**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.

1. Recovering a NOARCHIVELOG Database with Incremental Backups
2. Restoring the Server Parameter File
3. 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"})

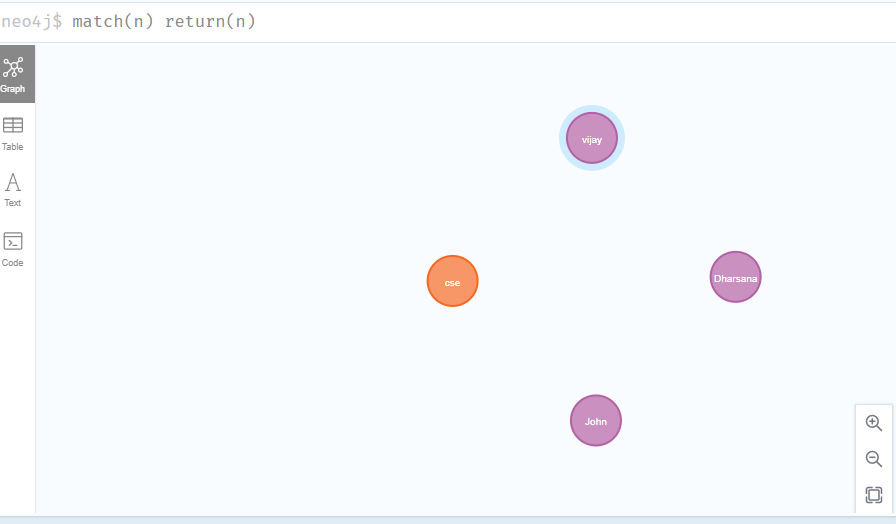
**OUTPUT:**

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

**Select all the nodes in your database using match command**

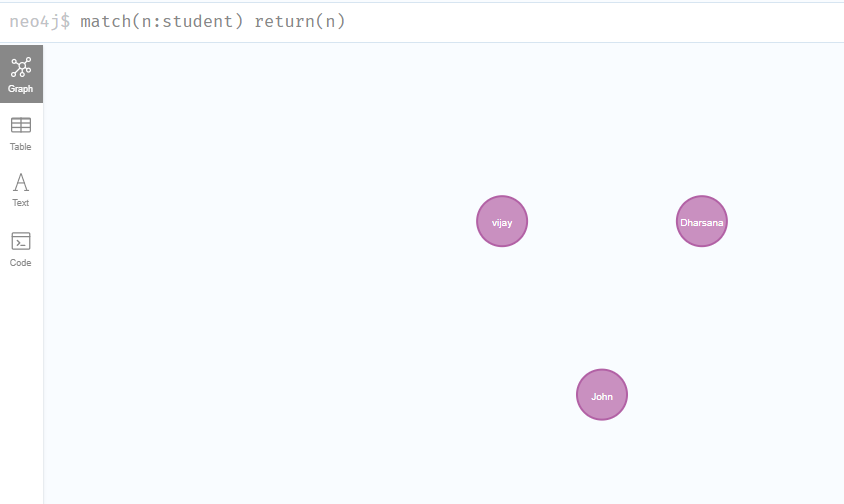
* **match(n) return(n)**

**OUTPUT**



* **match(n:student) return(n)**

**OUTPUT:**



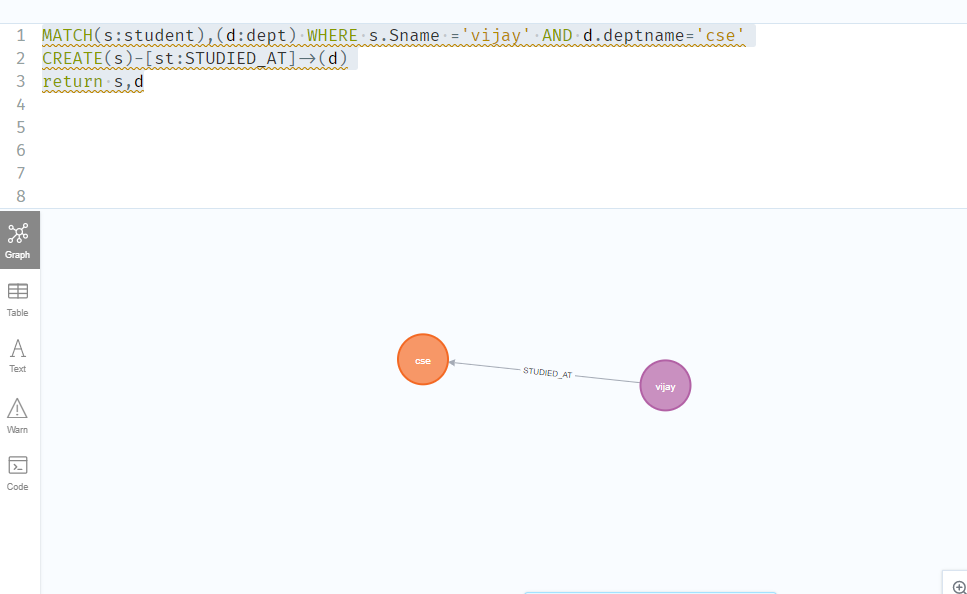
