ADT Practical Slip Solutions.

SLIP NO. 1

Q.1) University Database in Neo4j (10 Marks)

Data Creation:

```
// Create Students
CREATE (s1:Student {name: 'Anita', roll: 101})
CREATE (s2:Student {name: 'Rahul', roll: 102})
CREATE (s3:Student {name: 'Priya', roll: 103})
CREATE (s4:Student {name: 'Amit', roll: 104})
CREATE (s5:Student {name: 'Neha', roll: 105})

// Create Courses
CREATE (c1:Course {name: 'Database Systems', code: 'CS101'})
CREATE (c2:Course {name: 'Machine Learning', code: 'CS102'})
CREATE (c3:Course {name: 'Data Structures', code: 'CS103'})

// Create Professors
CREATE (p1:Professor {name: 'Dr. Sharma', dept: 'CS'})
CREATE (p2:Professor {name: 'Dr. Mehta', dept: 'CS'})
CREATE (p3:Professor {name: 'Dr. Gupta', dept: 'IT'})

// Create Clubs
CREATE (cl1:Club {name: 'Coding Club'})
CREATE (cl2:Club {name: 'Music Club'})
```

```
CREATE (cl3:Club {name: 'Sports Club'})

// Create Relationships
CREATE (s1)-[:ENROLLED_IN]->(c1)
CREATE (s2)-[:ENROLLED_IN]->(c1)
CREATE (s3)-[:ENROLLED_IN]->(c2)
CREATE (s4)-[:ENROLLED_IN]->(c1)
CREATE (s5)-[:ENROLLED_IN]->(c3)

CREATE (p1)-[:TEACHES]->(c1)
CREATE (p1)-[:TEACHES]->(c2)
CREATE (p2)-[:TEACHES]->(c3)

CREATE (s1)-[:MEMBER_OF]->(cl1)
CREATE (s1)-[:MEMBER_OF]->(cl2)
CREATE (s2)-[:MEMBER_OF]->(cl3)

CREATE (p2)-[:ADVISOR]->(s1)
CREATE (p2)-[:ADVISOR]->(s2)
CREATE (p1)-[:ADVISOR]->(s3)
```

a. List the names of students enrolled in course "Database Systems".

```
MATCH (s:Student)-[:ENROLLED_IN]->(c:Course {name: 'Database Systems'})
RETURN s.name AS StudentName
```

b. List the courses taught by professor "Dr. Sharma".

```
MATCH (p:Professor {name: 'Dr. Sharma'})-[:TEACHES]->(c:Course)
RETURN c.name AS CourseName
```

c. Find the count of students enrolled in each course.

```
MATCH (s:Student)-[:ENROLLED_IN]->(c:Course)
RETURN c.name AS CourseName, COUNT(s) AS StudentCount
```

d. List the clubs in which student "Anita" is a member.

```
MATCH (s:Student {name: 'Anita'})-[:MEMBER_OF]->(cl:Club)
RETURN cl.name AS ClubName
```

e. List the students advised by "Dr. Mehta".

```
MATCH (p:Professor {name: 'Dr. Mehta'})-[:ADVISOR]->(s:Student)
RETURN s.name AS StudentName
```

Q.2) StudentDB in MongoDB (10 Marks)

i) Create Database and Insert Five Documents:

```
use StudentDB

db.students.insertMany([
    {name: "Alice", age: 21, department: "Computer Science", grade: "A"},
    {name: "Bob", age: 22, department: "Electronics", grade: "B+"},
    {name: "Charlie", age: 20, department: "Computer Science", grade: "A-"},
    {name: "Diana", age: 23, department: "Mechanical", grade: "B"},
    {name: "Eve", age: 21, department: "Computer Science", grade: "A"}
])
```

ii) Retrieve all students from "Computer Science" department:

```
db.students.find({department: "Computer Science"})
```

iii) Update the grade of "Alice" to "A+":

```
db.students.updateOne({name: "Alice"}, {$set: {grade: "A+"}})
```

Q.3) Book-Author Database in SQL (10 Marks)

Data Creation:

```
-- Create Tables
CREATE TABLE Author (
    Ano INT PRIMARY KEY,
    Aname VARCHAR(50) NOT NULL
);

CREATE TABLE Book (
    Bno INT PRIMARY KEY,
    Bname VARCHAR(100),
    Pubname VARCHAR(50) NOT NULL,
    Price DECIMAL(10,2) CHECK (Price > 0)
);

CREATE TABLE Book_Author (
    Bno INT,
    Ano INT,
    PRIMARY KEY (Bno, Ano),
    FOREIGN KEY (Bno) REFERENCES Book(Bno),
    FOREIGN KEY (Ano) REFERENCES Author(Ano)
);

-- Insert Data
```

```
INSERT INTO Author VALUES (1, 'Lee'), (2, 'Smith'), (3, 'Johnson'), (4, 'Brown'), (5, 'Wi

INSERT INTO Book VALUES
(101, 'Zero to One', 'Weilly', 450),
(102, 'Database Concepts', 'Pearson', 550),
(103, 'Zen Programming', 'Weilly', 350),
(104, 'Data Science', 'McGraw', 600),
(105, 'Zephyr Guide', 'Pearson', 400);

INSERT INTO Book_Author VALUES (101, 1), (102, 2), (103, 3), (104, 1), (105, 4);
```

a) Count number of books of each publisher:

```
SELECT Pubname, COUNT(*) AS BookCount
FROM Book
GROUP BY Pubname;
```

b) Display Author-wise list of books:

```
SELECT A.Aname, B.Bname
FROM Author A
JOIN Book_Author BA ON A.Ano = BA.Ano
JOIN Book B ON BA.Bno = B.Bno
ORDER BY A.Aname;
```

c) Display all details of book whose name starts with 'Z':

```
SELECT * FROM Book WHERE Bname LIKE 'Z%';
```

d) Display the book-name whose author is 'Lee' and publisher is 'Weilly':

```
SELECT B.Bname
FROM Book B
JOIN Book_Author BA ON B.Bno = BA.Bno
JOIN Author A ON BA.Ano = A.Ano
WHERE A.Aname = 'Lee' AND B.Pubname = 'Weilly';
```

