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.

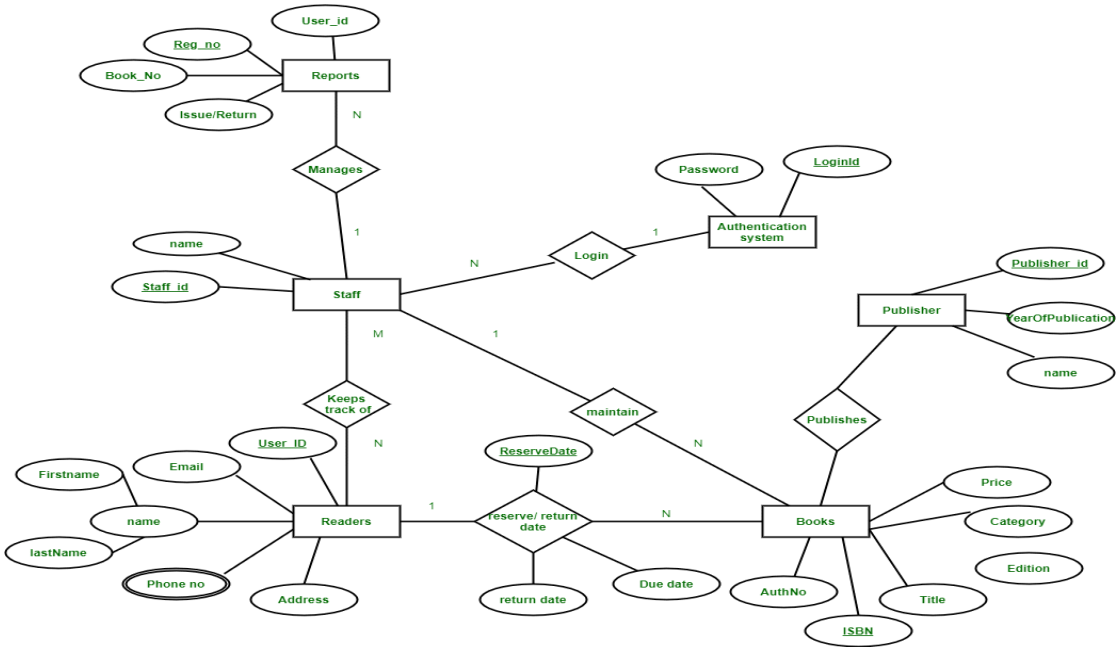
Sample Output

- 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.

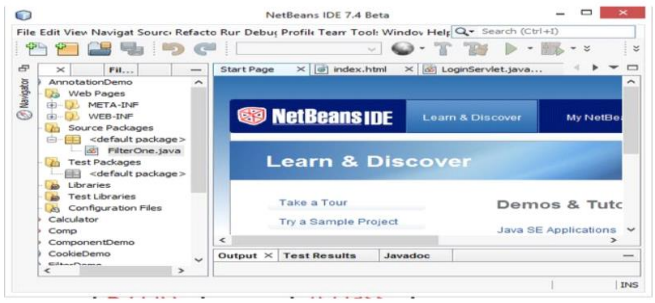
Sample input:

Make a report of students with their details

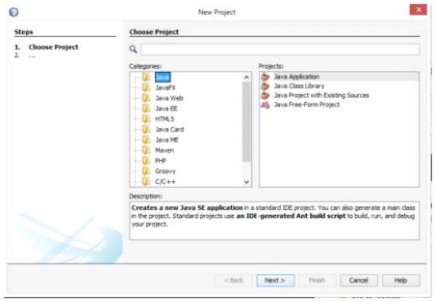
Sample output:

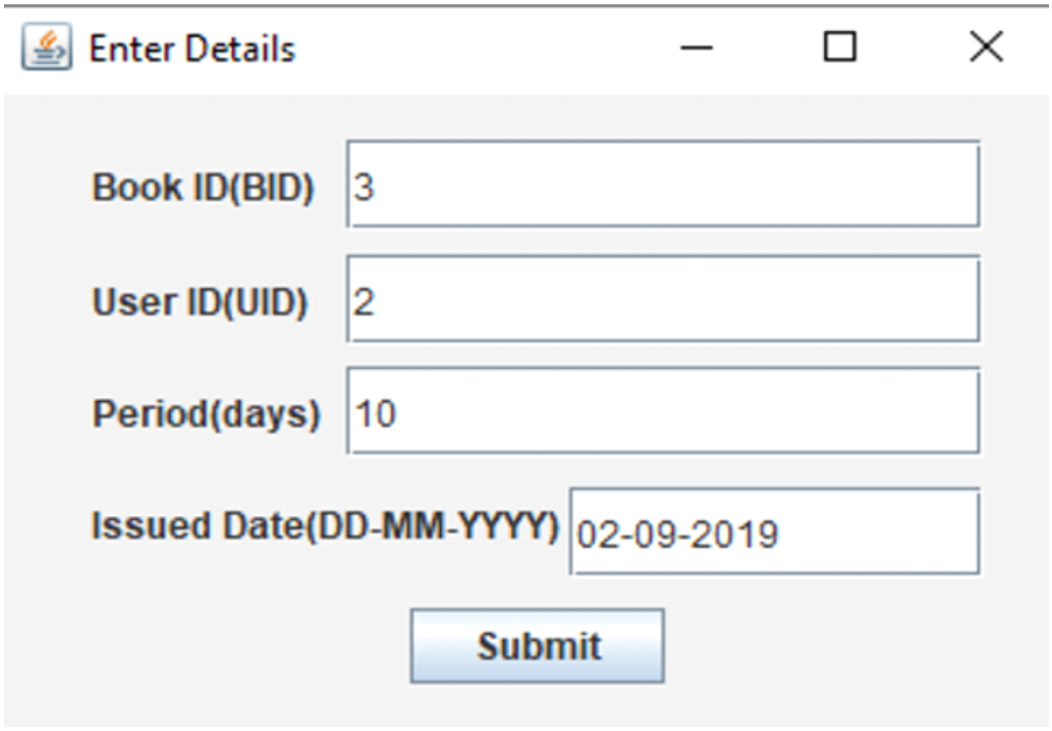
s_id	s_name	year	dept_id	dob	gender	place	phone_no	email_id
1234	TEJA	3	301	17/08/2001	M	Chennai	9876543210	xyz@gmail.com

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.

Part-II

**In perusal to the above task, Every student shall select and incorporate for one of the following use cases for Task-12.

Use case-1: User Module.

This module is further divided into various sub-modules describing the user in a better way:

- **New user register:**
- To sign up a new user to this system.
- **Student Login:**
- So as to confirm that only an authenticated user is using the project.
- **Search book:**
- The user can search book based on book id, book name, or by author name.
- **Issue Book:**
- To help the user get the required books issued.
- **Return Book:**
- To return the book before the last date without fine, or after the specified time duration with a late fine.

Use case-2: Admin Module

It is to be operated by the admin with a unique id and password. The admin is the person who decides authentication and authorization for all the different users of the application.

It further can be subdivided as:

- Register user.
- Issue Book.
- Maintain books in a stack, which means record the availability at a regular time interval.

Use case-3: Librarian Module

Includes all the library staff who are required to enter the records in the system and keep an eye on the various activities like the issue of the book, the return of the book, non-availability of books, etc. through the developed system.

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.

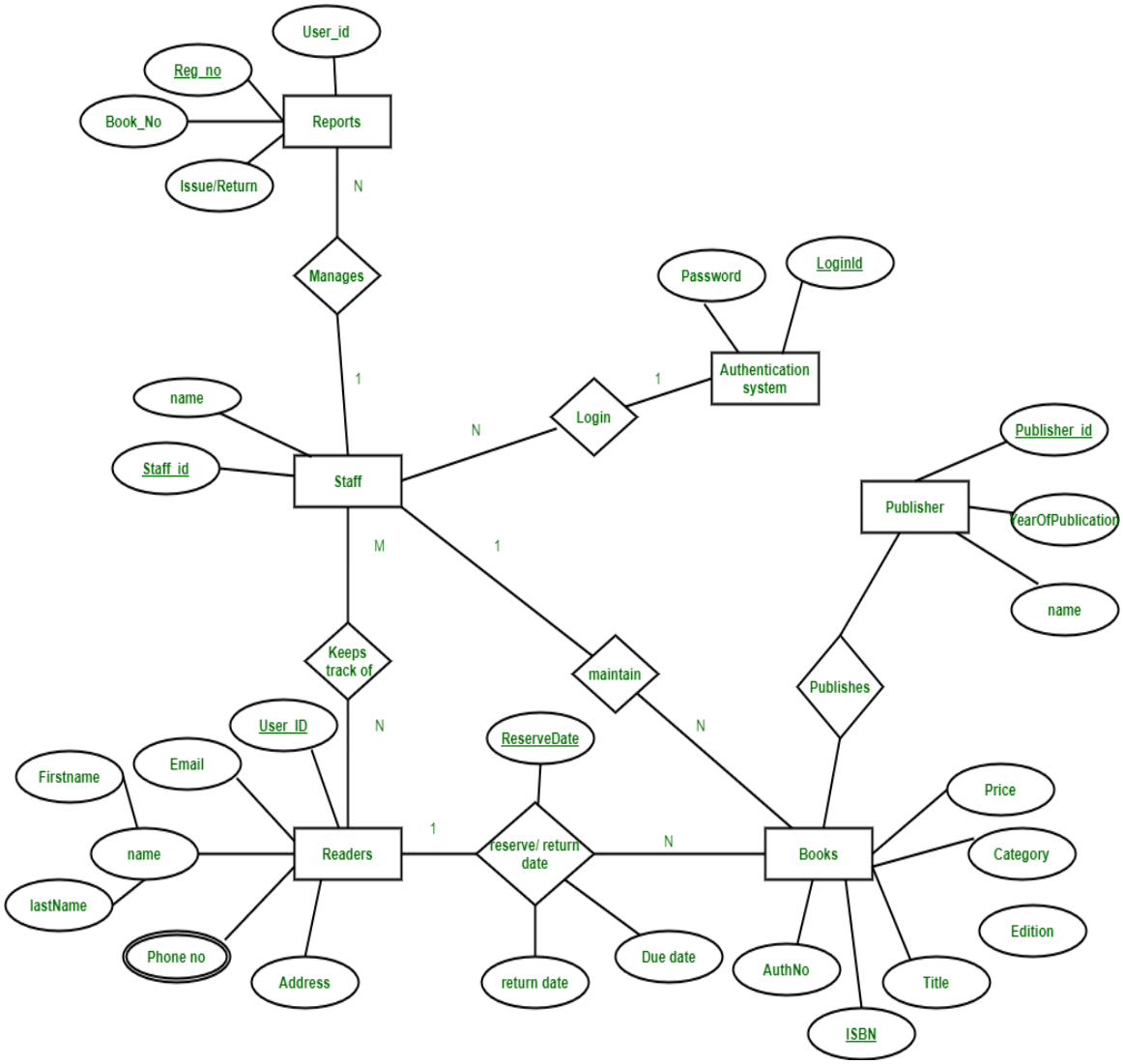
- Admin provides login to multiple staffs. The relation is 1:N.

