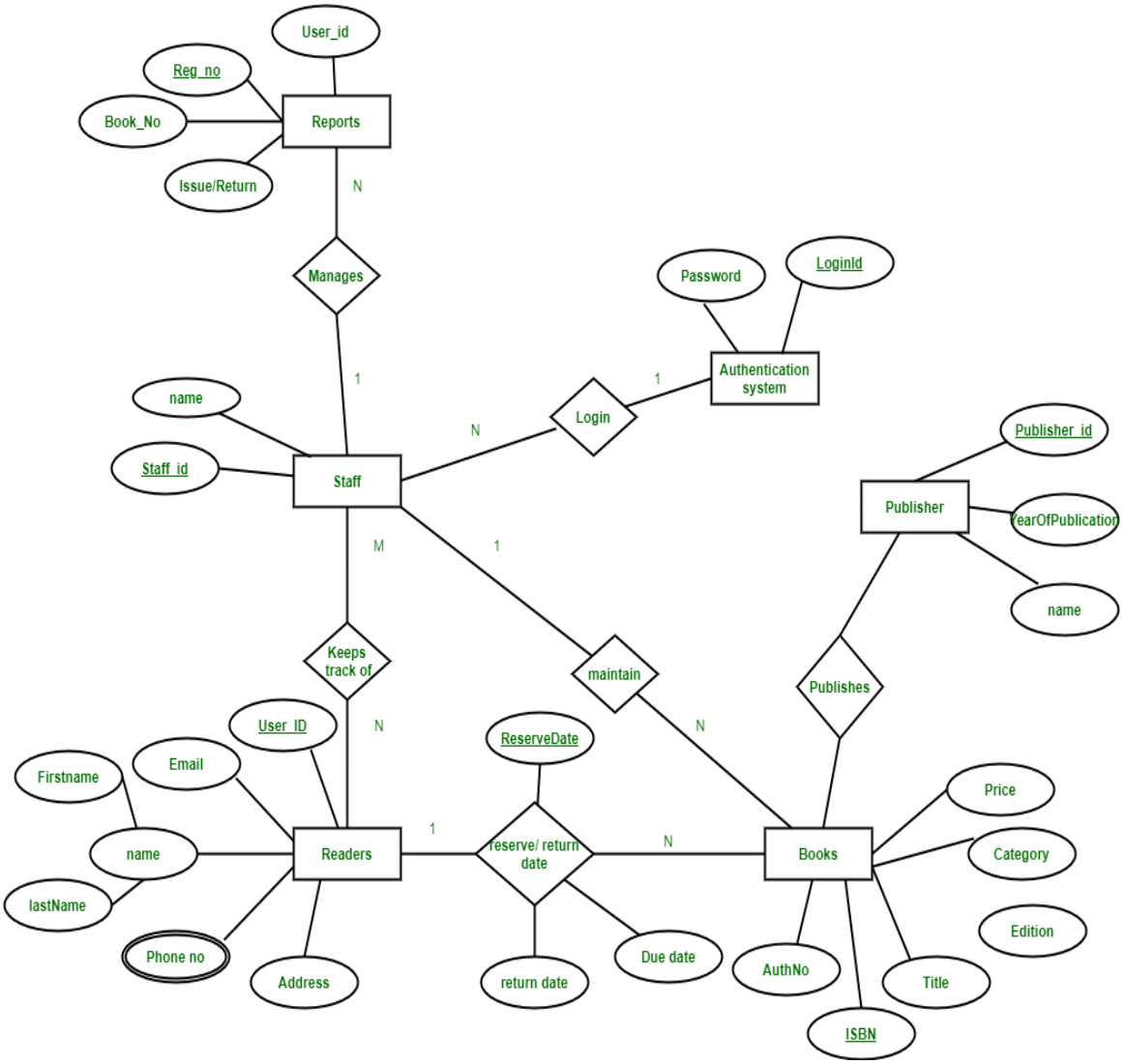
- A reader can reserve N books but one book can be reserved by only one reader. The relationship 1:N.
- A publisher can publish many books but a book is published by only one publisher. The relationship 1:N.
- Admin keeps track of readers. The relationship is M:N.
- Admin maintains multiple reports. The relationship 1:N.
- Admin maintains multiple Books. The relationship 1:N.
- Admin provides login to multiple staffs. The relation is 1:N.

3. Reframing the relations with keys and constraint.

Sample Output

- {isbn number} is the Primary Key for Book Entity.
- {authno, isbn number, title} – Super key of Book entity.
- {PublisherId} is the foreign key of Book entity.
- {authno,title} – Candidate key of Book entity.
- UserId is the Primary Key for Readers entity.
- PublisherID is the Primary Key for Publisher entity
- LoginID as Primary Key for Authentication system entity.
- Reg_no is the Primary Key of reports entity.

- Using creately, develop ER diagram.



RESULT: The task to create an E-R diagram of library management system is executed successfully.

TASK 2 - GENERATING DESIGN OF OTHER TRADITIONAL DATABASE MODEL

Title: Implementation of DDL commands of SQL with suitable examples.

- ∀ Create table
- ∀ Alter table
- ∀ Drop Table

Objective

To understand the different issues involved in the design and implementation of a database system

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 liketables and indexes.
- **PL/SQL:** PL/SQL can be used to developed programs for different applications.
  - ∀ **Oracle Forms**: This tool allows you to create a data entry screen along with the suitable menu objects. Thus it is the oracle forms tool that handles data gathering and data validation in a commercial application.
  - ∀ **Report Writer:** Report writer allows programmers to prepare innovative reports using data from the oracle structures like tables, views etc. It is the report writer tool that handles the reporting section of commercial 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 originally based upon Relational Algebra. Its scope includes data query and update, schema creation and modification, and data access control.

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

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

DATA TYPES

• **CHAR (Size):** This data type is used to store character strings values of fixed length. The size in bracketsdetermines the number of characters the cell can hold. The maximum number of character is 255 characters.

• **VARCHAR (Size) / VARCHAR2 (Size):** This data type is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.

• **NUMBER (P, S):** The NUMBER data type is used to store number (fixed or floating point). Number of virtually any magnitude may be stored up to 38 digits of precision. Number as large as 9.99 * 10 124. The precision (p) determines the number of places to the right of the decimal. If scale is omitted then the defaultis zero. If precision is omitted, values are stored with their original precision up to the maximum of 38 digits.

• **DATE**: This data type is used to represent date 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 date in the 24-Hours format. By default the time in a date field is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day the current month.

• **LONG:** This data type is used to store variable length character strings containing up to 2GB. Long datacan be used to store arrays of binary data in ASCII format. LONG values cannot be indexed, and the normalcharacter functions such as SUBSTR cannot be applied.

• **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 type can contain up to 2GB.

SQL language is sub-divided into several language elements, including:

- ∀ **Clauses**, which are in some cases optional, constituent components of statements and queries.
- ∀ **Expressions**, which can produce either scalar values or tables consisting of columns and rows of data.
- ∀ **Predicates** which specify conditions that can be evaluated to SQL three-valued logic (3VL) Boolean truthvalues and which are used to limit the effects of statements and queries, or to change program flow.
- ∀ **Queries** which retrieve data based on specific criteria.
- ∀ **Statements** which may have a persistent effect on schemas and data, or which may control transactions,program flow, connections, sessions, or diagnostics.
- ∀ SQL statements also include the **semicolon** (";") statement terminator. Though not required on everyplatform, it is defined as a standard part of the SQL grammar.
- ∀ **Insignificant white space** is generally ignored in SQL statements and queries, making it easier to formatSQL code for readability.

There are five types of SQL statements. They are:

• DATA DEFINITION LANGUAGE (DDL)

- DATA MANIPULATION LANGUAGE (DML)
- DATA RETRIEVAL LANGUAGE (DRL)
- TRANSATIONAL CONTROL LANGUAGE (TCL)
- DATA CONTROL LANGUAGE (DCL)

1.**DATA DEFINITION LANGUAGE (DDL):** The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structureand usage of four basic DDL commands:

1. CREATE 2. ALTER 3. DROP 4. RENAME

- **CREATE**

(a) **CREATE TABLE**: This is used to create a new relation (table)

syntax: CREATE TABLE <relation_name/table_name > (field_1 data_type(size),field_2 data_type(size),);

Example:

create table member(membno int, name varchar(20), department varchar(20),gender varchar(10));

Member

membno	name	department	gender
empty			

create table book(isbn int, title varchar(30), authors varchar(10),pagecount int, publisher varchar(10));

Book

isbn	title	authors	pagecount	publisher
empty				

create table borrowed(membno int, isbn int, issuedate date);

Borrowed

membno	isbn	issuedate
empty		

- **ALTER**
- **ALTER TABLE ...ADD**...: This is used to add some extra fields into existing relation.

Syntax: ALTER TABLE relation_name ADD (new field_1 data_type(size), new field_2data_type(size),..);

Example: SQL>Alter table borrowed add name VARCHAR(10)

Borrowed

membno	isbn	issuedate	name
empty			

- **DROP TABLE**: This is used to delete the structure of a relation. It permanently deletes the recordsin the table.

Syntax: DROP TABLE relation_name;

Example: SQL>DROP TABLE borrowed;

- **RENAME:** It is used to modify the name of the existing database object.

Syntax: RENAME TABLEold_relation_name TO new_relation_name;
Example: SQL> ALTER TABLE borrowed RENAME COLUMN name to membername;

Borrowed

membno	isbn	issuedate	membername
empty			

RESULT: The task to create, delete and alter the table are executed successfully.

TASK 3 - USING CLAUSES, OPERATORS AND FUNCTIONS IN QUERIES

Title : Implementation of DML commands using clauses, operators and functions in queries.

- Insert table
- Select table
- Update table
- Delete Table

Objective :

- To understand the different issues involved in the design and implementation of a database system

Theory:

DATA MANIPULATION LANGUAGE (DML): The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users duringthe routine operation of the database. Let's take a brief look at the basic DML commands:

1. INSERT 2. UPDATE 3. DELETE

- **INSERT INTO:** This is used to add records into a relation. These are three type of INSERT INTOqueries which are as

- **Inserting a single record**

Syntax: INSERT INTO < relation/table name>
(field_1,field_2……field_n)VALUES (data_1,data_2,.data_n);
Example: SQL> insert into member values(116,'Shan','CSE','male');

Inserting a single record

Member

membno	name	department	gender
111	jeni	IT	Female
111	Jaz	CSE	Female
113	John	CSE	Female
114	BEn	CSE	Female
115	Ann	CSE	Female
116	Shan	CSE	male

Borrowed

membno	isbn	issuedate
111	11110	2023-01-02
111	11111	2023-02-05
113	11110	2023-02-07

- **UPDATE-SET-WHERE**: This is used to update the content of a record in a relation.

Syntax: SQL>UPDATE relation name SET
Field_name1=data,field_name2=data, WHEREfield_name=data;

Example: SQL>UPDATE member SET sname = 'kumar' WHERE sno=1;

- **DELETE-FROM:** This is used to delete all the records of a relation but it will retain the structure ofthat relation.
- **DELETE-FROM**: This is used to delete all the records of relation.

Syntax:SQL>DELETE FROM relation_name;

Example: SQL>DELETE FROM std;

- **DELETE -FROM-WHERE**: This is used to delete a selected record from a relation.

Syntax: SQL>DELETE FROM relation_name WHERE condition;

Example: SQL>DELETE FROM student WHERE sno = 2;

5. **TRUNCATE**: This command will remove the data permanently. But structure will not be removed.

Difference between Truncate & Delete

By using truncate command data will be removed permanently & will not get back where as by usingdelete command data will be removed temporally & get back by using roll back command.