SLIP NO. 2

Q.1) Library Management System in Neo4j (10 Marks)

Data Creation:

```
// Create Books
CREATE (b1:Book {title: 'Five Point Someone', isbn: 'B001'})
CREATE (b2:Book {title: 'Python Programming', isbn: 'B002'})
CREATE (b3:Book {title: 'The Alchemist', isbn: 'B003'})
CREATE (b4:Book {title: '2 States', isbn: 'B004'})
CREATE (b5:Book {title: 'Data Science Handbook', isbn: 'B005'})

// Create Authors
CREATE (a1:Author {name: 'Chetan Bhagat'})
CREATE (a2:Author {name: 'Paulo Coelho'})
CREATE (a3:Author {name: 'Mark Lutz'})

// Create Members
CREATE (m1:Member {name: 'Ravi', id: 'M001'})
CREATE (m2:Member {name: 'Sunita', id: 'M002'})
CREATE (m3:Member {name: 'Arun', id: 'M003'})

// Create Categories
CREATE (cat1:Category {name: 'Fiction'})
CREATE (cat2:Category {name: 'Science Fiction'})
CREATE (cat3:Category {name: 'Programming'})

// Create Relationships
CREATE (b1)-[:WRITTEN_BY]->(a1)
CREATE (b4)-[:WRITTEN_BY]->(a1)
CREATE (b2)-[:WRITTEN_BY]->(a3)
CREATE (b3)-[:WRITTEN_BY]->(a2)

CREATE (b1)-[:BORROWED_BY]->(m1)
CREATE (b2)-[:BORROWED_BY]->(m1)
CREATE (b3)-[:BORROWED_BY]->(m2)
CREATE (b5)-[:BORROWED_BY]->(m3)

CREATE (b1)-[:BELONGS_TO]->(cat1)
CREATE (b2)-[:BELONGS_TO]->(cat3)
CREATE (b3)-[:BELONGS_TO]->(cat2)
CREATE (b5)-[:BELONGS_TO]->(cat2)
```

a. List the books written by author "Chetan Bhagat".

```
MATCH (b:Book)-[:WRITTEN_BY]->(a:Author {name: 'Chetan Bhagat'})
RETURN b.title AS BookTitle
```

b. List all books borrowed by member "Ravi".

```
MATCH (b:Book)-[:BORROWED_BY]->(m:Member {name: 'Ravi'})
RETURN b.title AS BookTitle
```

c. Find the number of books in each category.

```
MATCH (b:Book)-[:BELONGS_TO]->(cat:Category)
RETURN cat.name AS Category, COUNT(b) AS BookCount
```

d. List all categories for the book "Python Programming".

```
MATCH (b:Book {title: 'Python Programming'})-[:BELONGS_TO]->(cat:Category)
RETURN cat.name AS Category
```

e. List all members who borrowed books from category "Science Fiction".

```
MATCH (b:Book)-[:BELONGS_TO]->(cat:Category {name: 'Science Fiction'})
MATCH (b)-[:BORROWED_BY]->(m:Member)
RETURN DISTINCT m.name AS MemberName
```

Q.2) BookDB in MongoDB (10 Marks)

i) Create Database and Insert Five Documents:

```
use BookDB

db.books.insertMany([
    {title: "The Great Gatsby", author: "F. Scott Fitzgerald", year: 2020, price: 350},
    {title: "To Kill a Mockingbird", author: "Harper Lee", year: 2018, price: 400},
    {title: "1984", author: "George Orwell", year: 2019, price: 300},
    {title: "Pride and Prejudice", author: "Jane Austen", year: 2020, price: 280},
    {title: "Animal Farm", author: "George Orwell", year: 2021, price: 250}
])
```

ii) Find books by a specific author:

```
db.books.find({author: "George Orwell"})
```

iii) Find books published in a specific year:

```
db.books.find({year: 2020})
```

Q.3) Student-Teacher Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Student (
    sno INT PRIMARY KEY,
    s_name VARCHAR(50),
    s_class VARCHAR(20),
    s_addr VARCHAR(100)
);

CREATE TABLE Teacher (
    tno INT PRIMARY KEY,
    t_name VARCHAR(50),
    qualification VARCHAR(30),
    experience INT
);

CREATE TABLE Student_Teacher (
    sno INT,
    tno INT,
    Subject VARCHAR(50),
    PRIMARY KEY (sno, tno),
    FOREIGN KEY (sno) REFERENCES Student(sno),
    FOREIGN KEY (tno) REFERENCES Teacher(tno)
);

INSERT INTO Student VALUES
(1, 'Amit', 'SYBSc_IT', 'Pune'),
(2, 'Priya', 'TYBSc_IT', 'Mumbai'),
(3, 'Rahul', 'SYBSc_IT', 'Delhi'),
(4, 'Neha', 'FYBSc_IT', 'Pune'),
(5, 'Arun', 'SYBSc_IT', 'Chennai');

INSERT INTO Teacher VALUES
(101, 'Prof. Sharma', 'M.Tech', 10),
(102, 'Prof. Verma', 'PhD', 15),
(103, 'Prof. Gupta', 'M.Sc', 8);

INSERT INTO Student_Teacher VALUES
(1, 101, 'Database'), (2, 101, 'Database'), (3, 102, 'Networks'),
(1, 102, 'Networks'), (4, 103, 'Programming'), (5, 101, 'Database');
```

a) List the students details who study in class 'SYBSc_IT':