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 includesdata 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 brackets determines 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 datein the 24-Hours format. By default the time in a date field is 12:00:00 am, if no time portion is specified. Thedefault 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 tablesprovided 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 nameto 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 asthose 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;

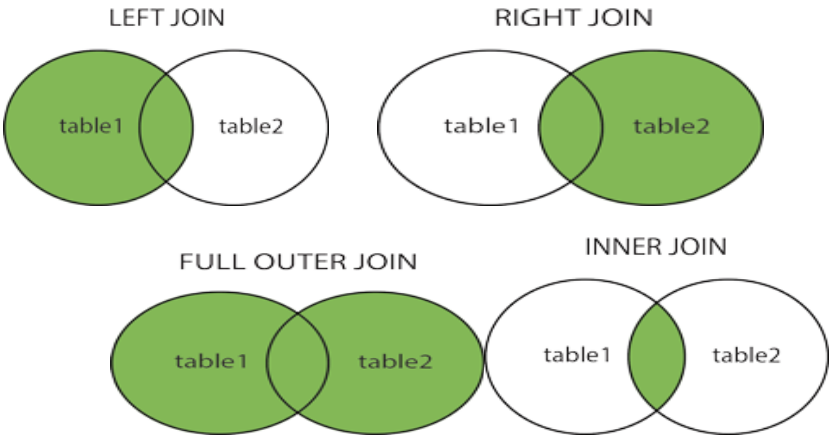
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from theleft table.

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.
```

TASK 7

Sample program for only Loops

- To print prime number customer id using loops

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

PL/SQL procedure successfully completed.
```

- To 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.
```

TASK 8: Normalizing databases using functional dependencies upto BCNF
(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).

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:

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,uom:"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: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"}}
])
{
"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"})
```

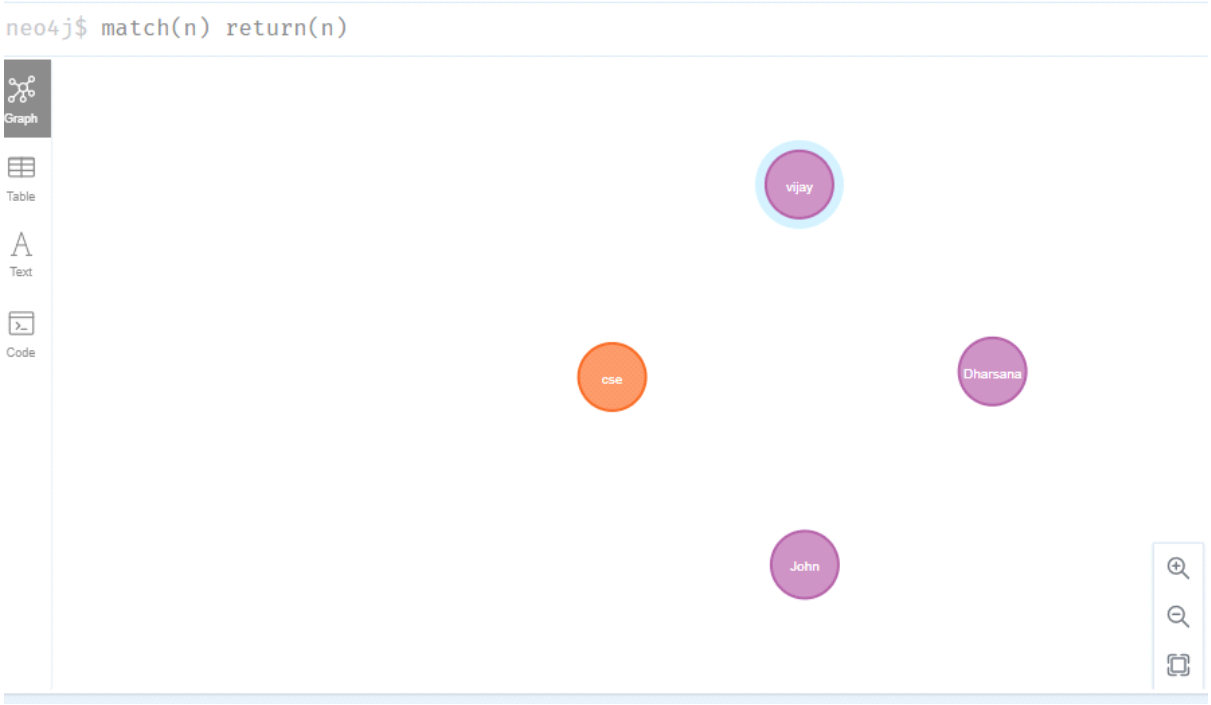
OUTPUT:

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

Select all the nodes in your database using match command

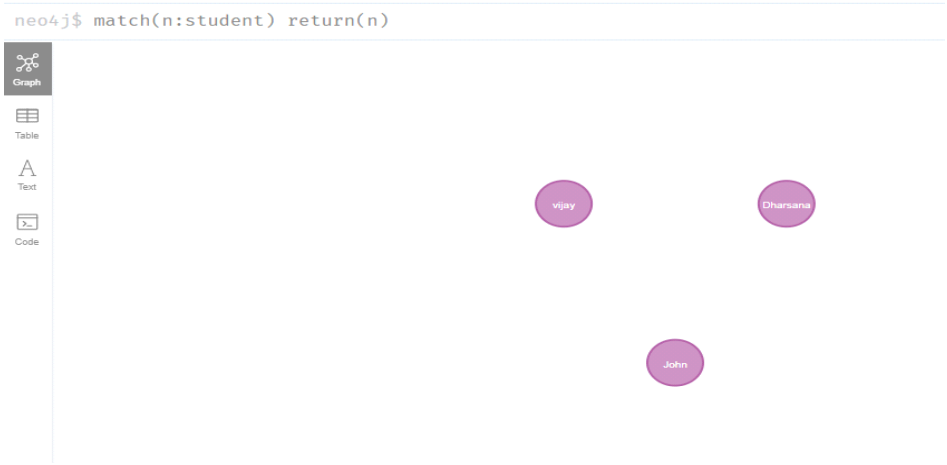
- **match(n) return(n)**

OUTPUT



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

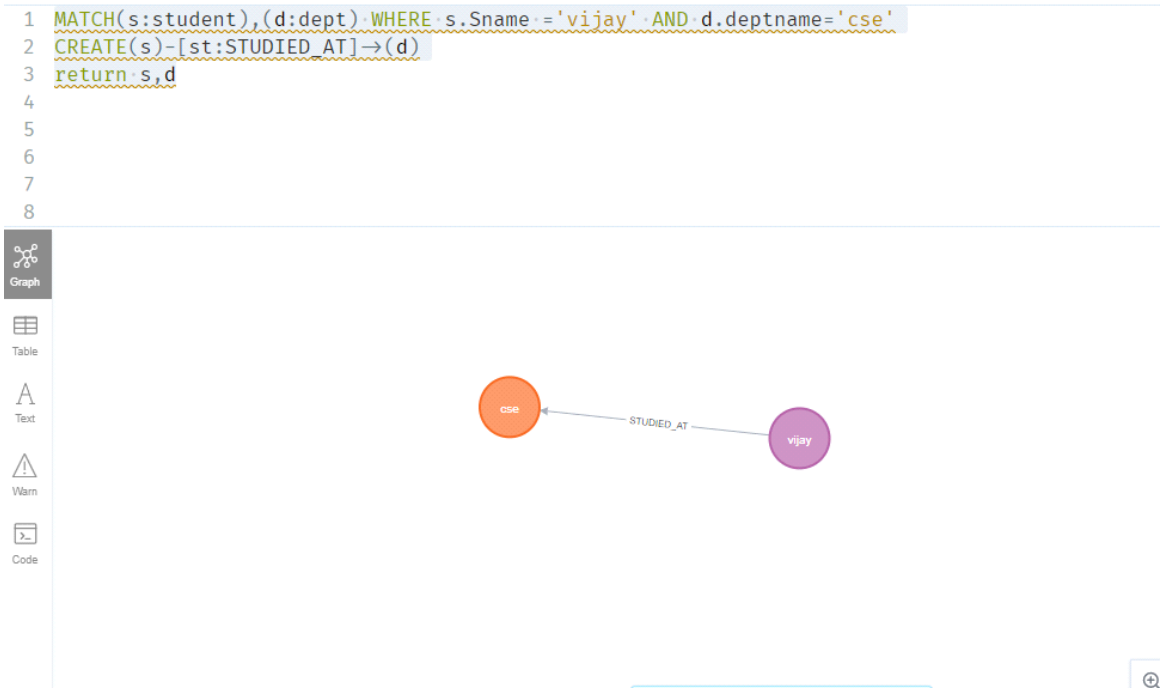
OUTPUT:



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

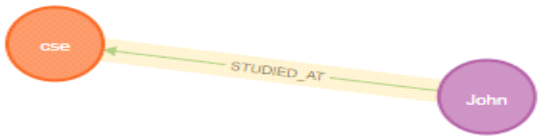
MATCH(s:student),(d:dept) WHERE s.Sname ='vijay' AND d.deptname='cse'
CREATE(s)-[st:STUDIED_AT]->(d)
return s,d

OUTPUT:

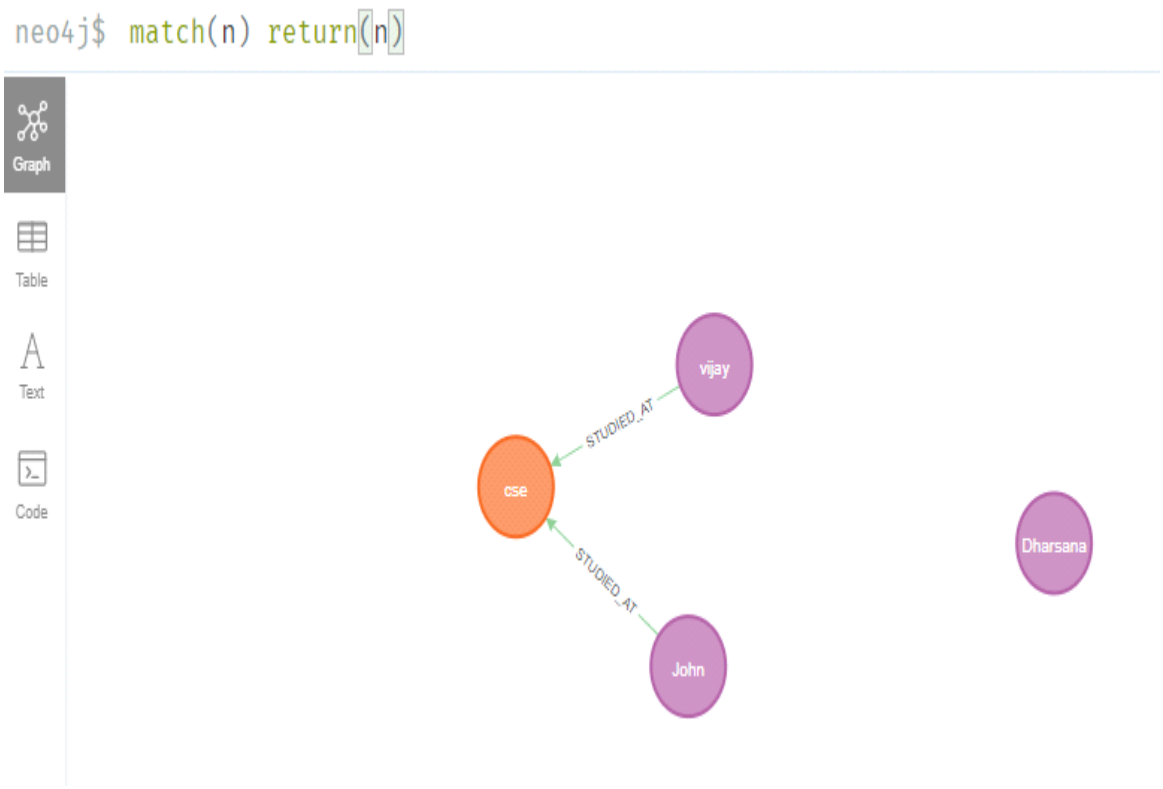


MATCH(s:student),(d:dept) WHERE s.Sname ='John' AND d.deptname='cse'
CREATE(s)-[st:STUDIED_AT]->(d)
return s,d

OUTPUT:



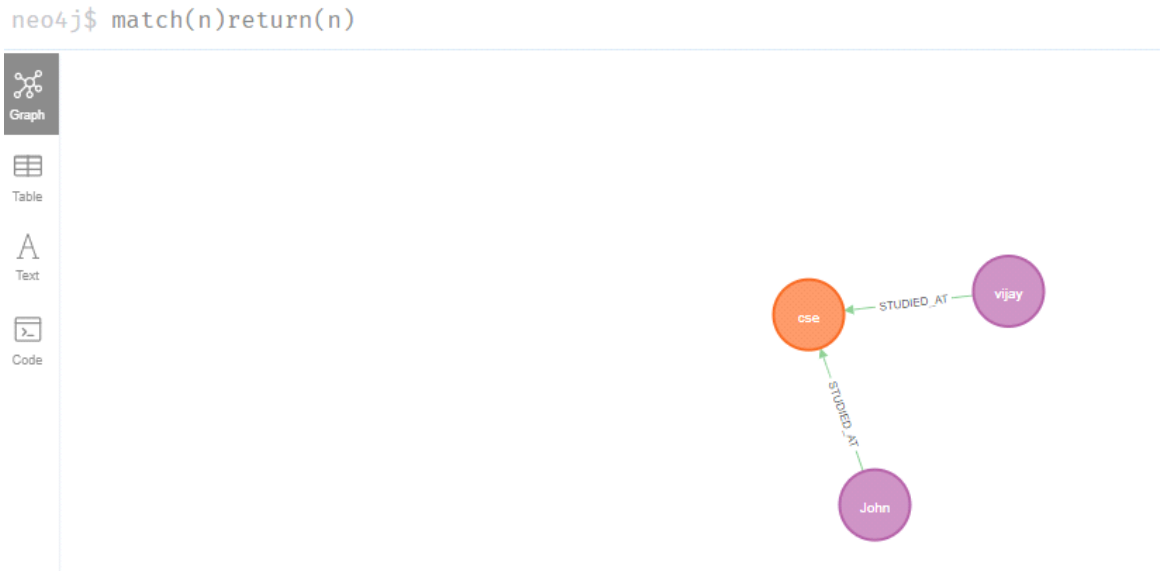
match(n) return(n)



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 executed.

TASK 12: MINI PROJECT

Viva Questions

- What is DBMS?

DBMS is a collection of programs that facilitates users to create and maintain a database.

- What is a database?

Database is a logical, consistent and organized collection of data that it can easily beaccessed, managed and updated.

- What is a database system?

The collection of database and DBMS software together is known as database system.

- What are the advantages of DBMS?

- Redundancy control
- Restriction for unauthorized access
- Provides multiple user interfaces
- Provides backup and recovery
- Enforces integrity constraints

- What is checkpoint in DBMS?

A Checkpoint is like a snapshot of the DBMS state.

- When does checkpoint occur in DBMS?

By taking checkpoints, the DBMS can reduce the amount of work to be done during restart inthe event of subsequent crashes.

- What do you mean by transparent DBMS?

The transparent DBMS is a type of DBMS which keeps its physical structure hidden fromusers.

- What are the unary operations in Relational Algebra?

PROJECTION and SELECTION are the unary operations in relational algebra.

- How do you communicate with an RDBMS?

Using Structured Query Language (SQL) RDBMS can be communicated.

- How many types of

database languages are?
There are four types of
database languages:
1.Data Definition Language (DDL) e.g. CREATE,
ALTER, DROP etc. 2.Data Manipulation Language (DML)
e.g. SELECT, UPDATE, INSERT etc.3.DATA Control
Language (DCL) e.g. GRANT and REVOKE.
4.Transaction Control Language (TCL) e.g. COMMIT and ROLLBACK.

- What do you understand by Data Model?

Data model is specified as a collection of conceptual tools for describing data, datarelationships data semantics and constraints.

- Define a Relation Schema and a Relation.

A Relation Schema is denoted by R(A1, A2, ..., An) is made up of the relation name R andthe list of attributes Ai that it contains.

A relation is specified as a set of tuples. See this example:
Let r be the relation which contains set tuples (t1, t2, t3, ...,tn). Each tuple is an ordered list ofn- values t=(v1,v2, ..., vn).

- What is degree of a Relation?

The degree of relation is a number of attribute of its relation schema.

- What is Relationship?

Relationship is defined as an association among two or more entities.

- What are the disadvantages of file processing systems?

- Inconsistent
- Not secure
- Data redundancy
- Difficult in accessing data
- Data isolation
- Data integrity
- Concurrent access is not possible

- What is data abstraction in DBMS?

Data abstraction in DBMS is a process of hiding irrelevant details from users. Because Database systems are made of complex data structures so, it makes easy the user interaction with database.

- What are the three levels of data abstraction?Following are three levels of data abstraction:

- Physical level: It is the lowest level of abstraction. It describes how data are stored.
- Logical level: It is the next higher level of abstraction. It describes what data are stored in database and what relationship among those data.
- View level: It is the highest level of data abstraction. It describes only part of entire database.

- What is DDL (Data Definition Language)?

Data Definition Language (DDL) is a standard for commands which define the different structures in a database. Most commonly DDL statements are CREATE, ALTER, and DROP.

- What is DML (Data Manipulation Language)?

Data Manipulation Language (DML) is a language that enable user to access or manipulate data as organised by appropriate data model.

There are two type of DML:

- Procedural DML or Low level DML: It requires a user to specify what data are needed andhow to get those data.
- Non-Procedural DML or High level DML: It requires a user to specify what data are neededwithout specifying how to get those data.

- Explain the functionality of DML Compiler.

The DML Compiler translates DML statements in a query language that the query evaluation engine can understand.

- What is Relational Algebra?

Relational Algebra is a Procedural Query Language which contains a set of operations that take one or two relations as input and produce a new relation.

- What is Relational Calculus?

Relational Calculus is a Non-procedural Query Language which uses mathematical predicate calculus instead of algebra.

- What do you understand by query optimization?

The term query optimization specifies an efficient execution plan for evaluating a query that has the least estimated cost.

- What do you mean by durability in DBMS?

Once the DBMS informs the user that a transaction has successfully completed, its effects should persist even if the system crashes before all its changes are reflected on disk. This property is called durability.

- What is normalization?

Normalization is a process of analyzing the given relation schemas according to their functional dependencies. It is used to minimize redundancy and also minimize insertion,

deletion and update distractions.

- What is Denormalization?

Denormalization is the process of boosting up database performance and adding of redundantdata which helps to get rid of complex data.

- What is functional Dependency?

Functional Dependency is the starting point of normalization. It exists when a relationbetween two attributes allows you to uniquely determine the corresponding attribute's value.

- What is E-R model?

E-R model is a short name for Entity Relationship model. This model is based on real world. It contains basic objects (known as entities) and relationship among these objects.

- What is entity?

Entity is a set of attributes in a database.

- What is an Entity type?

An entity type is specified as a collection of entities, having same attributes.

- What is an Entity set?

The entity set specifies the collection of all entities of particular entity type in the database.

- What is an Extension of entity type?

An extension of entity type is specified as a collection of entities of a particular entity typeare grouped together into an entity set.

- What is Weak Entity set?