By using delete command data will be removed based on the condition where as by using truncatecommand there is no condition.

Truncate is a DDL command & delete

is a DML command.**Syntax:**

TRUNCATE TABLE <Table name>

Example: TRUNCATE TABLE

student;

Sample Queries and Output

- Retrieve member name end with letter 'n' and member no between 111 and 115.

Query:

```
SELECT first_name, last_name, salary FROM employees
  WHERE first_name LIKE '%m';
```

Output:

Output

name
John
BEn
Ann

- List the books where pagecount between 700 and 800 – between clause, and operator.

Query:

```
select * from book where pagecount between 700 and 800;
```

Output

isbn	title	authors	pagecount	publisher
11111	Operating Systems	Silberchatz	750	Mcgraw

- Find the records who has minimum number of pagecount – Aggregate

Query:

select min(pagecount) from book;

Output

min(pagecount)
330

- Find the records whose issue date greater than or equal to 07-02-2023

Query:

select * from borrowed where issuedate >='2023-02-07';

Output

membno	isbn	issuedate
113	11110	2023-02-07

- List the membno, but the same membno are listed ones.

Query:

select distinct membno from member;

membno
111
113
114
115
116

- Combine the rocords of member and book relation – Union

Query:

SELECT membno FROM member UNION SELECT membno FROM borrowed;

Output

membno
111
113
114
115
116
117
121

- Groupby the member number based on their gender and department.

Query

SELECT membno FROM member GROUP BY gender,department;

Output

membno
111
111
116
117

- Find the authors and their publication details using groupby and orderby clauses.

Query

SELECT authors,publisher, COUNT(*) no FROM book GROUP BY authors, publisher order by authors

Output

authors	publisher	No
MArk allen	Springer	1
Silberchatz	Mcgraw	1
Silberchatz	Springer	1
William Stallings	Ferrouzan	1

RESULT: The task to implement the DML commands are executed successfully.

TASK 4 - USING FUNCTIONS IN QUERIES AND WRITING SUBQUERIES

- A Subquery or Inner query or a Nested query is a query within another SQL query andembedded within the WHERE clause.
- A subquery is used to return data that will be used in the main query as a condition to furtherrestrict the data to be retrieved.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statementsalong with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow −

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are inthe main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use anORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators suchas the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, orNCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can beused within the subquery.

Subqueries with the SELECT Statement

Syntax:

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
  (SELECT column_name [, column_name ]
  FROM table1 [, table2 ]
  [WHERE])
```

- **Write the subqueries to display the isbn number for the records whose page count is less than 500.**

```
SELECT *
  FROM book
  WHERE isbn IN (SELECT isbn
      FROM book
      WHERE pagecount > 500) ;
```

Output

isbn	title	authors	pagecount	publisher
1111	OS	Silberchatz	850	springer

- **Select the title and authors whose pagecount is minimum using subquery and functions.**

Query
SELECT title,authors FROM book
WHERE pagecount = (SELECT MIN(pagecount) FROM book)

Output

title	authors
DBMS	Silberchatz

- **Write a subquery to return the values from multiple tables.**

Query
SELECT name, department FROM member WHERE membno == (SELECT isbn
FROM book)

Output

name	department
jeni	CSE

Subqueries with the INSERT Statement

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
  SELECT [ *|column1 [, column2 ]
  FROM table1 [, table2 ]
  [ WHERE VALUE OPERATOR ]
```

- **Insert all records in book table into bookduplicate table.**

```
INSERT INTO bookduplicate
  SELECT * FROM book
  WHERE title IN (SELECT title
  FROM book) ;
```

Bookduplicate

isbn	title	authors	pagecount	publisher
111	DBMS	Silberchatz	250	springer
1111	OS	Silberchatz	850	springer

Subqueries with the UPDATE Statement

Syntax

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
  (SELECT COLUMN_NAME
  FROM TABLE_NAME)
```

[WHERE)]

- **updates pagecount by 2 times in the book table for all the customers whose pagecount is greater than or equal to 300.**

UPDATE book SET pagecount = pagecount * 2
WHERE title IN (SELECT title FROM bookduplicate WHERE pagecount <= 300);

Book

isbn	title	authors	pagecount	publisher
111	DBMS	Silberchatz	500	springer
1111	OS	Silberchatz	850	springer

Subqueries with the DELETE Statement

DELETE FROM TABLE_NAME
[WHERE OPERATOR [VALUE]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[WHERE)]

- **Deletes the records from the member table for all the members whose gender is equal to male.**

DELETE FROM member WHERE gender IN (SELECT gender
FROM member WHERE gender == 'male');
Output

Member

membno	name	department	gender
111	jeni	CSE	female
112	jaz	IT	female

RESULT: The implementation of SQL commands using functions and subqueries is executed successfully.

TASK 5 - WRITING JOIN QUERIES, EQUIVALENT, AND/OR RECURSIVE QUERIES

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

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

Aim:

To implement and execute JOIN queries, equivalent queries, and recursive queries using a university database scenario.

Procedure:

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

- Create the database and tables (Students, Departments, Courses, Enrollments).
- Insert sample data.
- Write SQL queries using different types of **JOINs**.
- Write **equivalent queries** (different approaches to get the same result).
- Implement a **recursive query** (using WITH RECURSIVE).
- Display results and verify correctness.

Step1:

Syntax:

SELECT column 1, column 2, column 3...FROM table_name1, table_name2 WHEREtable_name1.column name = table_name2.columnname;

Types of Joins :

- Simple Join
- Self Join
- Outer Join

Simple Join:

It is the most common type of join. It retrieves the rows from 2 tables having a common column and isfurther 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 performs the join statement. It retrieves rows from both the tables provided they both have the same id as specified by the where clause. Since the where clause uses the comparison operator (=) to perform a join, it is said to be equijoin. It combines the matched rows of tables.It can be used as follows:

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

Non Equi-join:

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

Example:

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 give an alias name to the table in the 'from' clause and use it instead of the name throughout the query.

Self join:

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

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

Example:

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

=y.deptno);

Outer Join:

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

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 righttable.

SELECT column_name(s) FROM table1

LEFT JOIN table2 ON table1.column_name = table2.column_name;

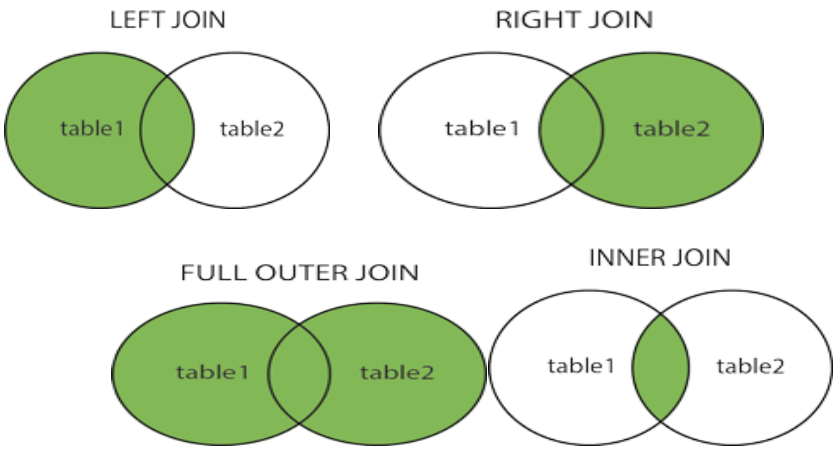
• RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from theleft table.

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 SELECTcolumn_name(s) FROM table1

FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;



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

Consider the following two tables –

Inner Join

The INNER JOIN keyword selects all rows from both tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies.

Syntax

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

Query

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

Output

membno	name
111	jeni

Left Join

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2).

Syntax

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

Query

```
SELECT member.NAME, borrowed.membno FROM member
```

LEFT JOIN borrowed ON borrowed.membno = member.membno;

SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1).

Syntax

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

SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Tip: FULL OUTER JOIN and FULL JOIN are the same.

Syntax :

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

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 x2 (result) AS (
   SELECT 1
   UNION ALL
   SELECT result*2 FROM x2)
SELECT * FROM x2 LIMIT 10;
```

Output

```
result
--------
     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)

Result:

The implementation of SQL commands using Joins and recursive queries are executed successfully.

Task -6

PL/SQL Procedures, functions, Loops

Aim:

To implement PL/SQL Procedures, Functions and loops on Number theory and business scenarios.

Procedure:

PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by 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, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, 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 handle 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 is22

PL/SQL procedure successfully completed.
```

Dynamic Input:

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

PL/SQL procedure successfully completed.
```

```
DECLARE
  hid number(3) := 100;
BEGIN
  IF ( hid = 10 ) THEN
    dbms_output.put_line('Value of hid is 10' );
  ELSIF ( hid = 20 ) THEN
    dbms_output.put_line('Value of hid is 20' );
  ELSIF ( hid = 30 ) THEN
    dbms_output.put_line('Value of hid is 30' );
  ELSE
     dbms_output.put_line('None of the values is matching');
  END IF;
  dbms_output.put_line('Exact value of hid is: '|| hid );
END;
/
None of the values is matching
Exact value of hid is: 100

PL/SQL procedure 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.
```

Sample program for only procedure:

```
SQL> create or replace procedure csinformation
  2  (c_id in number, c_name in varchar2)
  3  is
  4  begin
  5  dbms_output.put_line('ID: '||c_id);
  6  dbms_output.put_line('Name: '||c_name);
  7  end;
  8  /

Procedure created.

SQL> exec csinformation(101,'raam');

PL/SQL procedure successfully completed.

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

PL/SQL procedure successfully completed.
```

Sample program for only function:

```
SQL>create or replace function csinformation
(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;
```

/
Function created.

```
SQL> declare
  2  mesg varchar2(200);
  3  begin
  4  mesg :=csinformation2(102,'raam');
  5  dbms_output.put_line(mesg);
  6  end;
  7  /
vehicle available
```

```
SQL> declare
  2  mesg varchar2(200);
  3  begin
  4  mesg :=csinformation2(206,'raam');
  5  dbms_output.put_line(mesg);
  6  end;
  7  /
No vehicle available

PL/SQL procedure successfully completed.
```

Result:Hence the pL/SQL ,using loops and queries are successfully completed.

TASK 7

Sample program for only Loops

- To print prime number customer id using loopsTo check the given customer booking number is Armstrong number.