```
SELECT * FROM Student WHERE s_class = 'SYBSc_IT';
```

b) Find subject wise teacher details:

```
SELECT ST.Subject, T.*
FROM Teacher T
JOIN Student_Teacher ST ON T.tno = ST.tno
GROUP BY ST.Subject, T.tno, T.t_name, T.qualification, T.experience;
```

c) List total number of subjects taught by each teacher:

```
SELECT T.t_name, COUNT(DISTINCT ST.Subject) AS SubjectCount
FROM Teacher T
JOIN Student_Teacher ST ON T.tno = ST.tno
GROUP BY T.tno, T.t_name;
```

d) Alter table student to add attribute 'Phone_no':

```
ALTER TABLE Student ADD Phone_no VARCHAR(15);
```

SLIP NO. 3

Q.1) Hospital Management in Neo4j (10 Marks)

Data Creation:

```
// Create Doctors
CREATE (d1:Doctor {name: 'Dr. Verma', specialization: 'Cardiology'})
CREATE (d2:Doctor {name: 'Dr. Patel', specialization: 'Neurology'})
CREATE (d3:Doctor {name: 'Dr. Singh', specialization: 'Orthopedics'})
CREATE (d4:Doctor {name: 'Dr. Rao', specialization: 'Neurology'})

// Create Patients
CREATE (p1:Patient {name: 'Sita', age: 45})
CREATE (p2:Patient {name: 'Ramesh', age: 55})
CREATE (p3:Patient {name: 'Geeta', age: 35})
CREATE (p4:Patient {name: 'Mohan', age: 60})

// Create Departments
CREATE (dep1:Department {name: 'Cardiology'})
CREATE (dep2:Department {name: 'Neurology'})
CREATE (dep3:Department {name: 'Orthopedics'})

// Create Treatments
CREATE (t1:Treatment {name: 'Bypass Surgery', cost: 500000})
CREATE (t2:Treatment {name: 'Brain Scan', cost: 25000})
CREATE (t3:Treatment {name: 'Knee Replacement', cost: 300000})

// Relationships
```

```
CREATE (d1)-[:WORKS_IN]->(dep1)
CREATE (d2)-[:WORKS_IN]->(dep2)
CREATE (d3)-[:WORKS_IN]->(dep3)
CREATE (d4)-[:WORKS_IN]->(dep2)

CREATE (p1)-[:ASSIGNED_TO]->(d1)
CREATE (p2)-[:ASSIGNED_TO]->(d1)
CREATE (p3)-[:ASSIGNED_TO]->(d2)
CREATE (p4)-[:ASSIGNED_TO]->(d4)

CREATE (p1)-[:UNDERGOES]->(t1)
CREATE (p1)-[:UNDERGOES]->(t2)
CREATE (p3)-[:UNDERGOES]->(t2)

CREATE (t1)-[:PROVIDED_BY]->(dep1)
CREATE (t2)-[:PROVIDED_BY]->(dep2)
CREATE (t3)-[:PROVIDED_BY]->(dep3)
```

a. List all patients assigned to doctor "Dr. Verma".

```
MATCH (p:Patient)-[:ASSIGNED_TO]->(d:Doctor {name: 'Dr. Verma'})
RETURN p.name AS PatientName
```

b. List the treatments provided by department "Cardiology".

```
MATCH (t:Treatment)-[:PROVIDED_BY]->(dep:Department {name: 'Cardiology'})
RETURN t.name AS TreatmentName
```

c. Find the number of doctors in each department.

```
MATCH (d:Doctor)-[:WORKS_IN]->(dep:Department)
RETURN dep.name AS Department, COUNT(d) AS DoctorCount
```

d. Delete all treatments undergone by patient "Sita".

```
MATCH (p:Patient {name: 'Sita'})-[r:UNDERGOES]->(t:Treatment)
DELETE r
```

e. List all patients assigned to doctors of "Neurology" department.

```
MATCH (d:Doctor)-[:WORKS_IN]->(dep:Department {name: 'Neurology'})
MATCH (p:Patient)-[:ASSIGNED_TO]->(d)
RETURN DISTINCT p.name AS PatientName
```

Q.2) Olympic Information System in MongoDB (10 Marks)

Data Creation:

```
use OlympicDB

// Create Olympics collection
db.olympics.insertMany([
    {year: 2008, city: "Beijing", country: "China", games_count: 302},
    {year: 2012, city: "London", country: "UK", games_count: 302},
    {year: 2015, city: "Rio", country: "Brazil", games_count: 306},
    {year: 2016, city: "Rio", country: "Brazil", games_count: 306},
    {year: 2021, city: "Tokyo", country: "Japan", games_count: 339}
])

// Create Games collection
db.games.insertMany([
    {name: "Swimming", type: "Aquatics", olympic_year: 2008},
    {name: "Athletics", type: "Track", olympic_year: 2008},
    {name: "Gymnastics", type: "Artistic", olympic_year: 2015},
    {name: "Basketball", type: "Team", olympic_year: 2012},
    {name: "Tennis", type: "Racket", olympic_year: 2015}
])

// Create Countries collection
db.countries.insertMany([
    {name: "India", code: "IND", olympic_year: 2008, games: ["Athletics", "Shooting"]},
    {name: "USA", code: "USA", olympic_year: 2015, games: ["Swimming", "Basketball"]},
    {name: "China", code: "CHN", olympic_year: 2008, games: ["Gymnastics", "Diving"]},
    {name: "Brazil", code: "BRA", olympic_year: 2015, games: ["Football", "Volleyball"]},
    {name: "Japan", code: "JPN", olympic_year: 2015, games: ["Judo", "Wrestling"]}
])
```

1. List the details of Olympic Games conducted during the year 2008:

```
db.olympics.find({year: 2008})
```

2. List the names of the countries participating in 2015 Olympics, in ascending order:

```
db.countries.find({olympic_year: 2015}, {name: 1, _id: 0}).sort({name: 1})
```

Q.3) Property-Owner Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Owner (
    o_name VARCHAR(50) PRIMARY KEY,
    o_address VARCHAR(100),
    phone VARCHAR(15)
);

CREATE TABLE Property (
    pno INT PRIMARY KEY,
    description VARCHAR(200),
    area VARCHAR(50),
    o_name VARCHAR(50),
    FOREIGN KEY (o_name) REFERENCES Owner(o_name)
);

INSERT INTO Owner VALUES
('John', '123 MG Road', '9876543210'),
('Mary', '456 Park Street', '9876543211'),
('Peter', '789 Lake View', '9876543212'),
('Sarah', '101 Hill Road', '9876543213'),
('David', '202 Garden Lane', '9876543214');

INSERT INTO Property VALUES
(1, '3BHK Apartment', 'Bangalore', 'John'),
(2, '2BHK Flat', 'Bangalore', 'John'),
(3, 'Villa', 'Mumbai', 'Mary'),
(4, 'Plot', 'Delhi', 'Peter'),
(5, 'Office Space', 'Bangalore', 'Sarah');
```

a) List details of property where area is 'Bangalore':

```
SELECT * FROM Property WHERE area = 'Bangalore';
```

b) Update phone no. of 'John' to 988151110:

```
UPDATE Owner SET phone = '988151110' WHERE o_name = 'John';
```

c) Count owner wise number of property:

```
SELECT o_name, COUNT(*) AS PropertyCount
FROM Property
GROUP BY o_name;
```

d) Alter table owner to add owner_age attribute:

```
ALTER TABLE Owner ADD owner_age INT;
```

SLIP NO. 4

Q.1) Online Shopping System in Neo4j (10 Marks)

Data Creation:

```
// Create Customers
CREATE (c1:Customer {name: 'Amit', email: 'amit@email.com'})
CREATE (c2:Customer {name: 'Priya', email: 'priya@email.com'})
CREATE (c3:Customer {name: 'Rahul', email: 'rahul@email.com'})

// Create Products
CREATE (pr1:Product {name: 'Laptop', price: 75000})
CREATE (pr2:Product {name: 'Mobile', price: 25000})
CREATE (pr3:Product {name: 'Tablet', price: 35000})
CREATE (pr4:Product {name: 'Headphones', price: 5000})

// Create Categories
CREATE (cat1:Category {name: 'Electronics'})
CREATE (cat2:Category {name: 'Accessories'})
CREATE (cat3:Category {name: 'Computers'})

// Create Orders
CREATE (o1:Order {order_id: 'ORD123', date: '2024-01-15', total: 100000})
CREATE (o2:Order {order_id: 'ORD124', date: '2024-01-16', total: 25000})
CREATE (o3:Order {order_id: 'ORD125', date: '2024-01-17', total: 35000})

// Relationships
CREATE (c1)-[:PLACED]->(o1)
CREATE (c1)-[:PLACED]->(o3)
CREATE (c2)-[:PLACED]->(o2)

CREATE (o1)-[:CONTAINS]->(pr1)
CREATE (o1)-[:CONTAINS]->(pr2)
CREATE (o2)-[:CONTAINS]->(pr2)
CREATE (o3)-[:CONTAINS]->(pr3)

CREATE (pr1)-[:BELONGS_TO]->(cat1)
CREATE (pr1)-[:BELONGS_TO]->(cat3)
CREATE (pr2)-[:BELONGS_TO]->(cat1)
CREATE (pr3)-[:BELONGS_TO]->(cat1)
CREATE (pr4)-[:BELONGS_TO]->(cat2)
```

a. List all orders placed by customer "Amit".

```
MATCH (c:Customer {name: 'Amit'})-[:PLACED]->(o:Order)
RETURN o.order_id AS OrderID, o.date AS Date, o.total AS Total
```

b. List all products belonging to category "Electronics".

```
MATCH (p:Product)-[:BELONGS_TO]->(cat:Category {name: 'Electronics'})
RETURN p.name AS ProductName
```

c. Find the count of products ordered by each customer.

```
MATCH (c:Customer)-[:PLACED]->(o:Order)-[:CONTAINS]->(p:Product)
RETURN c.name AS Customer, COUNT(p) AS ProductsOrdered
```

d. Delete all categories of products in order "ORD123".

```
MATCH (o:Order {order_id: 'ORD123'})-[:CONTAINS]->(p:Product)-[r:BELONGS_TO]->(cat:Catego
DELETE r
```

e. List all customers who placed orders containing "Laptop".

```
MATCH (c:Customer)-[:PLACED]->(o:Order)-[:CONTAINS]->(p:Product {name: 'Laptop'})
RETURN DISTINCT c.name AS CustomerName
```

Q.2) Sales System in MongoDB (10 Marks)