When an entity set doesn't have sufficient attributes to form a primary key, and its primary keycompromises of its partial key and primary key of its parent entity, then it is called WeakEntityset.

- What is an attribute?

An attribute is a particular property, which describes the entity.

- What are the

integrity rules in
DBMS?There are two
integrity rules in
DBMS:

- Entity Integrity: It specifies that "Primary key cannot have NULL value."
- Referential Integrity: It specifies that "Foreign Key can be either a NULL value or should bePrimary Key value of other relation.

- What is Data Independence?

Data independence specifies that "the application is independent of the storage structure andaccess strategy of data".
It makes you able to modify the schema definition in one level should not affect the schemadefinition in the next higher level.
There are two types of Data Independence:

- Physical Data Independence: Modification in physical level should not affect the logicallevel.
- Logical Data Independence: Modification in logical level should affect the view level.NOTE: Logical Data Independence is more difficult to achieve.

- What are the three levels of data abstraction?

- Physical level: It is the lowest level of abstraction. It describes how data are stored. Logical level: It is the next higher level of abstraction. It describes what data are stored in database andwhat relationship among those data.

- View level: It is the highest level of abstraction. It describes only part of entire database.

- What is stored procedure?

A stored procedure is a named group of SQL statements that have been previously createdand stored in the server database.

- What is 1NF?

1NF is the First Normal Form. It is the simplest type of normalization that you canimplement in a database. The main objectives of 1NF are to:

- Remove duplicate columns from the same table
- Create separate tables for each group of related data and identify each row with a uniquecolumn

- What is 2NF?

2NF is the Second Normal Form. A table is said to be 2NF, if it follows the followingconditions:
The table is in 1NF.

Every non-prime attribute is fully functionally dependent on primary key.

- What is 3NF?

3NF stands for Third Normal Form. A database is called in 3NF if satisfies the followingconditions:
It is in second normal form.

There is no transitive functional dependency.

- What is BCNF?

BCMF stands for Boyce-Codd Normal Form. It is an advance version of 3NF so it is also

referred as 3.5NF. BCNF is stricter than 3NF.
A table complies with BCNF if it satisfies the following
conditions: It is in 3NF.For every functional dependency X->Y,
X should be the super key of the table.

- What is PL SQL ?

PL SQL is a procedural language which has interactive SQL, as well as proceduralprogramming language constructs like conditional branching and iteration.

- List the uses of database trigger.

A PL/SQL program unit associated with a particular database table is called a databasetrigger. It is used for :

- Audit data modifications.
- Log events transparently.
- Enforce complex business rules.
- Maintain replica tables
- Derive column values
- Implement Complex security authorizations

- What are the two types of exceptions.

Error handling part of PL/SQL block is called Exception. They have two types : user_definedand predefined.

- What is functions and procedures in a PL SQL block?

Function is called as a part of an expression. Procedure is called as a statement in PL/SQL.

- Explain Commit,

Rollback and Savepoint. For
a COMMIT statement, the
following is true:

- Other users can see the data changes made by the transaction.
- The locks acquired by the transaction are released.
- The work done by the transaction becomes permanent.

A ROLLBACK statement gets issued when the transaction ends, and
the following is true.The work done in a transition is undone as if it was never issued.
All locks acquired by transaction are released.It undoes all the work done by the user in atransaction.
With SAVEPOINT, only part of transaction can be undone.

- How many triggers can be applied to a table?

A maximum of 12 triggers can be applied to one table.

- List 3 basic parts of a trigger.

•A triggering statement or event.

•A restriction

•An action

- Mention what PL/SQL

package consists of?A

PL/SQL package consists of

- •PL/SQL table and record TYPE statements
- •Procedures and Functions
- •Cursors
- •Variables (tables, scalars, records, etc.) and constants
- •Exception names and pragmas for relating an error number with an exception
- •Cursors

- Define Atomicity and Aggregation.

Atomicity: It's an all or none concept which enables the user to be assured of incomplete transactions to be taken care of. The actions involving incomplete transactions are left undonein DBMS.

Aggregation: The collected entities and their relationship are aggregated in this model. It ismainly used in expressing relationships within relationships.

- Enlist the various

transaction phases.

The various

transaction phases

are:

- Analysis Phase.
- Redo Phase
- Undo Phase

- What is Storage Manager?

It is a program module that provides the interface between the low-level data stored indatabase, application programs and queries submitted to the system.

- What is Buffer Manager?

It is a program module, which is responsible for fetching data from disk storage into mainmemory and deciding what data to be cache in memory.

- What is Transaction Manager?

It is a program module, which ensures that database, remains in a consistent state despitesystem failures and concurrent transaction execution proceeds without conflicting.

- What is File Manager?

It is a program module, which manages the allocation of space on disk storage and datastructure used to represent information stored on a disk.

- What is Authorization and Integrity manager?

It is the program module, which tests for the satisfaction of integrity constraint and checksthe authority of user to access data.

- Describe ODBC

ODBC is a standard that contains an interface that provides a common language for applicationprograms to access and process SQL databases. In order to use ODBC, a driver, server name, database name, user id, and password are required. ODBC is important for Internet applicationsand has gained wide acceptance.

59).Provide an overview of XML.

XML is used to structure and manipulate data involved with a browser and is becoming the standard for e-commerce. Xl\IL uses tags that are similar to HTML in that they use the angle brackets, but XML describes the content whereas HTML describes the appearance. The MIL schema standard was published in May 2001.

60).Describe Website security issues.

Website security issues include unauthorized access to the several aspects of one's Website.. Security measures should include all aspects of the system such as the network, operating level, database, and Web server. Regular monitoring and security testing by a company should help to avoid intrusion into one's system.

- Explain the role of metadata for the three-layer architecture.

Each of the three layers has a metadata laver linked with it. The metadata layer describes the properties or characteristics of the data. The operational

metadata describe the data used in the various operational and external systems. The enterprise data warehouse metadata describe thereconciled data layer. The data mart metadata describes the derived data laver.

- Explain concurrency transparency.

Concurrency transparency is where each transaction in a distributed database is treated as if itis the only one in the system. Therefore if several transactions are running at one time. the results will be the same as if each transaction was run in serial order. The transaction managerhelps to provide concurrency control. The three methods that may be used are locking, versioning, and time stamping .

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

VTR UGE2021- (CBCS)

Academic Year: 2025–2026

SDG 4: Quality Education
Course Code : 10211CS207

Course Name : Database Management Systems

Slot No : S4 L5

DBMS PROJECT REPORT

Title: Banking Management System (BMS)
Submitted by:

VTUNO	REGISTER NUMBER	STUDENT NAME
VTU29497	24UECS0374	T.NAGA MANIKANTA
VTU27989	24UECS0926	S.ABHIRAM
VTU27667	24UECS1352	R.ANIRUTH
VTU27820	24UECS1192	E.B.V.KRISHNA TEJA
VTU28457	24UECS0728	M.VISHNU VARDHAN SAI

Under the guidance of:
Dr. Uma Nandhini.D
Professor

INDEX **PAGE**

Introduction

A Banking Management System (BMS) is an integrated software solution designed to automate and streamline the various operations performed in a banking environment. It plays a vital role in managing essential banking functions such as customer account creation, balance inquiries, deposits, withdrawals, fund transfers, loan processing, and report generation. The primary objective of this system is to eliminate the inefficiencies associated with manual banking processes by providing a reliable, accurate, and secure digital platform that enhances overall productivity and customer satisfaction.

In the past, most banking processes were executed manually using paper-based systems or outdated software tools. This approach made everyday operations slow, labor-intensive, and highly susceptible to human error. Activities such as updating customer records, calculating interest, or maintaining ledgers required significant time and attention. As the number of customers and transactions increased, these manual procedures became even more cumbersome, often leading to data inconsistency and service delays. To address these challenges, the introduction of a computerized banking management system has become essential in modern banking institutions.

A well-designed Banking Management System allows banks to manage all their functions efficiently under a centralized digital platform. It enables real-time transaction processing, ensuring that customers receive instant updates regarding their account balances and financial activities. The system's ability to securely store and retrieve data eliminates the need for redundant paperwork, thereby minimizing the risk of errors and loss of information. Furthermore, automation enhances accuracy in financial calculations, improves the reliability of reports, and ensures that all records remain consistent across different departments.

The BMS also strengthens data security and access control, which are critical in financial institutions. Sensitive customer information and transaction details are protected through encryption, authentication, and authorization mechanisms. This ensures that only authorized users can access or modify data, significantly reducing the risk of fraud and unauthorized access. Additionally, digital record-keeping supports better data backup and recovery options, safeguarding important financial information from accidental loss or physical damage.

From a customer perspective, the Banking Management System enhances convenience and accessibility. Customers can easily perform banking activities such as checking account balances, transferring funds, or applying for loans without visiting the bank in person.

Problem Statement

In traditional banking systems, most core operations such as account creation, deposits, withdrawals, fund transfers, and record management are performed manually. This manual approach often involves paperwork, ledgers, and physical files, which makes the overall process slow, inefficient, and prone to human errors. Such dependency on manual procedures not only increases the workload of bank employees but also leads to delays in providing services to customers. As a result, even basic operations like updating an account balance or generating a transaction report can consume significant time and effort.

Manual systems make it extremely difficult to maintain accurate, up-to-date records of customer activities. Small mistakes during data entry or record updates can result in discrepancies that may go unnoticed for long periods, leading to financial miscalculations or customer dissatisfaction. Moreover, retrieving customer details, verifying account histories, and preparing financial statements are all time-intensive processes that lack transparency and reliability. This can reduce the trust customers place in the banking institution and ultimately affect its reputation.

As the number of customers continues to grow, managing large volumes of data manually becomes increasingly complex. Outdated systems or paper-based storage methods are not scalable and cannot efficiently handle modern banking demands. Issues such as data redundancy, inconsistency, and difficulty in record retrieval become common. In addition, manual systems offer very limited data security, making them vulnerable to unauthorized access, data manipulation, or even loss due to physical damage to files.

Another significant limitation of traditional banking systems is the lack of automation and integration between departments. For example, when a customer requests a loan or a balance statement, different sections of the bank may need to process the request separately, resulting in unnecessary delays. This not only reduces operational efficiency but also leads to poor customer experience. In an era where customers expect fast, secure, and digital banking services, such inefficiencies can drive them toward more technologically advanced competitors.