```
SQL> declare
  2  bk number(5);
  3  s number:=0;
  4  r number;
  5  len number;
  6  m number;
  7  begin
  8  bk:=&bk;
  9  m:=bk;
 10  len:=length(to_char(bk));
 11  while bk>0
 12  loop
 13  r:=mod (bk,10);
 14  s:=s+power(r,len);
 15  bk:=trunc(bk/10);
 16  end loop;
 17  if
 18  m=s
 19  then
 20  dbms_output.put_line('given number is armstrong');
 21  else
 22  dbms_output.put_line('given number is not an armstrong');
 23  end if;
 24  end;
 25  /
Enter value for bk: 234
old   8: bk:=&bk;
new   8: bk:=234;
given number is not an armstrong
```

```
Enter value for bk: 1634
old   8: bk:=&bk;
new   8: bk:=1634;
given number is armstrong

PL/SQL procedure successfully completed.
```

Result:Hence the task 7 is successfully verified.

TASK 8: (Tool: GU/ Table Normalization Tool, ALM:Jigsaw) CO3, K3

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, Manager_ID, Hire_Date, Salary
- Department -> Manager_ID
- Manager_ID -> Name

Step 2: Convert to 1NF

1. Eliminate repeating groups or arrays (none in this example).
2. Create separate tables for each repeating group (none in this example).

Step 3: Convert to 2NF

1. Ensure each non-key attribute depends on the entire primary key.
2. Move non-key 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 (Manager_ID, Name).
- Update Department table: Department (Department_ID, Manager_ID).

Step 5: Convert to BCNF

1. Ensure every determinant is a candidate key.
2. Check for overlapping candidate keys.
3. Decompose relations to eliminate redundancy.

- No further decomposition needed.

Using Griffith Tool

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

Griffith Tool Steps

1. Create a new project in Griffith.
2. Define the relational schema and FDs.
3. Run the "Dependency Graph" tool.
4. Analyze the graph for normalization issues.
5. Apply transformations using the "Normalize" tool.
6. Verify BCNF compliance using the "BCNF Check" tool.

Normalized Schema

1. Employee (Employee_ID, Name, Department_ID, Job_Title, Hire_Date, Salary).
2. Department (Department_ID, Manager_ID).
3. Manager (Manager_ID, Name).

Result:Normalizing databases using functional dependencies upto BCNF is successfully completed.

TASK 9: Backing up and recovery in databases CO4, K3

Perform following backup and recovery scenarios.
a. Recovering a NOARCHIVELOG Database with Incremental Backups
b. Restoring the Server Parameter File
c. Performing Recovery with a Backup Control File

Scenario 1: Recovering a NOARCHIVELOG Database with Incremental Backups

```
-- Step 1: Backup Database
BACKUP DATABASE [database_name] TO DISK = 'backup_file.bak' WITH
NOFORMAT, NOINIT,  NAME = 'Full Database Backup', SKIP, REWIND,
NOUNLOAD,  STATS = 10

-- Step 2: Create Incremental Backup
BACKUP DATABASE [database_name] TO DISK = 'incremental_backup.bak'
WITH DIFFERENTIAL, NOFORMAT, NOINIT,  NAME = 'Incremental Database
Backup', SKIP, REWIND, NOUNLOAD,  STATS = 10

-- Step 3: Simulate Data Loss
-- 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 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
STARTUP MOUNT
RESTORE CONTROLFILE FROM FILE = 'controlfile.bak';
ALTER CONTROLFILE REUSE;

-- Step 4: Recover Database
RECOVER DATABASE USING BACKUP CONTROLFILE;

-- Step 5: Open Database
ALTER DATABASE OPEN RESETLOGS;
```

SQL Server Commands:
- BACKUP DATABASE
- RESTORE DATABASE
- RECOVER DATABASE
- ALTER DATABASE
- BACKUP SERVER PARAMETER FILE
- RESTORE SERVER PARAMETER FILE
- BACKUP CONTROLFILE
- RESTORE CONTROLFILE

Result:Hence Backing up and recovery in databases is successfully verified.

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, querying, finding and removing operations.

STEPS:

Step 1)install Mongo db using following 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 MongoDB Shell binary's location to your PATH environment variable:
Open the Control Panel.
In the System and Security category, click System.
Click Advanced system settings. The System Properties modal displays.
Click Environment Variables.
In the System variables section, select path and click Edit. The Edit environment variable modal displays.
Click New and add the filepath to your mongosh binary.
Click OK to confirm your changes. On each other modal, click OK to confirm your changes.
To confirm that your PATH environment variable is correctly configured to find mongosh, open a command prompt and enter the mongosh --help command.
If your PATH is configured correctly, a list of valid commands displays.

Step 4)Open mongo shell 4.0 from
c:\programfiles\mongoDB\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,u
om:"cm"}})
{
"acknowledged" : true,
"insertedId" : ObjectId("627d13acc73990c074e6397c")
}
> db.mylab.find({item:"canvas"})
```

```
{ "_id" : ObjectId("627d13acc73990c074e6397c"), "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:2
1,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"}}])
{
"acknowledged" : 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("627d1598c73990c074e6397e"), "item" : "mat", "qty" : 85 }
{ "_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("627d1598c73990c074e6397e"), "item" : "mat", "qty" : 85 }
{
"_id" : ObjectId("627d1598c73990c074e6397f"),
"item" : "mousepad",
"qty" : 25
}
> db.mylab.find({item:"canvas"}).pretty().sort({item:-1})
{
"_id" : ObjectId("627d13acc73990c074e6397c"),
"item" : "canvas",
"qty" : 100,
"tags" : [
"cotton"
    ],
"size" : {
"h" : 28,
"w" : 35.5,
"uom" : "cm"
```

```
        }
}
> db.mylab.deleteOne({item:"journal"}
...
...
> db.mylab.find({},{item:1,qty:1}).pretty()
{
"_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}
```

Result:

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

TASK 11- CRUD OPERATIONS IN GRAPH DATABASES

AIM:

To perform CRUD operations like creating, inserting, querying, finding, deleting 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 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 relationship using the CREATE clause.

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

- **Creating a Relationship Between the Existing Nodes**

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

Syntax

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

```
MATCH (a:LabeofNode1), (b:LabeofNode2)
 WHERE a.name = "nameofnode1" AND b.name = " nameofnode2"
CREATE (a)-[: Relation]->(b)
RETURN a,b
```

- **Deleting a Particular Node**

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

Syntax

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

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

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

```
create(n:student{Sid: "VTU14500",
Sname:"John",
deptname:"CSE" }
)
```

OUTPUT

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

```
Create(n:student {Sid: "VTU14501",
Sname:"Dharsana",
deptname:"EEE"})
```

OUTPUT

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

```
Create(w:student { Sid: "VTU14502",
Sname:"vijay",
deptname:"CSE"
})
```

OUTPUT

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

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

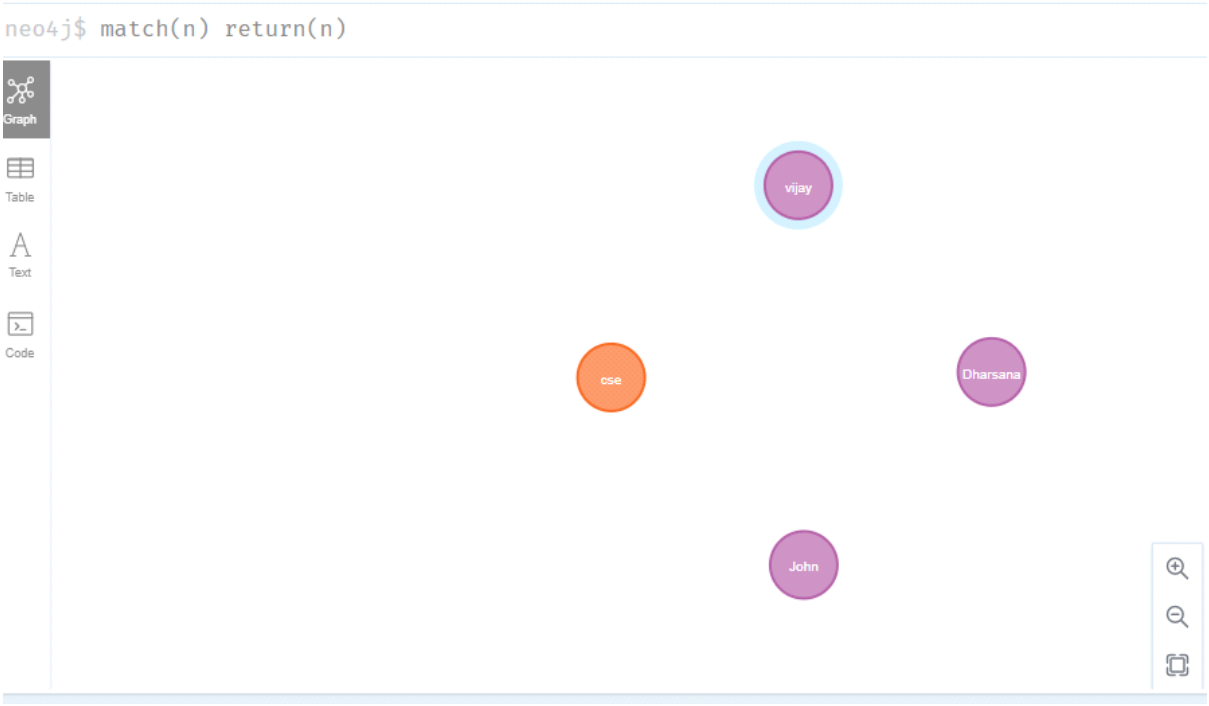
OUTPUT:

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

Select all the nodes in your database using match command

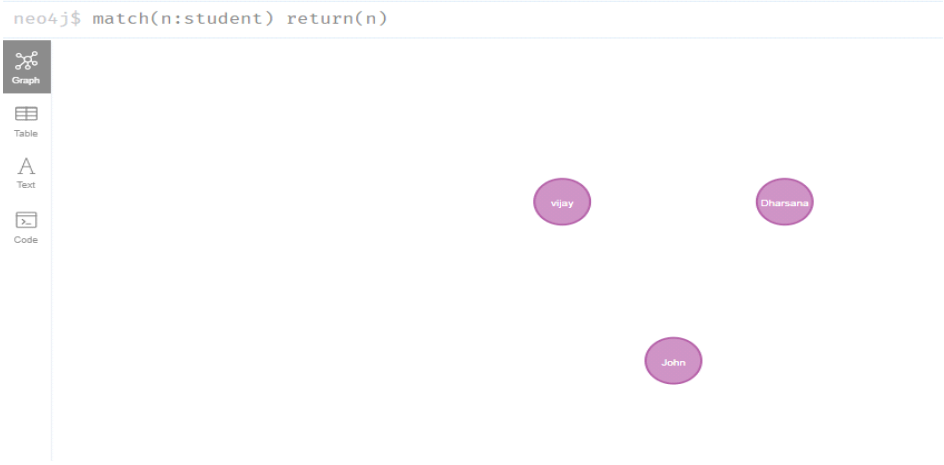
- **match(n) return(n)**

OUTPUT



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

OUTPUT:

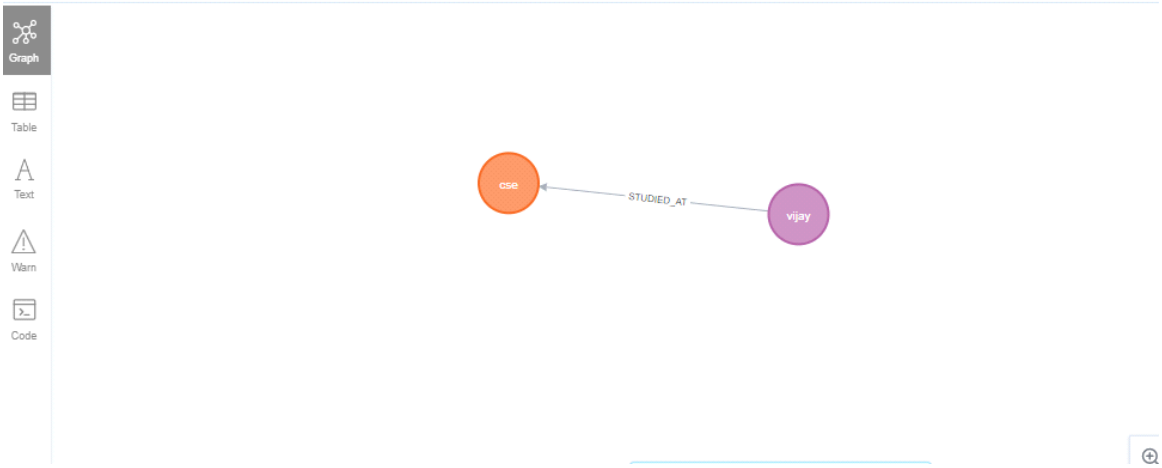


a. **Create relationship between student and cse .**

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

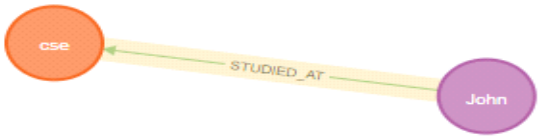
OUTPUT:

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

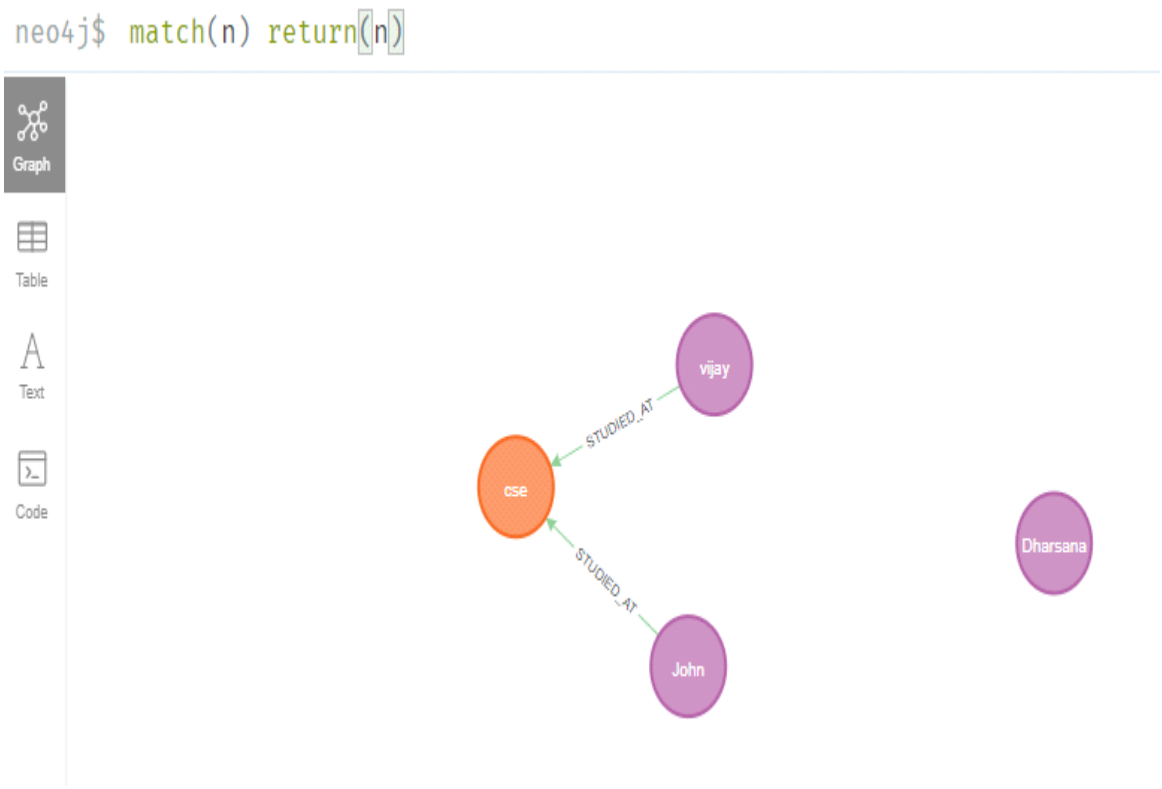


MATCH(s:student),(d:dept) WHERE s.Sname ='John' AND d.deptname='cse'
CREATE(s)-[st:STUDIED_AT]->(d)
return s,d

OUTPUT:



match(n) return(n

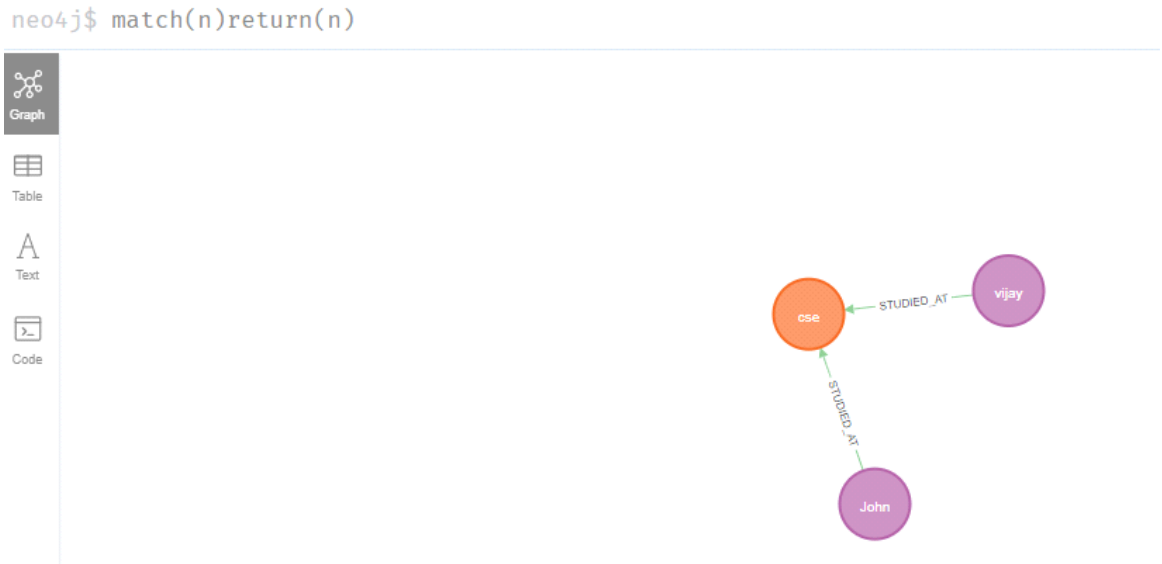


a. Delete a node from student

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

OUTPUT:

Deleted 1 node, complaeted after 10834 ms.



Result

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

TASK 12: MINI PROJECT

TITLE : BLOOD BANK MANAGEMENT

SYSTEM 1.ER Diagram:

Aim : To design the conceptual model for a Blood Bank Management System using the Entity-Relationship (ER) model.

E-R Diagram

Entity-Relationship model:

- ER model stands for an Entity-Relationship model. It is a highlevel data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data

WEAK ENTITY: An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.

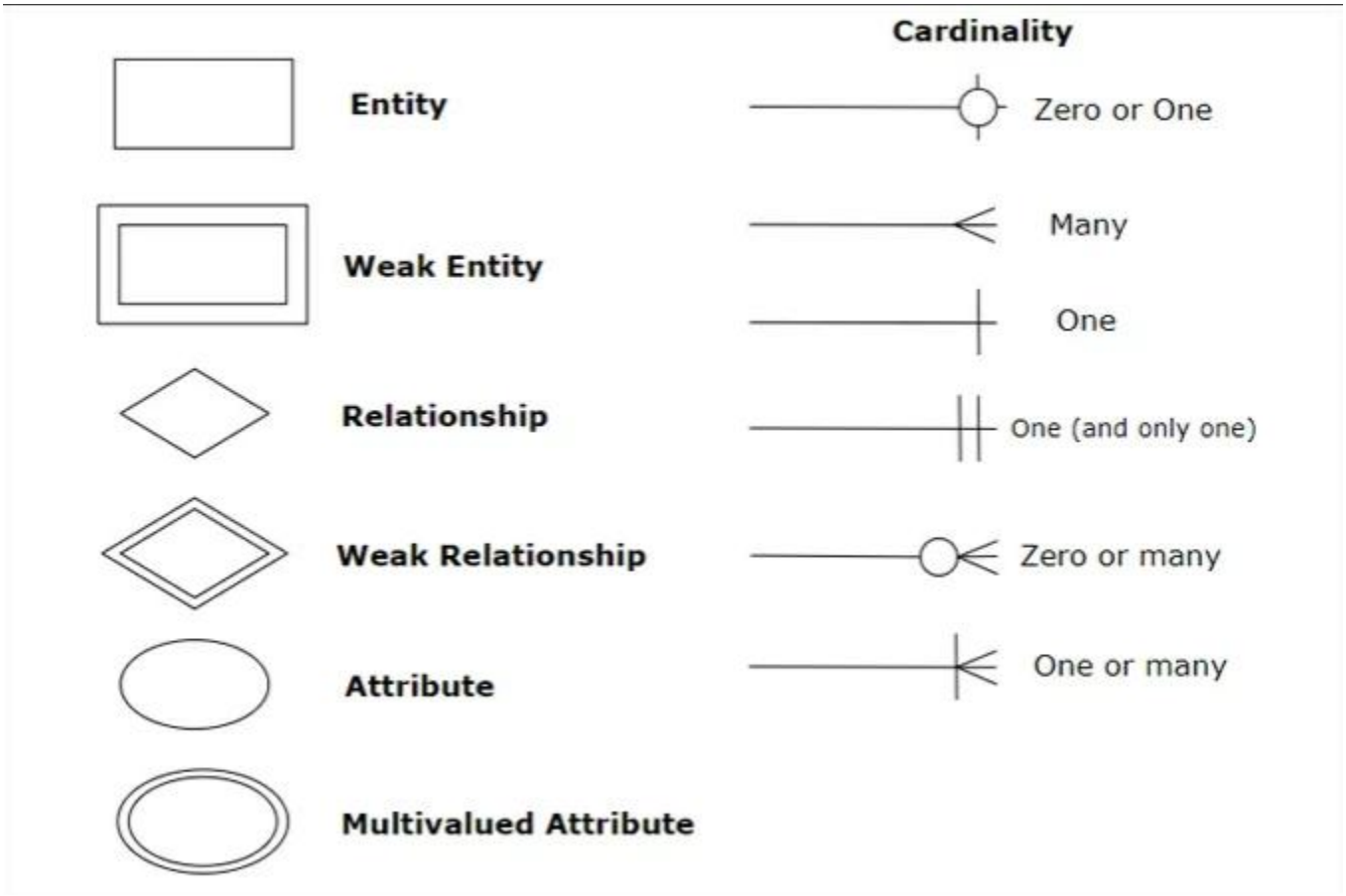
ATTRIBUTE: The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute

KEY ATTRIBUTE: The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.

COMPOSITE ATTRIBUTE: An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.