Data Creation:

```
use SalesDB

// Products collection
db.products.insertMany([
    {product_id: "P001", name: "Laptop", price: 75000, stock: 50},
    {product_id: "P002", name: "Mobile", price: 25000, stock: 100},
    {product_id: "P003", name: "Tablet", price: 35000, stock: 75},
    {product_id: "P004", name: "Monitor", price: 15000, stock: 60},
    {product_id: "P005", name: "Keyboard", price: 2000, stock: 200}
])

// Customers collection
db.customers.insertMany([
    {cust_id: "C001", name: "Amit", email: "amit@email.com", city: "Pune"},
    {cust_id: "C002", name: "Priya", email: "priya@email.com", city: "Mumbai"},
    {cust_id: "C003", name: "Rahul", email: "rahul@email.com", city: "Delhi"},
```

```
    {cust_id: "C004", name: "Neha", email: "neha@email.com", city: "Bangalore"},
    {cust_id: "C005", name: "Arun", email: "arun@email.com", city: "Chennai"}
])

// Orders collection
db.orders.insertMany([
    {order_id: "O001", cust_id: "C001", products: ["P001", "P002"], value: 100000, date:
    {order_id: "O002", cust_id: "C002", products: ["P003"], value: 35000, date: "2024-01-
    {order_id: "O003", cust_id: "C003", products: ["P001", "P004"], value: 90000, date: '
    {order_id: "O004", cust_id: "C004", products: ["P002", "P005"], value: 27000, date: '
    {order_id: "O005", cust_id: "C005", products: ["P001"], value: 75000, date: "2024-01-
])

// Invoices collection
db.invoices.insertMany([
    {invoice_id: "I001", order_id: "O001", amount: 100000, status: "Paid"},
    {invoice_id: "I002", order_id: "O002", amount: 35000, status: "Paid"},
    {invoice_id: "I003", order_id: "O003", amount: 90000, status: "Pending"},
    {invoice_id: "I004", order_id: "O004", amount: 27000, status: "Paid"},
    {invoice_id: "I005", order_id: "O005", amount: 75000, status: "Pending"}
])
```

1. List all products in the inventory:

```
db.products.find()
```

2. List the details of orders with a value > 10000:

```
db.orders.find({value: {$gt: 10000}})
```

Q.3) Game-Player Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Game (
    gno INT PRIMARY KEY,
    gname VARCHAR(50),
    no_of_player INT,
    coach_name VARCHAR(50),
    captain VARCHAR(50)
);

CREATE TABLE Player (
    p_no INT PRIMARY KEY,
    p_name VARCHAR(50)
);

CREATE TABLE Game_Player (
    gno INT,
```

```
    p_no INT,
    PRIMARY KEY (gno, p_no),
    FOREIGN KEY (gno) REFERENCES Game(gno),
    FOREIGN KEY (p_no) REFERENCES Player(p_no)
);

INSERT INTO Game VALUES
(1, 'Hockey', 11, 'Coach A', 'Captain X'),
(2, 'Football', 11, 'Coach B', 'Captain Y'),
(3, 'Cricket', 15, 'Coach C', 'Captain Z'),
(4, 'Basketball', 5, 'Coach D', 'Captain W');

INSERT INTO Player VALUES
(101, 'Amit'), (102, 'Zeben'), (103, 'Rahul'), (104, 'Priya'), (105, 'Neha');

INSERT INTO Game_Player VALUES
(1, 101), (1, 103), (2, 101), (2, 104), (3, 103), (3, 105), (1, 102);
```

a) List the names of players of game 'Hockey':

```
SELECT P.p_name
FROM Player P
JOIN Game_Player GP ON P.p_no = GP.p_no
JOIN Game G ON GP.gno = G.gno
WHERE G.gname = 'Hockey';
```

b) Delete the information of Player 'Zeben':

```
DELETE FROM Game_Player WHERE p_no = (SELECT p_no FROM Player WHERE p_name = 'Zeben');
DELETE FROM Player WHERE p_name = 'Zeben';
```

c) List the names of player playing more than one game:

```
SELECT P.p_name
FROM Player P
JOIN Game_Player GP ON P.p_no = GP.p_no
GROUP BY P.p_no, P.p_name
HAVING COUNT(GP.gno) > 1;
```

d) List the name of game with the highest no_of_player:

```
SELECT gname FROM Game WHERE no_of_player = (SELECT MAX(no_of_player) FROM Game);
```

SLIP NO. 5

Q.1) Movie Database in Neo4j (10 Marks)

Data Creation:

```
// Create Movies
CREATE (m1:Movie {title: '3 Idiots', year: 2009})
CREATE (m2:Movie {title: 'Piku', year: 2015})
CREATE (m3:Movie {title: 'PK', year: 2014})
CREATE (m4:Movie {title: 'Sholay', year: 1975})
CREATE (m5:Movie {title: 'Munna Bhai MBBS', year: 2003})

// Create Actors
CREATE (a1:Actor {name: 'Amitabh Bachchan'})
CREATE (a2:Actor {name: 'Aamir Khan'})
CREATE (a3:Actor {name: 'Sanjay Dutt'})

// Create Directors
CREATE (d1:Director {name: 'Rajkumar Hirani'})
CREATE (d2:Director {name: 'Shoojit Sircar'})

// Create Genres
CREATE (g1:Genre {name: 'Comedy'})
CREATE (g2:Genre {name: 'Drama'})
CREATE (g3:Genre {name: 'Action'})

// Relationships
CREATE (a1)-[:ACTED_IN]->(m2)
CREATE (a1)-[:ACTED_IN]->(m4)
CREATE (a2)-[:ACTED_IN]->(m1)
CREATE (a2)-[:ACTED_IN]->(m3)
CREATE (a3)-[:ACTED_IN]->(m5)

CREATE (m1)-[:DIRECTED_BY]->(d1)
CREATE (m3)-[:DIRECTED_BY]->(d1)
CREATE (m5)-[:DIRECTED_BY]->(d1)
CREATE (m2)-[:DIRECTED_BY]->(d2)

CREATE (m1)-[:HAS_GENRE]->(g1)
CREATE (m1)-[:HAS_GENRE]->(g2)
CREATE (m5)-[:HAS_GENRE]->(g1)
CREATE (m4)-[:HAS_GENRE]->(g3)
```

a. List all movies acted in by "Amitabh Bachchan".

```
MATCH (a:Actor {name: 'Amitabh Bachchan'})-[:ACTED_IN]->(m:Movie)
RETURN m.title AS MovieTitle
```

b. List all movies directed by "Rajkumar Hirani".