Furthermore, report generation, auditing, and compliance tracking become tedious and error-prone in a manual setup. Generating monthly or yearly summaries requires going through numerous paper records, which consumes valuable employee time and increases the chances of misreporting. The lack of centralized data storage also hinders real-time monitoring and decision-making for manage

Objectives

The main objective of the **Banking Management System (BMS)** is to design and develop a computerized system that can efficiently manage all banking operations, improve service delivery, and enhance the overall functioning of the bank. The system aims to automate manual banking tasks, reduce human error, and ensure accuracy, security, and speed in daily operations.

Specific Objectives:

1. **To automate core banking operations:**
    a. Replace manual processes with an automated system that handles account creation, deposits, withdrawals, fund transfers, and balance inquiries.
    b. Ensure all transactions are processed quickly, accurately, and consistently without manual intervention.
2. **To maintain accurate and secure customer records:**
    a. Store all customer data, account details, and transaction histories in a centralized digital database.
    b. Prevent data duplication, inconsistency, and loss through reliable storage and backup mechanisms.
3. **To minimize human errors:**
    a. Eliminate calculation mistakes, entry errors, and duplication that commonly occur in manual record-keeping.
    b. Incorporate validation checks and automated verification to ensure data integrity.
4. **To enable quick and efficient data retrieval:**
    a. Provide instant access to customer details, transaction history, and account status through a centralized database.
    b. Reduce the time required to search, verify, and update records.
5. **To improve customer service quality:**
    a. Offer faster transaction processing and immediate service response.
    b. Allow customers to access their accounts easily and perform operations like checking balances, making deposits, or applying for loans without delays.
6. **To simplify report generation and analysis:**
    a. Generate daily, monthly, or annual reports automatically for management and auditing purposes.
    b. Support data analysis for better financial planning and decision-making.
7. **To enhance data security and privacy:**
    a. Implement user authentication, access control, and encryption mechanisms to protect sensitive financial information.

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS

Component	Minimum Specification	Recommended Specification
Processor	Intel Core i3 or equivalent	Intel Core i5 or above
RAM	4 GB	8 GB or higher
Hard Disk	250 GB	500 GB or more
Display	1024 × 768 resolution	1366 × 768 or higher
Input Devices	Keyboard, Mouse	Keyboard, Mouse
Output Devices	Monitor, Printer	High-resolution Monitor, Printer (for reports)
Network	Basic LAN or Internet connection	High-speed Internet for online access

SOFTWARE REQUIREMENTS

Category	Specification
Operating System	Windows 10/11, Linux (Ubuntu), or maCOS
Database Software	MySQL / Oracle / PostgreSQL / MS Access
Front-End (for UI, optional)	HTML, CSS, Java, Python, PHP, or any front-end tool
Backend / Server-side Language	Java, Python (Flask/Django), PHP, or .NET (depending on project choice)
DBMS Tool	MySQL Workbench / Oracle SQL Developer /phpMyAdmin
Text Editor / IDE	Visual Studio Code, Eclipse, NetBeans, PyCharm, or any preferred IDE
Browser Generation (optional)	Google Chrome / Mozilla Firefox / Microsoft Edge MS Excel / PDF Report Tool / JasperReports

TITLE : BANKING MANAGEMENT SYSTEM(BMS)

1. **ER Diagram:**

Aim: to draw the conceptual design for banking System.

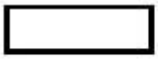
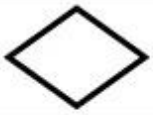
E-R Diagram :

Entity-Relationship model:

An E-R Diagram (Entity-Relationship Diagram) is a visual tool used to represent the structure of a database — showing entities (objects), their attributes (properties), and relationships.

 It helps design and organize a database clearly before implementation, ensuring proper data connections and reducing redundancy or errors.

 It develops a conceptual design for the database. It also develops a very simple and easy to design view of data

Symbol Name	Symbol	Represents
Rectangles		Represents Entity
Ellipses		Represents Attribute
Diamonds		Represents Relationship
Lines		Links Attribute(s) to entity set(s) or Entity set(s) to Relationship set(s)
Double Ellipses		Represents Multivalued Attributes
Primary key		Represents Key Attributes / Single Valued Attributes

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 BANKING MANAGEMENT SYSTEM

- **ENTITIES**: Bank, Branch, Customer, Account, Transaction, Loan, Employee

ATTRIBUTES

1. **Bank**
    a. Bank_ID (PK)
    b. Name
    c. Code
    d. Main_Office_Address
2. **Branch**
    a. Branch_ID (PK)
    b. Branch_Name
    c. Address
    d. Bank_ID (FK)
3. **Customer**
    a. Customer_ID (PK)
    b. Name
    c. Address
    d. Phone
    e. Email
4. **Account**
    a. Account_ID (PK)
    b. Account_Type

c. Balance
d. Customer_ID (FK)
e. Branch_ID (FK)

5. **Transaction**
    a. Transaction_ID (PK)
    b. Date
    c. Amount
    d. Type (Deposit/Withdrawal/Transfer)
    e. Account_ID (FK)
6. **Loan**
    a. Loan_ID (PK)
    b. Loan_Type
    c. Amount
    d. Interest_Rate
    e. Customer_ID (FK)
    f. Branch_ID (FK)
7. **Employee**

- Employee_ID (PK)
- Name
- Position
- Phone
- Branch_ID (FK)

RELATIONS

- **Bank – Branch** (1 : M)

One bank can have many branches.

- **Branch – Employee** (1 : M)

One branch can have many employees.

- **Branch – Account** (1 : M)

One branch can manage many accounts.

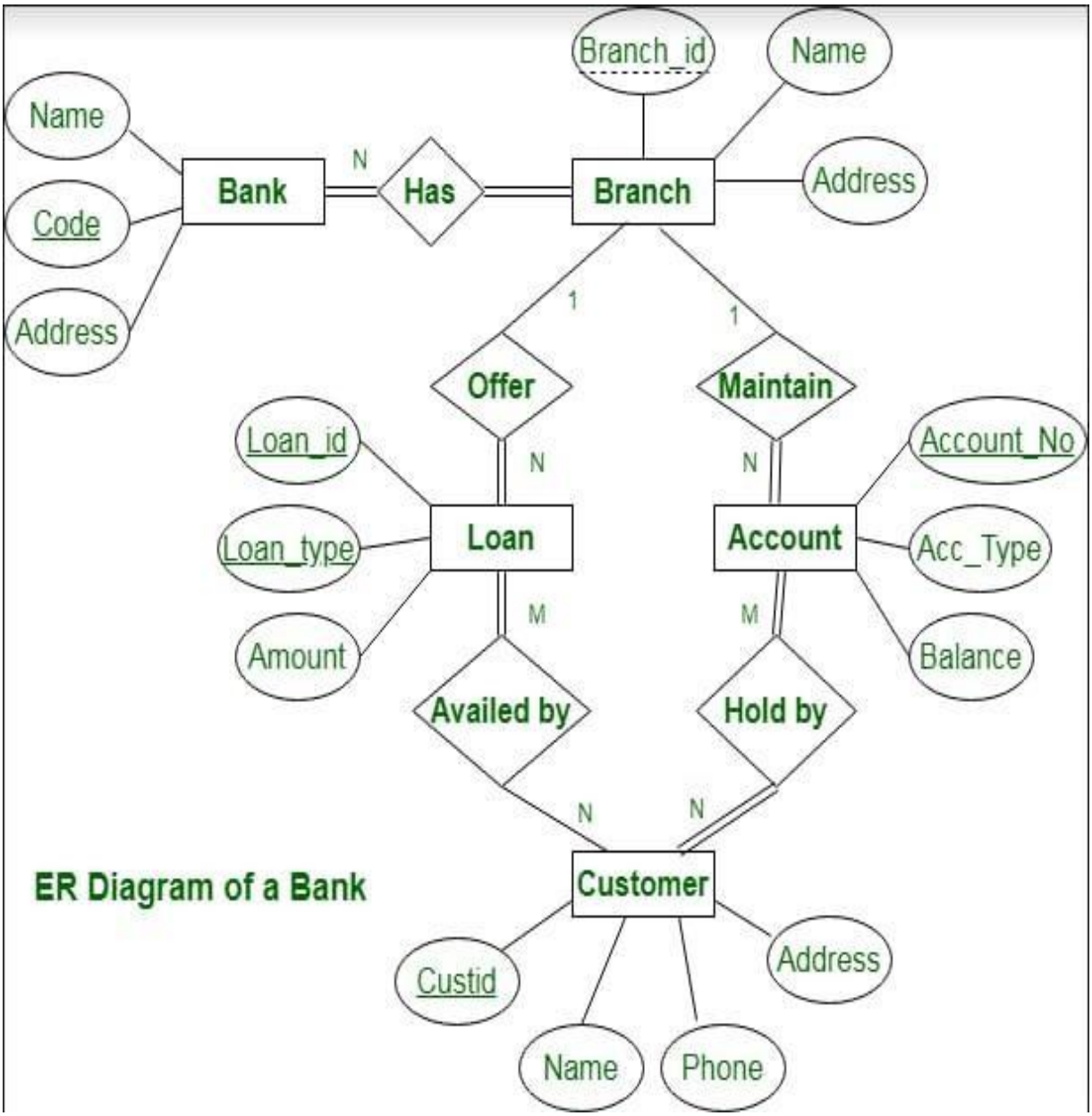
- **Branch – Loan** (1 : M)

One branch can issue many loans.

- **Customer – Account** (1 : M)

One customer can hold multiple accounts.