MULTI VALUED ATTRIBUTE : An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

DERIVED ATTRIBUTE: Attributes which are derived from other attributes



ER-MODEL FOR BLOOD BANK MANAGEMENT SYSTEM: Entity Attributes

Patient patient_ID (PK), patient_name, gender, date_of_birth, patient_age, blood_group, contact, address (door_no, street, state)

Blood blood_ID (PK), blood_group, cost, patient_ID (FK), blood_bank_ID

(FK) Hospital hospital_ID (PK), hospital_name, location, door_no, city,

state

Blood_Bank blood_bank_ID (PK), name, street, state,

no_of_blood_types Blood_Bank_BloodGroup blood_bank_ID (FK),

blood_group

Registers_For patient_ID (FK), blood_bank_ID (FK), registration_date

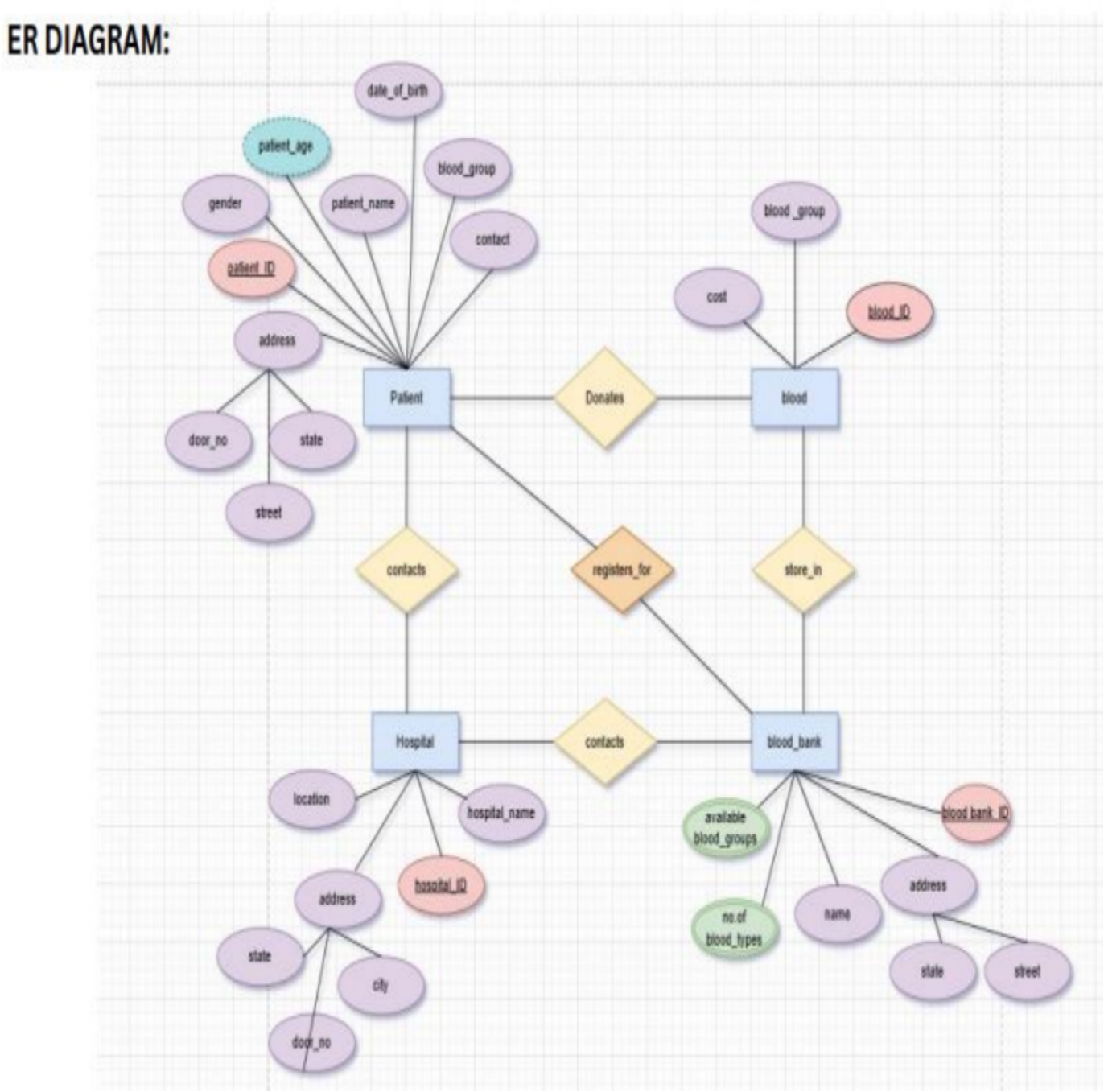
Patient_Contacts_Hospital patient_ID (FK), hospital_ID (FK), contact_date

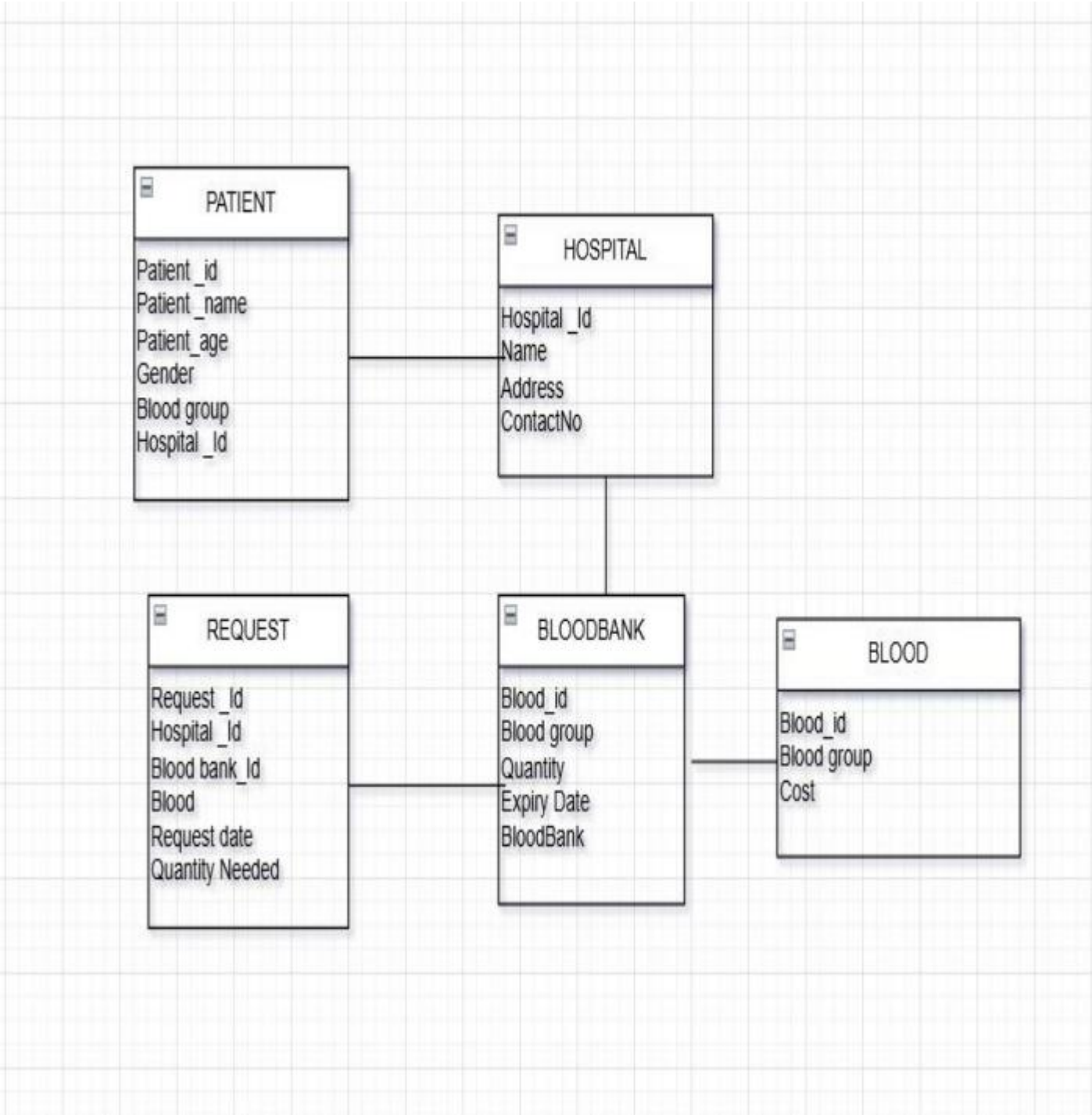
Hospital_Contacts_BloodBank hospital_ID (FK), blood_bank_ID (FK), contact_date

Relationships:

1. Patient donates Blood → One-to-Many
2. Blood stored in Blood_Bank → Many-to-One
3. Patient registers in Blood_Bank → Many-to-Many
4. Patient contacts Hospital → Many-to-Many
5. Hospital contacts Blood_Bank → Many-to-Many

ER DIAGRAM:





Result: Thus, the creating er diagram is completed successfully.

SQL QUERIES FOR BLOOD BANK MANAGEMENT SYSTEM:

AIM: To execute SQL commands for creating tables and performing relational operations,

aggregate functions, and join queries for the Blood Bank Management

System. Sql queries are :

```
CREATE TABLE Patient ( patient_ID INT PRIMARY KEY, patient_name
VARCHAR(50), gender VARCHAR(10), date_of_birth DATE, patient_age INT,
blood_group VARCHAR(5), contact

VARCHAR(15), door_no VARCHAR(10), street VARCHAR(50), state VARCHAR(30) );

 CREATE TABLE Blood_Bank ( blood_bank_ID INT PRIMARY KEY, name
VARCHAR(50), street VARCHAR(50), state VARCHAR(30), no_of_blood_types INT );

 CREATE TABLE Blood ( blood_ID INT PRIMARY KEY, blood_group VARCHAR(5),
 cost

DECIMAL(8,2), patient_ID INT, blood_bank_ID INT, FOREIGN KEY (patient_ID)
REFERENCES

Patient(patient_ID), FOREIGN KEY (blood_bank_ID)
REFERENCES Blood_Bank(blood_bank_ID) );

CREATE TABLE Hospital ( hospital_ID INT PRIMARY KEY, hospital_name
VARCHAR(50),

location VARCHAR(50), door_no VARCHAR(10), city VARCHAR(50), state
VARCHAR(30) );

 CREATE TABLE Registers_For ( patient_ID INT, blood_bank_ID INT,
registration_date DATE, PRIMARY KEY (patient_ID, blood_bank_ID), FOREIGN KEY
(patient_ID) REFERENCES

Patient(patient_ID), FOREIGN KEY (blood_bank_ID)
REFERENCES Blood_Bank(blood_bank_ID) );

CREATE TABLE Patient_Contacts_Hospital ( patient_ID INT, hospital_ID INT,
contact_date DATE, PRIMARY KEY (patient_ID, hospital_ID), FOREIGN KEY
(patient_ID) REFERENCES

Patient(patient_ID), FOREIGN KEY (hospital_ID) REFERENCES Hospital(hospital_ID)
);

 CREATE TABLE Hospital_Contacts_BloodBank ( hospital_ID INT, blood_bank_ID INT,

      contact_date DATE, PRIMARY KEY (hospital_ID, blood_bank_ID), FOREIGN KEY
                                                                  (hospital_ID)

REFERENCES Hospital(hospital_ID), FOREIGN KEY (blood_bank_ID)
REFERENCES Blood_Bank(blood_bank_ID);
```

INSERTING INTO TABLES:

Inserting into Patient table

INSERT INTO Patient (patient_ID, patient_name, gender, date_of_birth, patient_age, blood_group, contact, door_no, street, state)

VALUES (1, 'John Doe', 'Male', '1990-01-01', 32, 'A+', '1234567890', '123', 'Main Street', 'New York'),

(2, 'Jane Doe', 'Female', '1995-06-01', 27, 'O-', '9876543210', '456', 'Park Avenue', 'California'),

(3, 'Bob Smith', 'Male', '1980-03-01', 42, 'B+', '5551234567', '789', 'Broadway', 'Florida'); -

- Inserting into Blood_Bank table --

Inserting into Blood table

INSERT INTO Blood (blood_ID, blood_group, cost, patient_ID,

blood_bank_ID) VALUES (1, 'A+', 100.00, 1, 1),

(2, 'O-', 150.00, 2, 2), (3,

'B+', 120.00, 3, 3);

-- Inserting into Hospital table

INSERT INTO Hospital (hospital_ID, hospital_name, location, door_no, city,

state) VALUES (1, 'New York Hospital', 'New York', '1234', 'New York City', 'New York'),

(2, 'California Hospital', 'California', '5678', 'Los Angeles', 'California'),

(3, 'Florida Hospital', 'Florida', '9012', 'Miami', 'Florida'); -

- Inserting into Registers_For table

INSERT INTO Registers_For (patient_ID, blood_bank_ID,

registration_date) VALUES (1, 1, '2022-01-01'),

(2, 2, '2022-06-01'), (3,

3, '2022-03-01'); --

Inserting into Patient_Contacts_Hospital table

INSERT INTO Patient_Contacts_Hospital (patient_ID, hospital_ID,

contact_date) VALUES (1, 1, '2022-01-05'),

(2, 2, '2022-06-05'),

(3, 3, '2022-03-05');