```
MATCH (m:Movie)-[:DIRECTED_BY]->(d:Director {name: 'Rajkumar Hirani'})
RETURN m.title AS MovieTitle
```

c. Count the number of movies in each genre.

```
MATCH (m:Movie)-[:HAS_GENRE]->(g:Genre)
RETURN g.name AS Genre, COUNT(m) AS MovieCount
```

d. List all genres of the movie "3 Idiots".

```
MATCH (m:Movie {title: '3 Idiots'})-[:HAS_GENRE]->(g:Genre)
RETURN g.name AS Genre
```

e. List all actors who worked in movies of "Comedy" genre.

```
MATCH (a:Actor)-[:ACTED_IN]->(m:Movie)-[:HAS_GENRE]->(g:Genre {name: 'Comedy'})
RETURN DISTINCT a.name AS ActorName
```

Q.2) LibraryDB in MongoDB (10 Marks)

Data Creation:

```
use LibraryDB

db.books.insertMany([
    {title: "The Great Gatsby", author: "F. Scott Fitzgerald", price: 350},
    {title: "To Kill a Mockingbird", author: "Harper Lee", price: 400},
    {title: "1984", author: "George Orwell", price: 300},
    {title: "Pride and Prejudice", author: "Jane Austen", price: 280},
    {title: "The Catcher in the Rye", author: "J.D. Salinger", price: 320}
])
```

1) Retrieve all books from the collection:

```
db.books.find()
```

2) Find books whose price is greater than 300:

```
db.books.find({price: {$gt: 300}})
```

3) Update the price of "1984" to 275:

```
db.books.updateOne({title: "1984"}, {$set: {price: 275}})
```

Q.3) Banking System in SQL (10 Marks)

Data Creation:

```
CREATE TABLE accounts (
    account_no VARCHAR(10) PRIMARY KEY,
    holder_name VARCHAR(50),
    balance DECIMAL(15,2),
    account_type VARCHAR(20)
);

CREATE TABLE transactions (
    txn_id VARCHAR(10) PRIMARY KEY,
    account_no VARCHAR(10),
    amount DECIMAL(15,2),
    txn_type VARCHAR(20),
    FOREIGN KEY (account_no) REFERENCES accounts(account_no)
);

INSERT INTO accounts VALUES
('A101', 'Amit', 50000, 'Savings'),
('A102', 'Priya', 75000, 'Current'),
('A103', 'Rahul', 30000, 'Savings'),
('A104', 'Neha', 25000, 'Savings');

INSERT INTO transactions VALUES
('T001', 'A101', 5000, 'Deposit'),
('T002', 'A102', 10000, 'Withdrawal'),
('T003', 'A103', 3000, 'Deposit');
```

1. Insert 3 accounts and 3 transactions:

Same as above

2. Retrieve holder_name and balance of all accounts:

```
SELECT holder_name, balance FROM accounts;
```

3. List all transactions sorted by txn_type:

```
SELECT * FROM transactions ORDER BY txn_type;
```

4. Change the account_type of account A103 to "Current":

```
UPDATE accounts SET account_type = 'Current' WHERE account_no = 'A103';
```

5. Delete account A104:

```
DELETE FROM accounts WHERE account_no = 'A104';
```

SLIP NO. 6

Q.1) Social Media Platform in Neo4j (10 Marks)

Data Creation:

```
// Create Users
CREATE (u1:User {name: 'Rahul', username: 'rahul123'})
CREATE (u2:User {name: 'Priya', username: 'priya456'})
CREATE (u3:User {name: 'Amit', username: 'amit789'})

// Create Posts
CREATE (p1:Post {post_id: 'P123', content: 'Hello World!', date: '2024-01-15'})
CREATE (p2:Post {post_id: 'P456', content: 'Learning Neo4j', date: '2024-01-16'})
CREATE (p3:Post {post_id: 'P789', content: 'MongoDB is cool', date: '2024-01-17'})

// Create Comments
CREATE (c1:Comment {comment_id: 'C001', text: 'Nice post!'})
CREATE (c2:Comment {comment_id: 'C002', text: 'Great content!'})
CREATE (c3:Comment {comment_id: 'C003', text: 'Very informative'})

// Create Groups
CREATE (g1:Group {name: 'TechClub', members_count: 50})
CREATE (g2:Group {name: 'DataScience', members_count: 75})

// Relationships
CREATE (u1)-[:POSTED]->(p1)
CREATE (u1)-[:POSTED]->(p2)
CREATE (u2)-[:POSTED]->(p3)
```

```
CREATE (u2)-[:COMMENTED_ON]->(c1)
CREATE (u3)-[:COMMENTED_ON]->(c2)

CREATE (p1)-[:HAS_COMMENT]->(c1)
CREATE (p1)-[:HAS_COMMENT]->(c2)
CREATE (p2)-[:HAS_COMMENT]->(c3)

CREATE (u1)-[:MEMBER_OF]->(g1)
CREATE (u2)-[:MEMBER_OF]->(g1)
CREATE (u3)-[:MEMBER_OF]->(g2)
```

a. List all posts made by user "Rahul".

```
MATCH (u:User {name: 'Rahul'})-[:POSTED]->(p:Post)
RETURN p.post_id AS PostID, p.content AS Content
```

b. Find users who commented on post "P123".

```
MATCH (p:Post {post_id: 'P123'})-[:HAS_COMMENT]->(c:Comment)<-[:COMMENTED_ON]-(u:User)
RETURN DISTINCT u.name AS UserName
```

c. Count the number of members in each group.

```
MATCH (u:User)-[:MEMBER_OF]->(g:Group)
RETURN g.name AS GroupName, COUNT(u) AS MemberCount
```

d. Delete all comments on post "P456".

```
MATCH (p:Post {post_id: 'P456'})-[r:HAS_COMMENT]->(c:Comment)
DELETE r, c
```

e. Update group name "TechClub" to "AIClub".