- **Customer – Loan** (1 : M)



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

SQL QUERIES FOR HOSPITAL MANAGEMENT SYSTEM :

Sql queries are :

```
CREATE TABLE Branch (
  Branch_ID INT PRIMARY KEY,
  Branch_Name VARCHAR(50),
  Location VARCHAR(50)
);

CREATE TABLE Employee (
  Employee_ID INT PRIMARY KEY,
  Name VARCHAR(50),
  Position VARCHAR(30),
  Branch_ID INT
);

CREATE TABLE Customer (
  Customer_ID INT PRIMARY KEY,
  Name VARCHAR(50),
  Age INT,
  Gender VARCHAR(10),
  Address VARCHAR(100),
  Phone VARCHAR(15)
);

CREATE TABLE Account (
  Account_ID INT PRIMARY KEY,
```

```
  Customer_ID INT,
  Branch_ID INT,
  Account_Type VARCHAR(20),
  Balance DECIMAL(10,2)
);

CREATE TABLE BankTransaction (
  Transaction_ID INT PRIMARY KEY,
  Account_ID INT,
  Transaction_Date DATE,
  Type VARCHAR(10),
  Amount DECIMAL(10,2)
);

CREATE TABLE Loan (
  Loan_ID INT PRIMARY KEY,
  Customer_ID INT,
  Branch_ID INT,
  Loan_Type VARCHAR(20),
  Amount DECIMAL(10,2),
  Interest_Rate  DECIMAL(5,2)
);
```

2. **Insert Sample Data**

```
INSERT INTO Branch VALUES
```

```
(1, 'Main Branch', 'Delhi'),
(2, 'West Branch', 'Mumbai'),
(3, 'East Branch', 'Pune');

INSERT INTO Employee VALUES
(101, 'Amit Sharma', 'Manager', 1),
(102, 'Neha Verma', 'Clerk', 2),
(103, 'Rajesh Kumar', 'Cashier', 3);

INSERT INTO Customer VALUES
(201, 'Karan Mehta', 35, 'Male', 'Delhi', '9876543210'),
(202, 'Pooja Singh', 29, 'Female', 'Mumbai', '9988776655'),
(203, 'Ravi Desai', 42, 'Male', 'Pune', '9123456780');

INSERT INTO Account VALUES
(301, 201, 1, 'Savings', 50000.00),
(302, 202, 2, 'Current', 75000.00),
(303, 203, 3, 'Savings', 62000.00);

INSERT INTO BankTransaction VALUES
(401, 301, '2025-10-01', 'Deposit', 10000.00),
(402, 302, '2025-10-02', 'Withdrawal', 5000.00),
(403, 303, '2025-10-03', 'Deposit', 8000.00);

INSERT INTO Loan VALUES
(501, 201, 1, 'Home Loan', 1000000.00, 7.5),
```

(502, 202, 2, 'Car Loan', 500000.00, 8.0),

(503, 203, 3, 'Personal Loan', 300000.00, 10.0);

Output: Banking Management System

-- Branch Table

Branch_ID	Branch_Name	Location
1	Main Branch	Delhi
2	West Branch	Mumbai
3	East Branch	Pune

- Employee Table

Employee_ID	Name	Position	Branch_ID
101	Amit Sharma	Manager	1
102	Neha Verma	Clerk	2
103	Rajesh Kumar	Cashier	3

-- Customer Table

Customer_ID	Name	Age	Gender	Address	Phone
201	Karan Mehta	35	Male	Delhi	9876543210
202	Pooja Singh	29	Female	Mumbai	9988776655
203	Ravi Desai	42	Male	Pune	9123456780

-- Account Table

Account_ID	Customer_ID	Branch_ID	Account_Type	Balance
301	201	1	Savings	50000.00
302	202	2	Current	75000.00
303	203	3	Savings	62000.00

-- Transaction Table

Transaction_ID	Account_ID	Transaction_Date	Type	Amount
401	301	2025-10-01	Deposit	10000.00
402	302	2025-10-02	Withdrawal	5000.00
403	303	2025-10-03	Deposit	8000.00

-- Loan Table

Loan_ID	Customer_ID	Branch_ID	Loan_Type	Amount	Interest_Rate
501	201	1	Home Loan	1000000.00	7.5
502	202	2	Car Loan	500000.00	8.0
503	203	3	Personal Loan	300000.00	10.0

Aggregate Functions

These queries summarize data across multiple rows:

1. Total Balance in Each Branch

Input

```
SELECT Branch_ID, SUM(Balance) AS Total_Balance
FROM Account
GROUP BY Branch_ID;
```

Output

Branch_ID	Total_Balance
1	50000
2	75000
3	62000

2. Average Loan Amount by Loan Type

```
SELECT Loan_Type, AVG(Amount) AS Avg_Loan_Amount
FROM Loan
GROUP BY Loan_Type;
```

Output | Available Tables

Loan_Type	Avg_Loan_Amount
Car Loan	500000
Home Loan	1000000
Personal Loan	300000

3. Number of Transactions per Account

```
SELECT Account_ID, COUNT(*) AS Transaction_Count
FROM BankTransaction
GROUP BY Account_ID;
```

Output | Available Tables

Account_ID	Transaction_Count
301	1
302	1
303	1

4. Maximum and Minimum Account Balances

```
SELECT MAX(Balance) AS Max_Balance, MIN(Balance) AS Min_Balance
FROM Account;
```

Output | Available Tables

Max_Balance	Min_Balance
75000	50000

NESTED QUERIES FOR BANKING SYSTEM

1. Employees and Their Branches

```sql
SELECT
    e.Employee_ID, e.Name AS Employee_Name, e.Position,
    b.Branch_Name, b.Location
FROM
    Employee e
JOIN
    Branch b ON e.Branch_ID = b.Branch_ID;
```

```
+-------------+--------------+----------+--------------+--------+
| Employee_ID | Employee_Name| Position | Branch_Name  |Location|
+-------------+--------------+----------+--------------+--------+
|    101      | Amit Sharma  | Manager  | Main Branch  | Delhi  |
|    102      | Neha Verma   | Clerk    | West Branch  | Mumbai |
|    103      | Rajesh Kumar | Cashier  | East Branch  | Pune   |
+-------------+--------------+----------+--------------+--------+
```

2. Customers and Their Accounts

```sql
SELECT
    c.Customer_ID, c.Name AS Customer_Name, c.Phone,
    a.Account_ID, a.Account_Type, a.Balance
FROM
    Customer c
JOIN
    Account a ON c.Customer_ID = a.Customer_ID;
```

Text Copy

Customer_ID	Customer_Name	Phone	Account_ID	Account_Type	Balance
201	Karan Mehta	9876543210	301	Savings	50000.00
202	Pooja Singh	9988776655	302	Current	75000.00
203	Ravi Desai	9123456780	303	Savings	62000.00

3. Accounts and Their Branches

```sql
SELECT
    a.Account_ID, a.Account_Type, a.Balance,
    b.Branch_Name, b.Location
FROM
    Account a
JOIN
    Branch b ON a.Branch_ID = b.Branch_ID;
```

```
+------------+--------------+---------+--------------+--------+
| Account_ID | Account_Type | Balance | Branch_Name  |Location|
+------------+--------------+---------+--------------+--------+
|    301     | Savings      | 50000.00| Main Branch  | Delhi  |
|    302     | Current      | 75000.00| West Branch  | Mumbai |
|    303     | Savings      | 62000.00| East Branch  | Pune   |
+------------+--------------+---------+--------------+--------+
```

4.Transactions with Customer Detail

```sql
SELECT
    bt.Transaction_ID, bt.Transaction_Date, bt.Type, bt.Amount,
    c.Name AS Customer_Name, c.Phone
FROM
    BankTransaction bt
JOIN
    Account a ON bt.Account_ID = a.Account_ID
JOIN
    Customer c ON a.Customer_ID = c.Customer_ID;
```

RESULT:Hence the joins is Sucessfully verified for the Banking System.

NORMALISATION FOR BANKING MANAGEMENT SYSTEM

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 (2 NF):**

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 using commas 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:

Step-by-Step Normalization

🟩 First Normal Form (1NF)

Goal: Eliminate repeating groups and ensure atomicity of data.

All your tables already satisfy 1NF:

- Each column contains atomic values.
- Each row is uniquely identified by a primary key.
- No multivalued or composite attributes.

🟩 **All entities are in 1NF.**

🟩 Second Normal Form (2NF)

Goal: Eliminate partial dependencies (i.e., no non-key attribute depends on part of a composite key).

Since all your tables use **single-column primary keys**, there are **no composite keys**, so partial dependencies don't exist.

However, we ensure that:

- All non-key attributes depend **fully** on the primary key.

🟩 **All entities are in 2NF.**

🔵 Third Normal Form (3NF)

Goal: Eliminate transitive dependencies (i.e., non-key attributes should not depend on other non-key attributes).

Let's analyze each table:1. **Branch**

- **Primary Key:** Branch_ID
- No transitive dependencies. 🟩 Already in 3NF.

2. **Employee**

- **Primary Key:** Employee_ID
- Branch_ID is a foreign key, not derived from other non-key attributes. 🟩 Already in 3NF.

3. **Customer**

- **Primary Key:** Customer_ID
- All attributes depend directly on Customer_ID. 🟩 Already in 3NF.

4. **Account**

- **Primary Key:** Account_ID
- Customer_ID and Branch_ID are foreign keys.
- Account_Type and Balance depend only on Account_ID. ✅ Already in 3NF.