-- Inserting into Hospital_Contacts_BloodBank table

INSERT INTO Hospital_Contacts_BloodBank (hospital_ID, blood_bank_ID, contact_date) VALUES (1, 1, '2022-01-10'),

(2, 2, '2022-06-10'), (3, 3, '2022-03-10');

patient_ID	patient_name	gender	date_of_birth	patient_age	blood_group	contact	door_no	street	state
1	John Doe	Male	1990-01-01	32	A+	1234567890	123	Main Street	New York
2	Jane Doe	Female	1995-06-01	27	O-	9876543210	456	Park Avenue	California
3	Bob Smith	Male	1980-03-01	42	B+	5551234567	789	Broadway	Florida

Output

blood_bank_ID	name	street	state	no_of_blood_types
1	Red Cross Blood Bank	101 Red Cross Street	New York	8
2	Blood Donors Association	202 Blood Donors Avenue	California	8
3	Local Blood Bank	303 Local Street	Florida	8

Output

blood_ID	blood_group	cost	patient_ID	blood_bank_ID
1	A+	100	1	1
2	O-	150	2	2
3	B+	120	3	3

Output

hospital_ID	hospital_name	location	door_no	city	state
1	New York Hospital	New York	1234	New York City	New York
2	California Hospital	California	5678	Los Angeles	California
3	Florida Hospital	Florida	9012	Miami	Florida

Output

patient_ID	blood_bank_ID	registration_date
1	1	2022-01-01
2	2	2022-06-01
3	3	2022-03-01

Output

patient_ID	hospital_ID	contact_date
1	1	2022-01-05
2	2	2022-06-05
3	3	2022-03-05

Output

hospital_ID	blood_bank_ID	contact_date
1	1	2022-01-10
2	2	2022-06-10
3	3	2022-03-10

SQLAGGREATE FUNCTIONS:

Input

```
SELECT COUNT(*) FROM Patient;
SELECT AVG(blood_group) AS Patient_age FROM Patient;
SELECT MAX(Patient_age) FROM Patient;
```

Output | Available Tables

COUNT(*)
3

Patient_age
0

MAX(Patient_age)
42

Input

```
SELECT MIN(Cost)  FROM Blood;
SELECT MIN(Patient_age)  FROM Patient;
select count(cost) from Blood;
```

Output | Available Tables

MIN(Cost)

MIN(Patient_age)
27

count(cost)
0

NESTED QUERIES FOR BLOOD BANK MANAGEMENT SYSTEM: Patients Who Contacted Hospitals That Contacted Blood Banks

```
SELECT patient_name
FROM Patient
WHERE patient_ID IN (
    SELECT patient_ID
    FROM Patient_Contacts_Hospital
    WHERE hospital_ID IN (
        SELECT hospital_ID
        FROM Hospital_Contacts_BloodBank
    )
);
```

Output

patient_name
John Doe
Jane Doe
Bob Smith

```
SELECT hospital_name
FROM Hospital
WHERE state = (
    SELECT state
    FROM Patient
    WHERE patient_ID = (
        SELECT patient_ID
        FROM Blood
        ORDER BY cost DESC
        LIMIT 1
    )
);
```

Output

hospital_name
California Hospital

Blood Banks Registered by Patients Who Bought Blood Costing More Than ₹120

```
SELECT name
FROM Blood_Bank
WHERE blood_bank_ID IN (
    SELECT blood_bank_ID
    FROM Registers_For
    WHERE patient_ID IN (
        SELECT patient_ID
        FROM Blood
        WHERE cost > 120
    )
);
```

Output

name
Blood Donors Association

Hospitals Located in the Same State as the Patient Who Bought the Most Expensive Blood

```
SELECT hospital_name
FROM Hospital
WHERE state = (
    SELECT state
    FROM Patient
    WHERE patient_ID = (
        SELECT patient_ID
        FROM Blood
        ORDER BY cost DESC
        LIMIT 1
    )
);
```

Output

hospital_name
California Hospital

Result: Hence the SQL Queries are successfully completed.

JOIN QUERIES :

INNER JOIN: Patients and Their Blood Purchases

```
SELECT p.patient_name, b.blood_group, b.cost
FROM Patient p
INNER JOIN Blood b ON p.patient_ID = b.patient_ID;
```

Output | Available Tables

patient_name	blood_group	cost
John Doe	A+	100
Jane Doe	O-	150
Bob Smith	B+	120

LEFT JOIN: All Patients and Any Blood Purchased

```
SELECT p.patient_name, b.blood_group, b.cost
FROM Patient p
LEFT JOIN Blood b ON p.patient_ID = b.patient_ID;
```

Output | Available Tables

patient_name	blood_group	cost
John Doe	A+	100
Jane Doe	O-	150
Bob Smith	B+	120

RIGHT JOIN

Patients and Blood Purchases (including unmatched blood records)

```
SELECT b.blood_ID, b.blood_group, b.cost, p.patient_name
FROM Health_Patient p
RIGHT JOIN Health_Blood b ON p.patient_ID = b.patient_ID;
```

Output | Available Tables

Error: RIGHT and FULL OUTER JOINs are not currently supported

TRIGGERS:

Auto-calculate patient age on insert

```
CREATE TRIGGER trg_log_patient_insert
AFTER INSERT ON Health_Patient
FOR EACH ROW
BEGIN
    INSERT INTO Health_PatientLog (patient_ID)
    VALUES (NEW.patient_ID);
END;
```

Output

SQL query successfully executed. However, the result set is empty.

Create a trigger to log each donation

```
CREATE TRIGGER trg_log_blood_donation1
AFTER INSERT ON Health_BloodDonation
FOR EACH ROW
BEGIN
    INSERT INTO Health_BloodDonationLog (donation_ID)
    VALUES (NEW.donation_ID);
END;
```

Output

SQL query successfully executed. However, the result set is empty.

```
CREATE TRIGGER trg_log_blood_donation
AFTER INSERT ON Health_BloodDonation
FOR EACH ROW
BEGIN
    INSERT INTO Health_BloodDonationLog (donation_ID)
    VALUES (NEW.donation_ID);
END;

INSERT INTO Health_BloodDonation (donor_name, blood_group)
VALUES ('Alice', 'A+'), ('Bob', 'O-');
SELECT * FROM Health_BloodDonationLog;
```

Output

log_ID	donation_ID	log_time
1	1	2025-10-18 17:36:04
2	1	2025-10-18 17:36:04
3	2	2025-10-18 17:36:04
4	2	2025-10-18 17:36:04

RESULT: Hence the Joins and Triggers are successfully completed

Normalization

Normalization in the context of databases refers to the process of organizing

data in a database efficiently. The goal is to reduce data redundancy and dependency by organizing fields and table of a 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 (1NF):

- 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 1NF.
- 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

Steps to follow for doing normalisation using Griffith normalisation process:

Step1: search for Griffith University normalisation tool in web browser

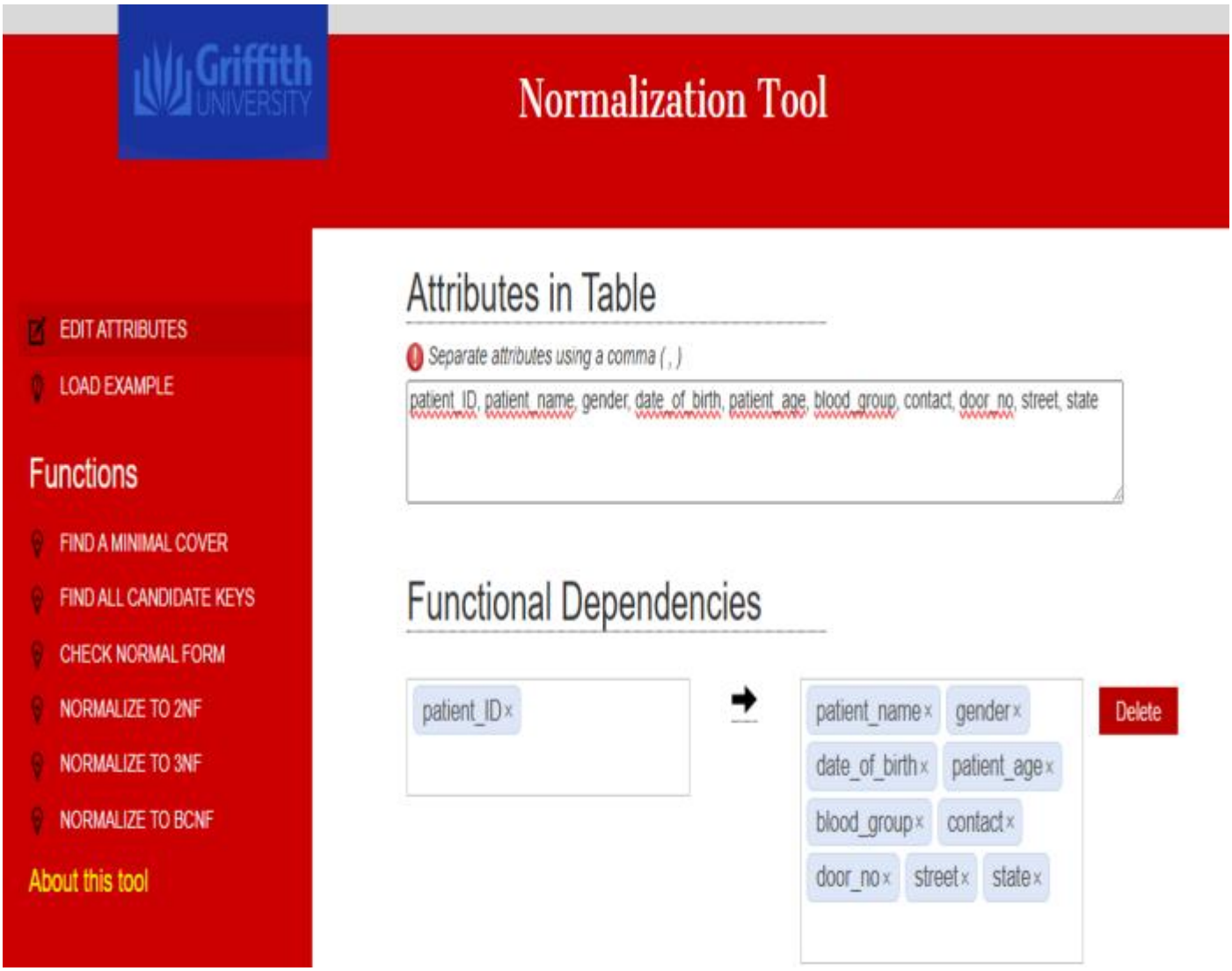
Step2: After opening the tool enter the attributes of the entity . Make sure to separate the attributes usingcommas in between them.

Step3: Add the dependencies of the attributes

Do as per your entity and add the dependencies as shown in the fig.