```
MATCH (g:Group {name: 'TechClub'})
SET g.name = 'AIClub'
RETURN g
```

Q.2) CompanyDB in MongoDB (10 Marks)

Data Creation:

```
use CompanyDB

db.employees.insertMany([
    {name: "Alice", position: "Developer", department: "IT", salary: 60000},
    {name: "Bob", position: "Developer", department: "IT", salary: 55000},
    {name: "Charlie", position: "Manager", department: "HR", salary: 75000},
    {name: "Diana", position: "Analyst", department: "Finance", salary: 50000},
    {name: "Eve", position: "Designer", department: "Marketing", salary: 52000}
])
```

1) Insert 5 documents into the employees collection:

Same as above

2) Retrieve all documents in the collection:

```
db.employees.find()
```

3) Find employees whose salary is greater than 55000:

```
db.employees.find({salary: {$gt: 55000}})
```

4) Update the position of Bob to Senior Developer:

```
db.employees.updateOne({name: "Bob"}, {$set: {position: "Senior Developer"}})
```

Q.3) University Courses in SQL (10 Marks)

Data Creation:

```
CREATE TABLE students (
    roll_no VARCHAR(10) PRIMARY KEY,
    name VARCHAR(50),
    major VARCHAR(50),
    year INT
);

CREATE TABLE subjects (
    subject_code VARCHAR(10) PRIMARY KEY,
    subject_name VARCHAR(50),
    credits INT
);

INSERT INTO students VALUES
```

```
('S101', 'Amit', 'Physics', 2),
('S102', 'Priya', 'Chemistry', 3),
('S103', 'Rahul', 'Mathematics', 1),
('S104', 'Neha', 'Biology', 2);

INSERT INTO subjects VALUES
('SUB101', 'Calculus', 4),
('SUB102', 'Quantum Physics', 3),
('SUB103', 'Organic Chemistry', 4);
```

1. Insert 3 records students and subjects:

Same as above

2. Retrieve name, major, and year of all students:

```
SELECT name, major, year FROM students;
```

3. List all subjects sorted by subject_name:

```
SELECT * FROM subjects ORDER BY subject_name;
```

4. Change the major of student S103 to "Computer Science":

```
UPDATE students SET major = 'Computer Science' WHERE roll_no = 'S103';
```

5. Delete student S104:

```
DELETE FROM students WHERE roll_no = 'S104';
```

SLIP NO. 7

Q.1) Banking System in Neo4j (10 Marks)

Data Creation:

```
// Create Customers
CREATE (c1:Customer {name: 'Sunita', cust_id: 'C001'})
CREATE (c2:Customer {name: 'Rajesh', cust_id: 'C002'})
CREATE (c3:Customer {name: 'Meera', cust_id: 'C003'})

// Create Accounts
CREATE (a1:Account {acc_no: 'A101', balance: 50000, type: 'Savings'})
CREATE (a2:Account {acc_no: 'A202', balance: 75000, type: 'Current'})
```

```
CREATE (a3:Account {acc_no: 'A303', balance: 30000, type: 'Savings'})

// Create Transactions
CREATE (t1:Transaction {txn_id: 'T001', amount: 5000, type: 'Deposit'})
CREATE (t2:Transaction {txn_id: 'T002', amount: 10000, type: 'Withdrawal'})
CREATE (t3:Transaction {txn_id: 'T003', amount: 3000, type: 'Transfer'})

// Create Branches
CREATE (b1:Branch {name: 'Mumbai Branch', code: 'MUM001'})
CREATE (b2:Branch {name: 'Delhi Branch', code: 'DEL001'})
CREATE (b3:Branch {name: 'Pune Branch', code: 'PUN001'})

// Relationships
CREATE (c1)-[:OWNS]->(a1)
CREATE (c1)-[:OWNS]->(a3)
CREATE (c2)-[:OWNS]->(a2)

CREATE (a1)-[:TRANSACTS]->(t1)
CREATE (a1)-[:TRANSACTS]->(t2)
CREATE (a2)-[:TRANSACTS]->(t3)

CREATE (a1)-[:LOCATED_IN]->(b1)
CREATE (a2)-[:LOCATED_IN]->(b2)
CREATE (a3)-[:LOCATED_IN]->(b1)
```

a. List all accounts owned by "Sunita".

```
MATCH (c:Customer {name: 'Sunita'})-[:OWNS]->(a:Account)
RETURN a.acc_no AS AccountNumber, a.balance AS Balance
```

b. Find all transactions made from account "A101".

```
MATCH (a:Account {acc_no: 'A101'})-[:TRANSACTS]->(t:Transaction)
RETURN t.txn_id AS TransactionID, t.amount AS Amount, t.type AS Type
```

c. Count the number of accounts in each branch.

```
MATCH (a:Account)-[:LOCATED_IN]->(b:Branch)
RETURN b.name AS Branch, COUNT(a) AS AccountCount
```

d. Delete all transactions of account "A202".

```
MATCH (a:Account {acc_no: 'A202'})-[r:TRANSACTS]->(t:Transaction)
DELETE r, t
```

e. Update branch of account "A303" to "Delhi Branch".