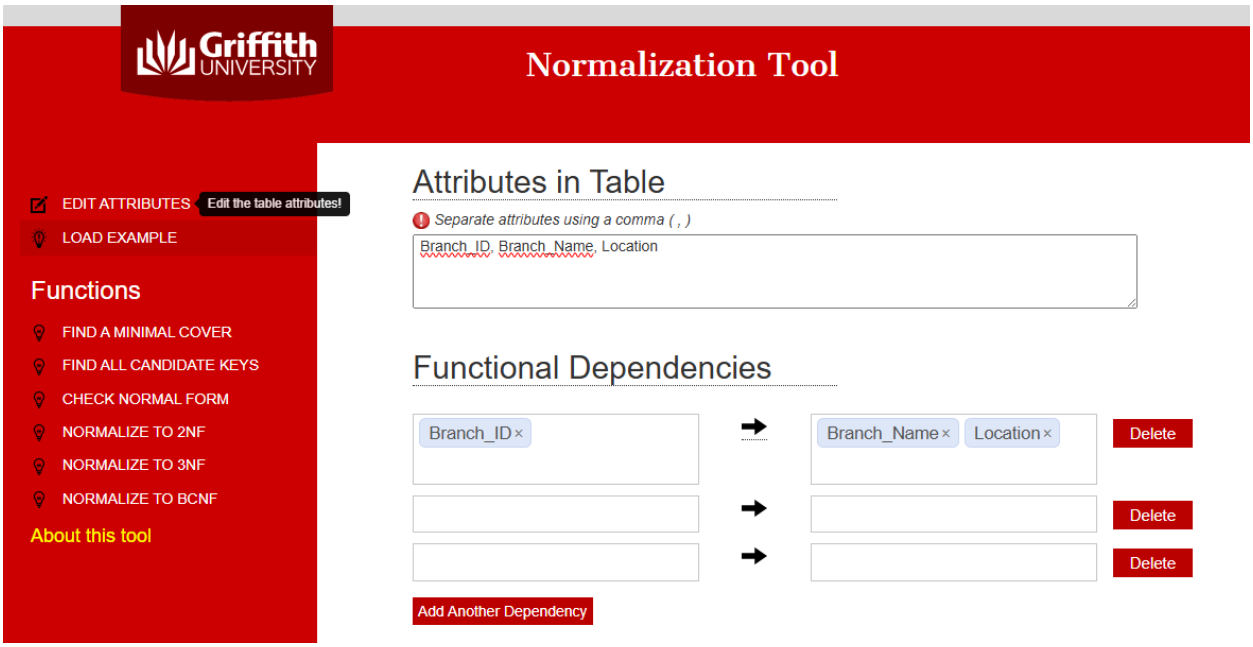
5. **BankTransaction**

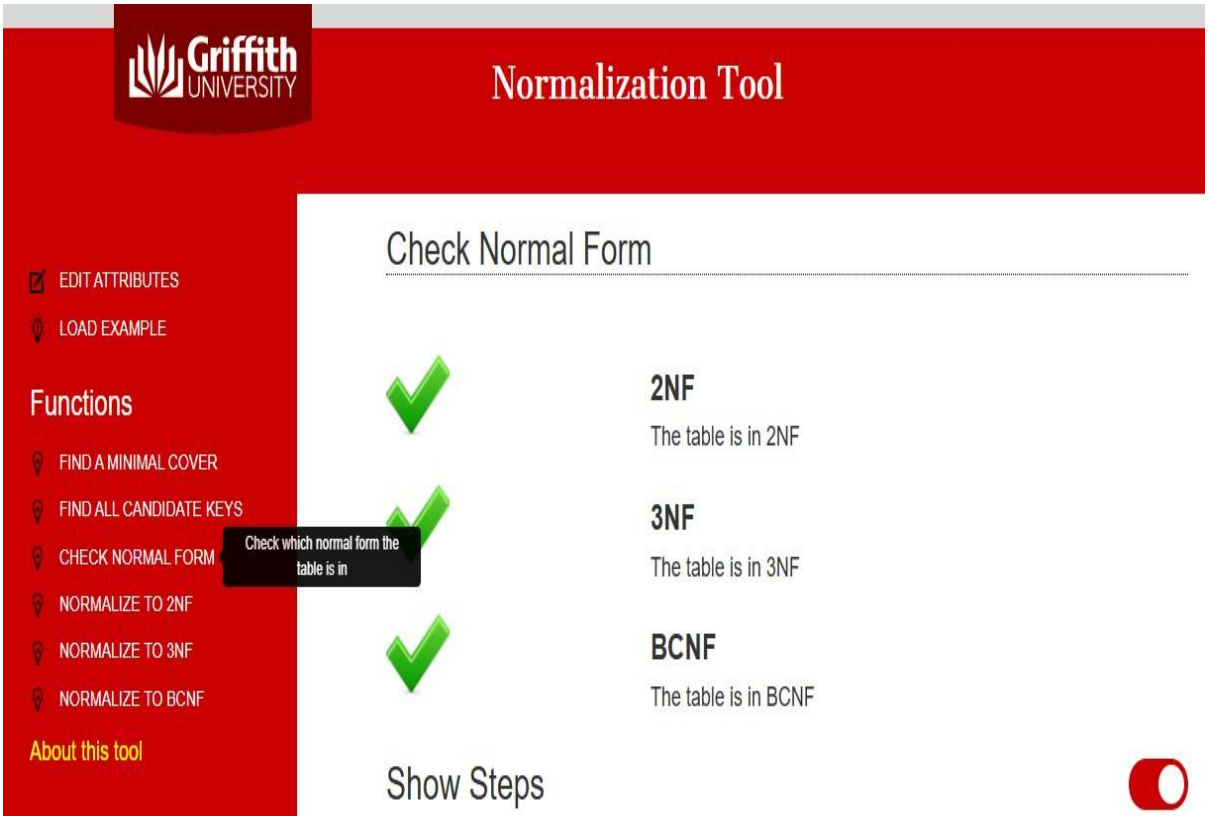
- **Primary Key:** Transaction_ID
- Account_ID is a foreign key.
- Transaction_Date, Type, Amount depend only on Transaction_ID. ✅ Already in 3NF.

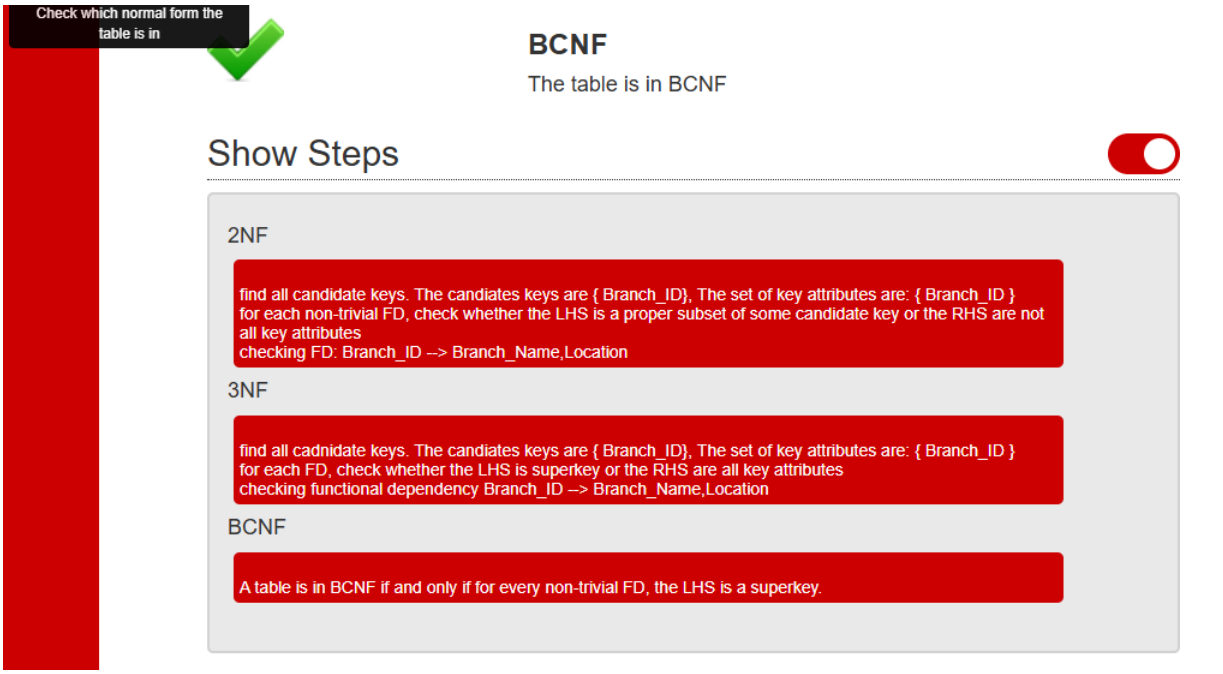
6. **Loan**

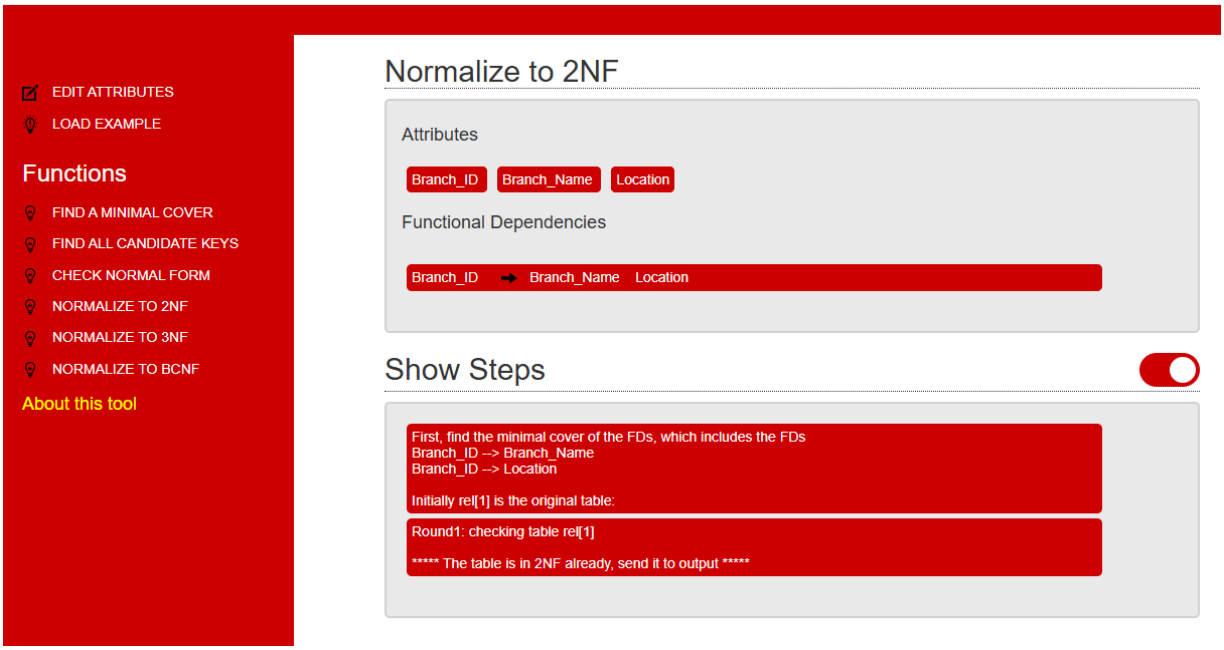
- **Primary Key:** Loan_ID
- Customer_ID and Branch_ID are foreign keys.
- Loan_Type, Amount, Interest_Rate depend only on Loan_ID. ✅ Already in 3NF.