Step4: In the left if the window below functions on check normal form.We will get

the screen shown above the normal form of the given attributes is checked (BCNF) And the following steps are displayed below:



Griffith UNIVERSITY

Normalization Tool

EDIT ATTRIBUTES

LOAD EXAMPLE

Functions

FIND A MINIMAL COVER

Find the minimal cover

FIND ALL CANDIDATE KEYS

CHECK NORMAL FORM

NORMALIZE TO 2NF

NORMALIZE TO 3NF

NORMALIZE TO BCNF

About this tool

Find Minimal Cover

patient_ID → patient_name

patient_ID → gender

patient_ID → date_of_birth

patient_ID → patient_age

patient_ID → blood_group

patient_ID → contact

patient_ID → door_no

patient_ID → street

patient_ID → state

Show Steps

Step 1: Rewrite the FD into those with only one attribute on RHS. We obtain:

patient_ID → patient_name

patient_ID → gender

patient_ID → date_of_birth

patient_ID → patient_age

patient_ID → blood_group

patient_ID → contact

patient_ID → door_no

patient_ID → street

patient_ID → state

Step 2: Remove trivial FDs (those where the RHS is also in the LHS). We obtain:

patient_ID → patient_name

patient_ID → gender

patient_ID → date_of_birth

patient_ID → patient_age

patient_ID → blood_group

patient_ID → contact

patient_ID → state

Step 3: Minimize LHS of each FD. We obtain:

patient_ID → patient_name

patient_ID → gender

patient_ID → date_of_birth

patient_ID → patient_age

patient_ID → blood_group

patient_ID → contact

patient_ID → door_no

patient_ID → street

patient_ID → state

Step 4: Remove redundant FDs (those that are implied by others). We obtain:

patient_ID → patient_name

patient_ID → gender

patient_ID → date_of_birth

patient_ID → patient_age

patient_ID → blood_group

patient_ID → contact

patient_ID → door_no

patient_ID → street

EDIT ATTRIBUTES

LOAD EXAMPLE

Functions

FIND A MINIMAL COVER

FIND ALL CANDIDATE KEYS

CHECK NORMAL FORM

NORMALIZE TO 2NF

NORMALIZE TO 3NF

NORMALIZE TO BCNF

About this tool

Find Candidate Keys

Candidate Keys Found

- patient_ID

Show Steps

Step 1: Find the minimal cover of FDs, which contains
patient_ID -> patient_name
patient_ID -> gender
patient_ID -> date_of_birth
patient_ID -> patient_age
patient_ID -> blood_group
patient_ID -> contact
patient_ID -> door_no
patient_ID -> street
patient_ID -> state

Step 2: Find the set of attributes not on the RHS of any FD, which is NotOnRHS = {patient_ID}. Every CK must contain these attributes.

Step 3: NotOnRHS is a superkey, so it is the only candidate key

Griffith UNIVERSITY

Normalization Tool

- EDIT ATTRIBUTES
- LOAD EXAMPLE

Functions

- FIND A MINIMAL COVER
- FIND ALL CANDIDATE KEYS
- CHECK NORMAL FORM
- NORMALIZE TO 2NF
- NORMALIZE TO 3NF
- NORMALIZE TO BCNF

About this tool

Check Normal Form

2NF

The table is in 2NF

3NF

The table is in 3NF

BCNF

The table is in BCNF

Show Steps

NORMALIZE TO 2NF

NORMALIZE TO 3NF

NORMALIZE TO BCNF

About this tool

BCNF

The table is in BCNF

Show Steps

2NF

find all candidate keys. The candiates keys are { patient_ID}, The set of key attributes are: { patient_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: patient_ID ->
patient_name,gender,date_of_birth,patient_age,blood_group,contact,door_no,street,state

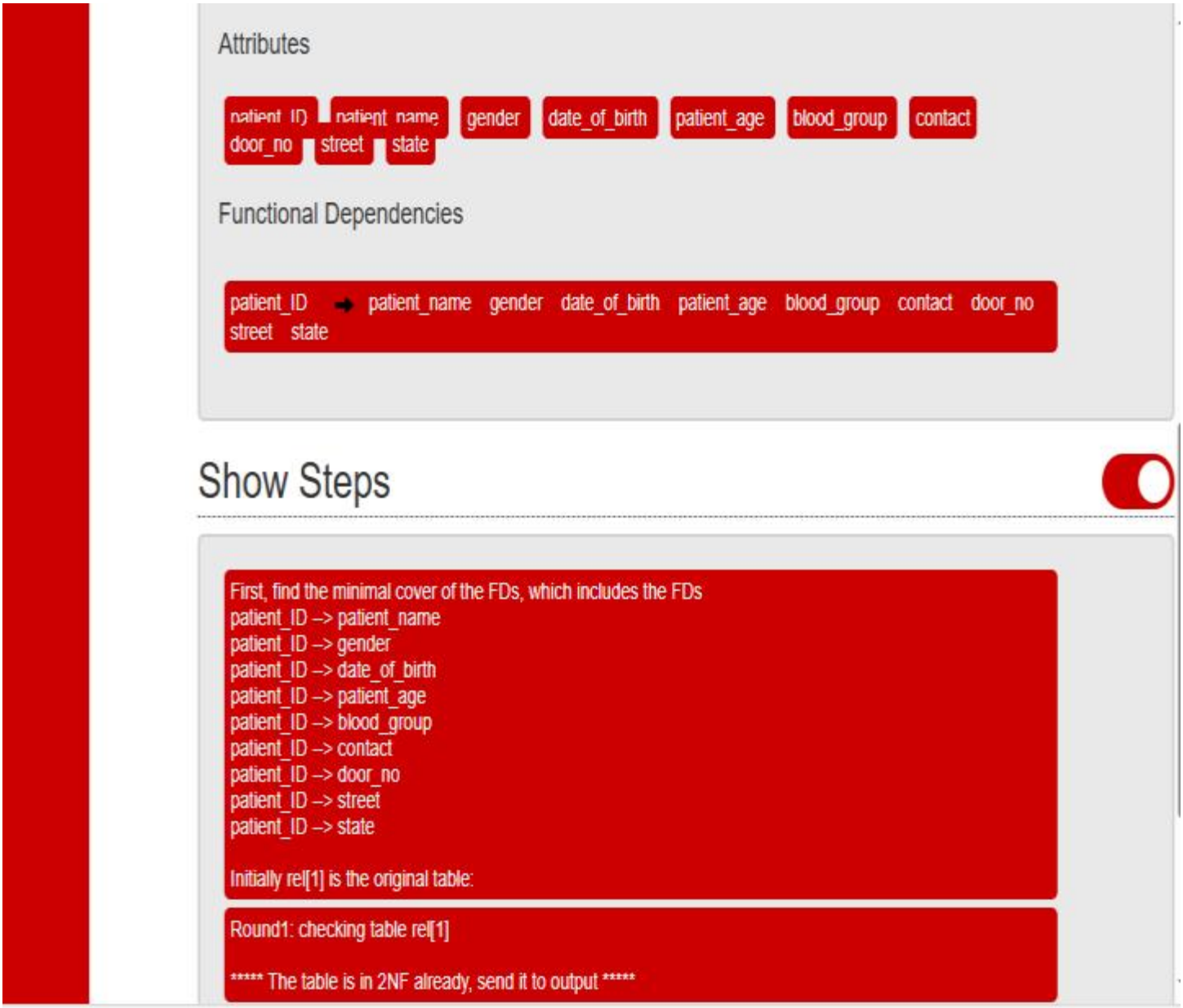
3NF

find all cadnidate keys. The candiates keys are { patient_ID}, The set of key attributes are: { patient_ID }
for each FD, check whether the LHS is superkey or the RHS are all key attributes
checking functional dependency patient_ID ->
patient_name,gender,date_of_birth,patient_age,blood_group,contact,door_no,street,state