1. **Create relationship between student and cse** .

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

CREATE(s)-[st:STUDIED\_AT]->(d)

return s,d

**OUTPUT:**

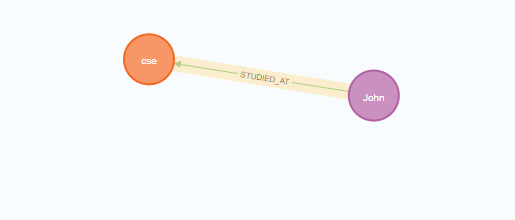


MATCH(s:student),(d:dept) WHERE s.Sname ='John' AND d.deptname='cse'

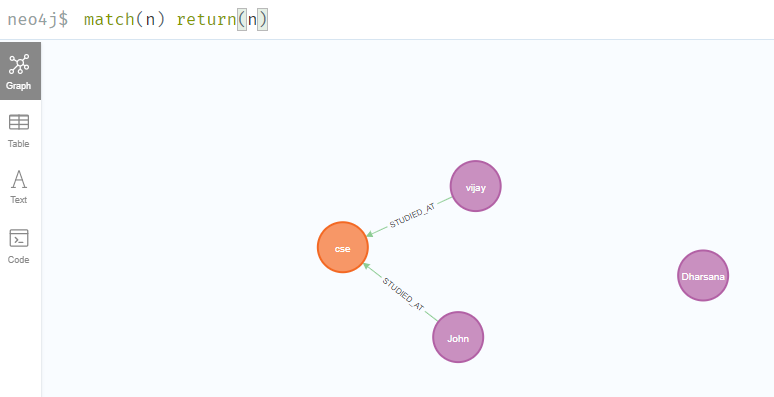
CREATE(s)-[st:STUDIED\_AT]->(d)

return s,d

OUTPUT:



**match(n) return(n)**

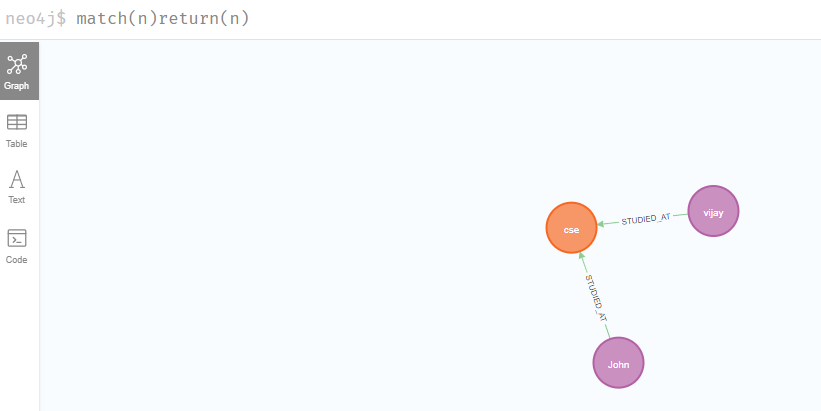


1. **Delete a node from student**

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.