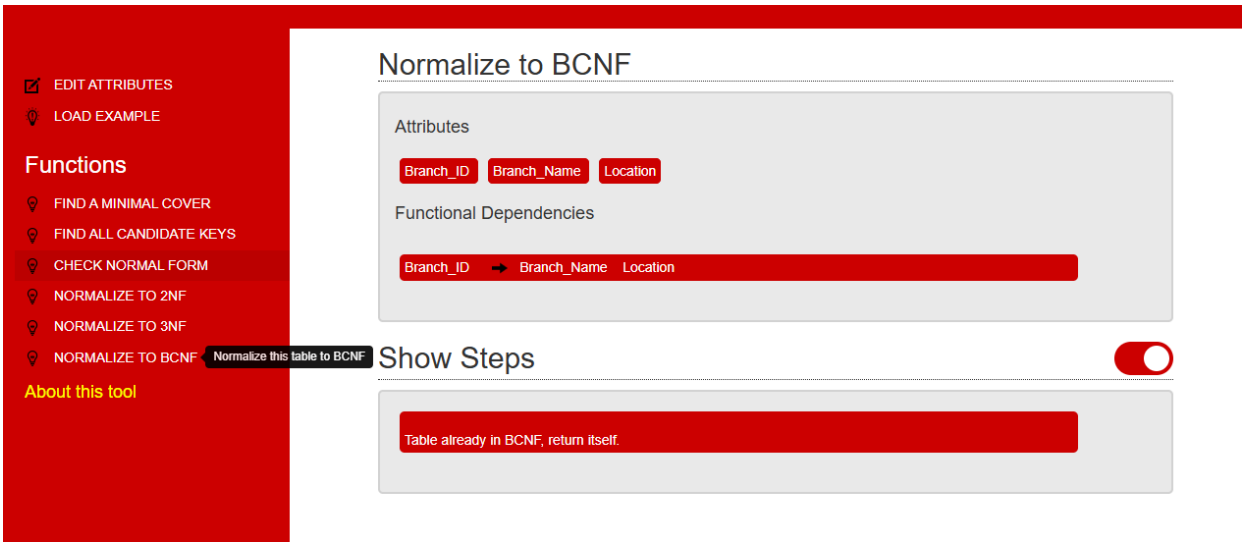
Table	1NF	2NF	3NF
Branch	✅	✅	✅
Employee	✅	✅	✅
Customer	✅	✅	✅
Account	✅	✅	✅
BankTransaction	✅	✅	✅
Loan	✅	✅	✅


Griffith UNIVERSITY
Normalization Tool
EDIT ATTRIBUTES
Edit the table 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
Attributes in Table
Separate attributes using a comma (,)
Branch_ID, Branch_Name, Location
Functional Dependencies
Branch_ID ×
Branch_Name ×
Location ×
Delete
Delete
Delete
Add Another Dependency


Griffith UNIVERSITY
Normalization Tool
EDIT ATTRIBUTES
LOAD EXAMPLE
Functions
FIND A MINIMAL COVER
FIND ALL CANDIDATE KEYS
CHECK NORMAL FORM
Check which normal form the table is in
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

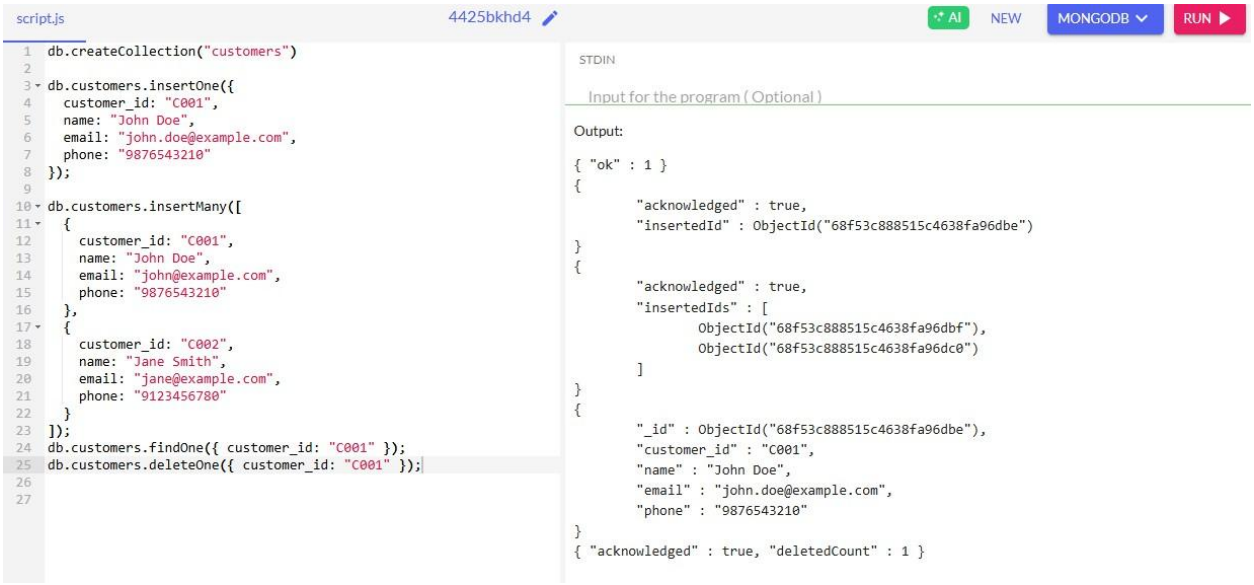

Check which normal form the table is in
BCNF
The table is in BCNF
Show Steps
2NF
find all candidate keys. The candiates keys are { Branch_ID}, The set of key attributes are: { Branch_ID }
for each non-trivial FD, check whether the LHS is a proper subset of some candidate key or the RHS are not all key attributes
checking FD: Branch_ID --> Branch_Name,Location
3NF
find all cadnidate keys. The candiates keys are { Branch_ID}, The set of key attributes are: { Branch_ID }
for each FD, check whether the LHS is superkey or the RHS are all key attributes
checking functional dependency Branch_ID --> Branch_Name,Location
BCNF
A table is in BCNF if and only if for every non-trivial FD, the LHS is a superkey.


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
Normalize to 2NF
Attributes
Branch_ID
Branch_Name
Location
Functional Dependencies
Branch_ID → Branch_Name Location
Show Steps
First, find the minimal cover of the FDs, which includes the FDs
Branch_ID --> Branch_Name
Branch_ID --> Location
Initially rel[1] is the original table:
Round1: checking table rel[1]
***** The table is in 2NF already, send it to output *****



RESULT: Hence the normalisation concept is successfully verified

IMPLEMENTATION OF DOCUMENT DATABASE FOR BANKING SYSTEM




```
db.createCollection("customers")

db.customers.insertOne({
  customer_id: "C001",
  name: "John Doe",
  email: "john.doe@example.com",
  phone: "9876543210"
});

db.customers.insertMany([
  {
    customer_id: "C001",
    name: "John Doe",
    email: "john@example.com",
    phone: "9876543210"
  },
  {
    customer_id: "C002",
    name: "Jane Smith",
    email: "jane@example.com",
    phone: "9123456780"
  }
]);
db.customers.findOne({ customer_id: "C001" });
db.customers.deleteOne({ customer_id: "C001" });
```

```
db.createCollection("accounts")
db.accounts.insertOne({
  account_id: "A001",
  customer_id: "C001",
  account_type: "Savings",
  balance: 5000
})
db.accounts.insertMany([
  {
    account_id: "A001",
    customer_id: "C001",
    account_type: "Savings",
    balance: 5000
  },
  {
    account_id: "A002",
    customer_id: "C002",
    account_type: "Current",
    balance: 10000
  }
])
db.accounts.findOne({ account_id: "A001" })
db.accounts.deleteOne({ account_id: "A001" })
```

Output

```
  }
}
mycompiler_mongodb> {
  _id: ObjectId('68f53ecb27b544309e6b128c'),
  account_id: 'A001',
  customer_id: 'C001',
  account_type: 'Savings',
  balance: 5000
}
mycompiler_mongodb> { acknowledged: true, deletedCount: 1 }
mycompiler_mongodb>

[Execution complete with exit code 0]
```

Output

```
 mycompiler_mongodb> { ok: 1 }
mycompiler_mongodb> ... ... ... ... ... {
  acknowledged: true,
  insertedId: ObjectId('68f53ecb27b544309e6b128c')
}
mycompiler_mongodb> ... ... ... ... ... ... ... ... ... ... ... ... ... {
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68f53ecb27b544309e6b128d'),
    '1': ObjectId('68f53ecb27b544309e6b128e')
  }
}
```

```
db.createCollection("transactions");
db.transactions.insertOne({
  transaction_id: "T001",
  account_id: "A001",
  type: "Deposit",
  amount: 1000,
  date: new Date("2025-10-19T10:00:00Z")});
db.transactions.insertMany([
 {
    transaction_id: "T001",
    account_id: "A001",
    type: "Deposit",
    amount: 1000,
    date: new Date("2025-10-19T10:00:00Z")},{
    transaction_id: "T002",
    account_id: "A002",
    type: "Withdrawal",
    amount: 500,
    date: new Date("2025-10-20T11:30:00Z") }
])
db.transactions.findOne({ transaction_id: "T001" })
db.transactions.deleteOne({ transaction_id: "T001" })
```

Output

```
 mycompiler_mongodb> { ok: 1 }
mycompiler_mongodb> ... ... ... ... ... ... {
  acknowledged: true,
  insertedId: ObjectId('68f540a6ac59e4fff66b128c')
}
mycompiler_mongodb> ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... {
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68f540a6ac59e4fff66b128d'),
    '1': ObjectId('68f540a6ac59e4fff66b128e')
  }
}
```

Output

```
}
mycompiler_mongodb> {
  _id: ObjectId('68f540a6ac59e4fff66b128c'),
  transaction_id: 'T001',
  account_id: 'A001',
  type: 'Deposit',
  amount: 1000,
  date: ISODate('2025-10-19T10:00:00.000Z')
}
mycompiler_mongodb> { acknowledged: true, deletedCount: 1 }
mycompiler_mongodb>

[Execution complete with exit code 0]
```

RESULT: Hence the document database is successfully completed