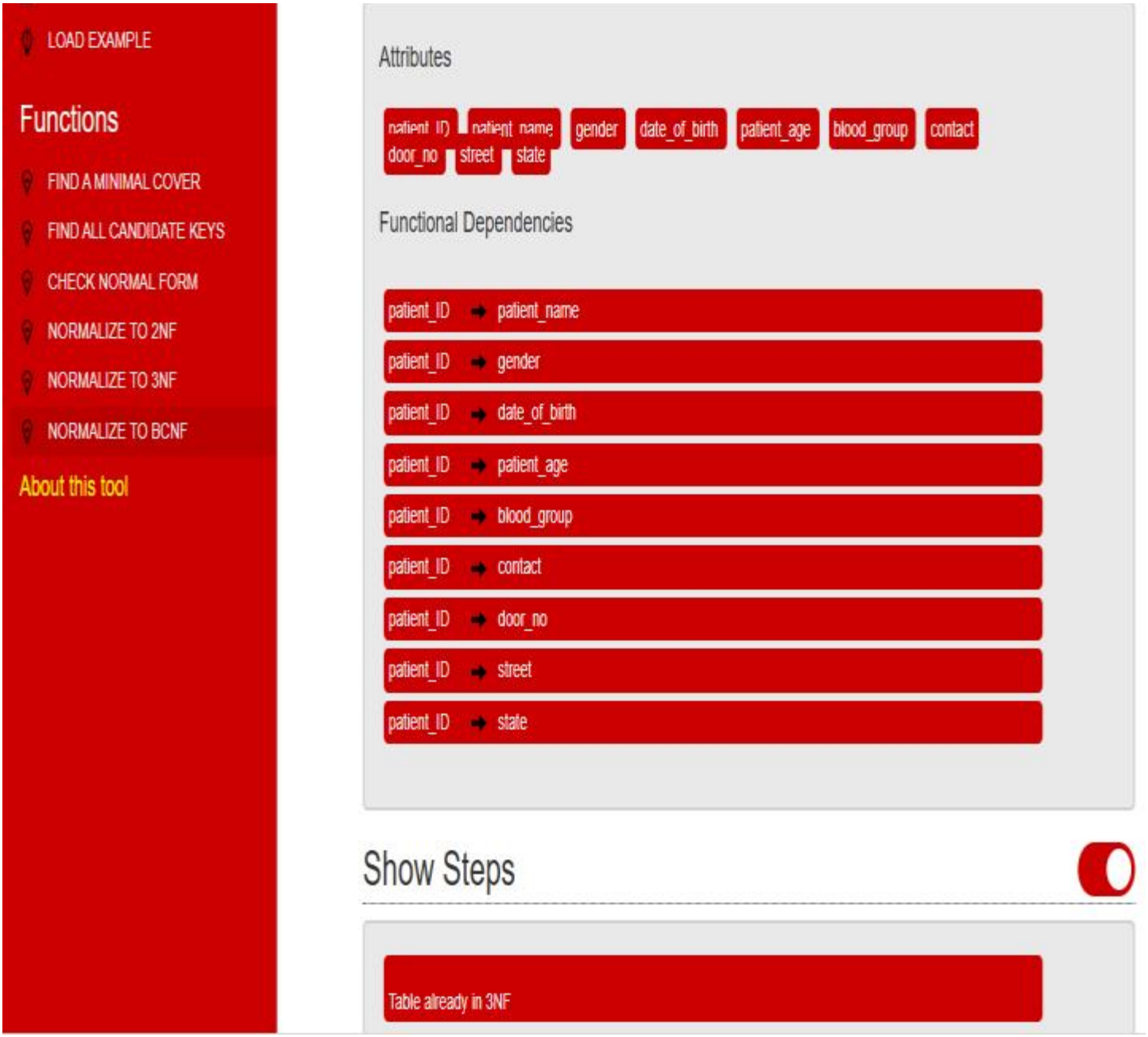
BCNF

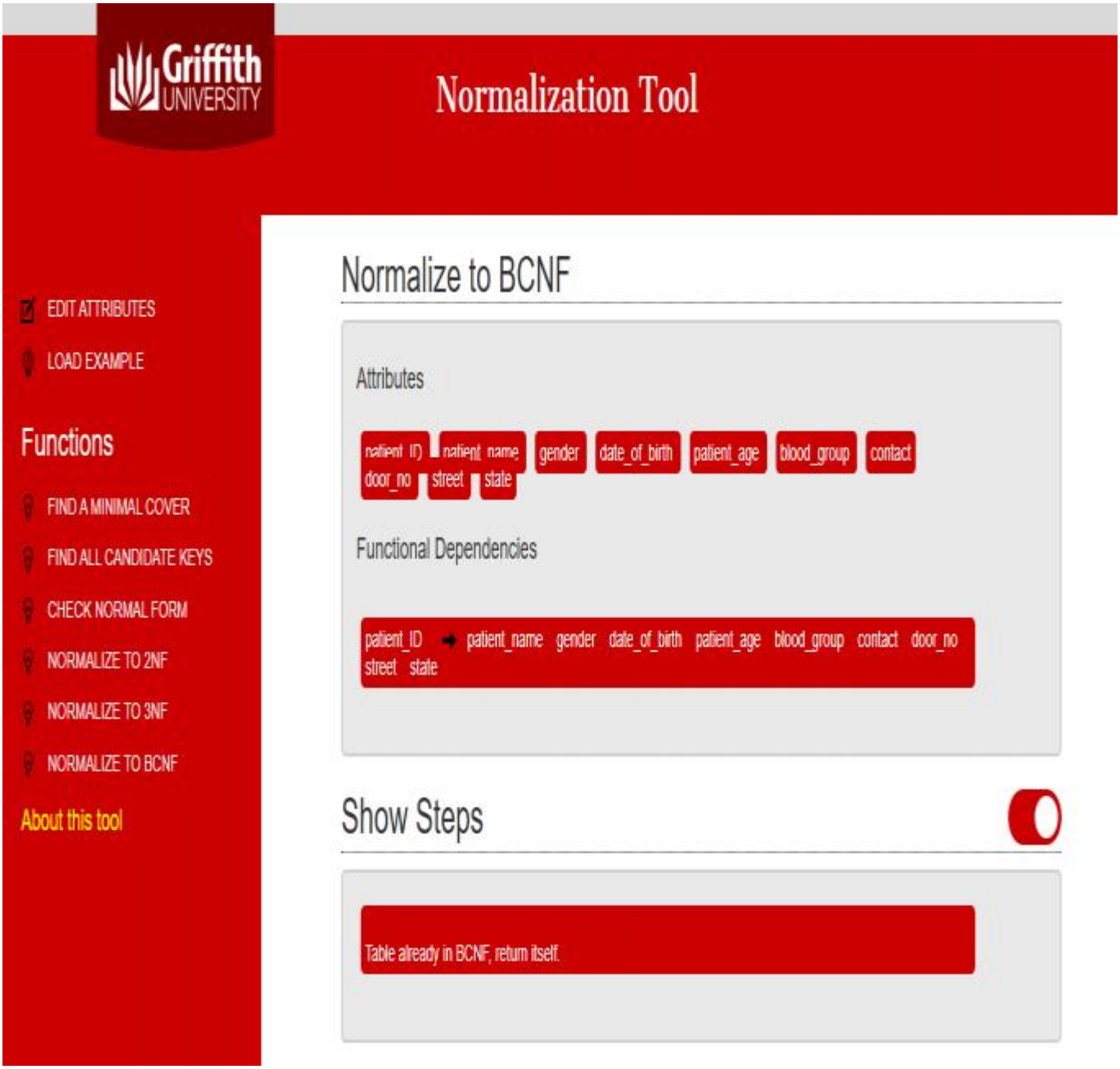
A table is in BCNF if and only if for every non-trivial FD, the LHS is a superkey.

Normalize to 2NF



1NF to 3NF


LOAD EXAMPLE
Functions
FIND A MINIMAL COVER
FIND ALL CANDIDATE KEYS
CHECK NORMAL FORM
NORMALIZE TO 2NF
NORMALIZE TO 3NF
NORMALIZE TO BCNF
About this tool
Attributes
patient_ID
patient_name
gender
date_of_birth
patient_age
blood_group
contact
door_no
street
state
Functional Dependencies
patient_ID → patient_name
patient_ID → gender
patient_ID → date_of_birth
patient_ID → patient_age
patient_ID → blood_group
patient_ID → contact
patient_ID → door_no
patient_ID → street
patient_ID → state
Show Steps
Table already in 3NF



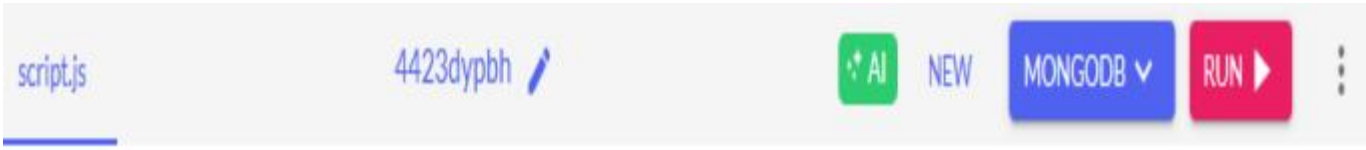
Result :Thus the Normalisation of 1NF,2NF,3NF,BCNF is successfully completed.

IMPLEMENTION OF THE DOCUMENT DATABASE FOR BLOOD BANK MANAGEMENT:

Aim:To implement the document database and graph dataset by using MongoDB

1. Donors Collection

2. Blood Inventory Collection



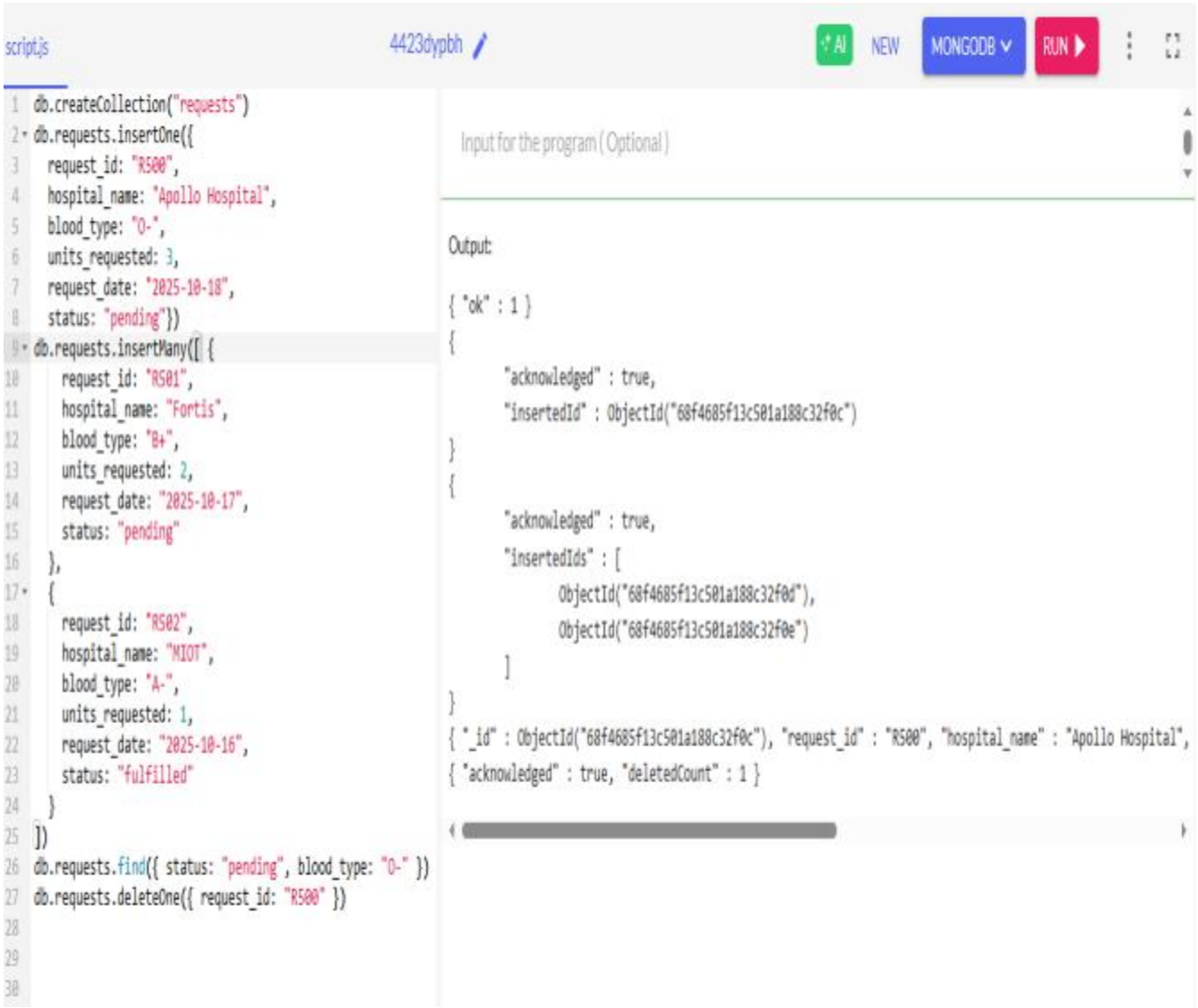
STDIN

Input for the program (Optional)

Output:

```
{ "ok" : 1 }
{
        "acknowledged" : true,
        "insertedId" : ObjectId("68f4671ebd96bfaa1e2f629b")
}
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("68f4671ebd96bfaa1e2f629c"),
                ObjectId("68f4671ebd96bfaa1e2f629d")
        ]
}
{ "_id" : ObjectId("68f4671ebd96bfaa1e2f629b"), "unit_id" : "U100", "blood_type" : "A+", "donation_date" : "2025-10-01", "
{ "acknowledged" : true, "deletedCount" : 1 }
```

3.Requests Collection



count Available Units by Blood Type

script.js 4423dypbh AI NEW

```
db.createCollection("blood_units")
db.blood_units.insertMany([
  {
    unit_id: "U101",
    blood_type: "B-",
    donation_date: "2025-10-05",
    expiry_date: "2025-11-05",
    status: "available",
    storage_location: "Fridge-2"
  },
  {
    unit_id: "U102",
    blood_type: "AB+",
    donation_date: "2025-10-07",
    expiry_date: "2025-11-07",
    status: "available",
    storage_location: "Fridge-3"
  }
])
db.blood_units.aggregate([
  { $match: { status: "available" } },
  { $group: { _id: "$blood_type", count: { $sum: 1 } } }
])
```

Input for the program (Optional)

Output:

```
{ "ok" : 1 }
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("68f469d2cd6b03a84a4984c7"),
                ObjectId("68f469d2cd6b03a84a4984c8")
        ]
}
{ "_id" : "AB+", "count" : 1 }
{ "_id" : "B-", "count" : 1 }
```

List of Unique Storage Locations

script.js 4423dypbh AI NEW

```
db.createCollection("blood_units")
db.blood_units.insertMany([
  {
    unit_id: "U101",
    blood_type: "B-",
    donation_date: "2025-10-05",
    expiry_date: "2025-11-05",
    status: "available",
    storage_location: "Fridge-2"
  },
  {
    unit_id: "U102",
    blood_type: "AB+",
    donation_date: "2025-10-07",
    expiry_date: "2025-11-07",
    status: "available",
    storage_location: "Fridge-3"
  }
])
db.blood_units.aggregate([
  { $match: { status: "available" } },
  { $group: { _id: "$blood_type", count: { $sum: 1 } } }
])
db.blood_units.aggregate([
  { $group: { _id: "$storage_location" } }
])
```

Input for the program (Optional)

Output:

```
{ "ok" : 1 }
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("68f46aca540b017f5afa2a07"),
                ObjectId("68f46aca540b017f5afa2a08")
        ]
}
{ "_id" : "B-", "count" : 1 }
{ "_id" : "AB+", "count" : 1 }
{ "_id" : "Fridge-3" }
{ "_id" : "Fridge-2" }
```

RESULT: Hence the implementation of Document Database is successfully complemed for Blood Bank Management System.