```
MATCH (a:Account {acc_no: 'A303'})-[r:LOCATED_IN]->(b:Branch)
DELETE r
WITH a
MATCH (newBranch:Branch {name: 'Delhi Branch'})
CREATE (a)-[:LOCATED_IN]->(newBranch)
```

Q.2) SchoolDB in MongoDB (10 Marks)

Data Creation:

```
use SchoolDB

db.students.insertMany([
    {name: "Amit", class: "10th", marks: 85, section: "A"},
    {name: "Priya", class: "10th", marks: 92, section: "B"},
    {name: "Rahul", class: "9th", marks: 78, section: "A"},
    {name: "Ankit", class: "10th", marks: 75, section: "A"},
    {name: "Neha", class: "11th", marks: 88, section: "B"}
])
```

1) Insert 5 documents into the students collection:

Same as above

2) Retrieve all students in the 10th class:

```
db.students.find({class: "10th"})
```

3) Find students with marks greater than 80:

```
db.students.find({marks: {$gt: 80}})
```

4) Update the marks of Ankit to 82:

```
db.students.updateOne({name: "Ankit"}, {$set: {marks: 82}})
```

Q.3) Employee-Department Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Department (
    dno INT PRIMARY KEY,
    dname VARCHAR(50),
    dloc VARCHAR(50)
);

CREATE TABLE Employee (
    eno INT PRIMARY KEY,
    ename VARCHAR(50),
    designation VARCHAR(50),
    sal DECIMAL(10,2),
    dno INT,
    FOREIGN KEY (dno) REFERENCES Department(dno)
);

INSERT INTO Department VALUES
(1, 'IT', 'Pune'),
(2, 'HR', 'Mumbai'),
(3, 'Finance', 'Delhi');

INSERT INTO Employee VALUES
(101, 'Suresh', 'Developer', 55000, 1),
(102, 'Ramesh', 'Manager', 75000, 2),
(103, 'Sita', 'Analyst', 45000, 3),
(104, 'Geeta', 'Developer', 52000, 1),
(105, 'Mohan', 'Accountant', 48000, 3);
```

a) List the name of employees whose salary is above 50000:

```
SELECT ename FROM Employee WHERE sal > 50000;
```

b) Count the number of employees in each department:

```
SELECT D.dname, COUNT(E.eno) AS EmployeeCount
FROM Department D
LEFT JOIN Employee E ON D.dno = E.dno
GROUP BY D.dno, D.dname;
```

c) Update all employees' salary increase by 25%:

```
UPDATE Employee SET sal = sal * 1.25;
```

d) Find the employee details whose name starts with 'S':

```
SELECT * FROM Employee WHERE ename LIKE 'S%';
```

SLIP NO. 8

Q.1) Airline Reservation System in Neo4j (10 Marks)

Data Creation:

```
// Create Passengers
CREATE (p1:Passenger {name: 'Rohit', id: 'P001'})
CREATE (p2:Passenger {name: 'Neha', id: 'P002'})
CREATE (p3:Passenger {name: 'Amit', id: 'P003'})

// Create Tickets
CREATE (t1:Ticket {ticket_no: 'TK001', seat: '12A', class: 'Business'})
CREATE (t2:Ticket {ticket_no: 'TK002', seat: '24B', class: 'Economy'})
CREATE (t3:Ticket {ticket_no: 'TK003', seat: '8C', class: 'Business'})

// Create Flights
CREATE (f1:Flight {flight_no: 'F123', airline: 'Air India', time: '10:00'})
CREATE (f2:Flight {flight_no: 'F456', airline: 'IndiGo', time: '14:00'})
CREATE (f3:Flight {flight_no: 'F789', airline: 'SpiceJet', time: '18:00'})

// Create Airports
CREATE (a1:Airport {name: 'Delhi Airport', code: 'DEL'})
CREATE (a2:Airport {name: 'Mumbai Airport', code: 'BOM'})
CREATE (a3:Airport {name: 'Bangalore Airport', code: 'BLR'})

// Relationships
CREATE (p1)-[:BOOKED]->(t1)
CREATE (p1)-[:BOOKED]->(t2)
CREATE (p2)-[:BOOKED]->(t3)

CREATE (t1)-[:FOR_FLIGHT]->(f1)
CREATE (t2)-[:FOR_FLIGHT]->(f2)
CREATE (t3)-[:FOR_FLIGHT]->(f1)

CREATE (f1)-[:DEPARTS_FROM]->(a1)
CREATE (f2)-[:DEPARTS_FROM]->(a1)
CREATE (f3)-[:DEPARTS_FROM]->(a2)

CREATE (f1)-[:ARRIVES_AT]->(a2)
CREATE (f2)-[:ARRIVES_AT]->(a3)
CREATE (f3)-[:ARRIVES_AT]->(a1)
```

a. List all tickets booked by passenger "Rohit".

```
MATCH (p:Passenger {name: 'Rohit'})-[:BOOKED]->(t:Ticket)
RETURN t.ticket_no AS TicketNo, t.seat AS Seat, t.class AS Class
```

b. Find all flights departing from "Delhi Airport".

```
MATCH (f:Flight)-[:DEPARTS_FROM]->(a:Airport {name: 'Delhi Airport'})
RETURN f.flight_no AS FlightNo, f.airline AS Airline
```

c. Count the number of passengers on each flight.

```
MATCH (p:Passenger)-[:BOOKED]->(t:Ticket)-[:FOR_FLIGHT]->(f:Flight)
RETURN f.flight_no AS Flight, COUNT(DISTINCT p) AS PassengerCount
```

d. Delete all tickets of passenger "Neha".

```
MATCH (p:Passenger {name: 'Neha'})-[r:BOOKED]->(t:Ticket)
DETACH DELETE t
```

e. Update destination airport of flight "F123" to "Mumbai".

```
MATCH (f:Flight {flight_no: 'F123'})-[r:ARRIVES_AT]->(a:Airport)
DELETE r
WITH f
MATCH (newAirport:Airport {name: 'Mumbai Airport'})
CREATE (f)-[:ARRIVES_AT]->(newAirport)
```

Q.2) ShopDB in MongoDB (10 Marks)

Data Creation:

```
use ShopDB

db.products.insertMany([
    {name: "Laptop", category: "Electronics", price: 75000, stock: 50},
    {name: "Mobile", category: "Electronics", price: 25000, stock: 100},
    {name: "Tablet", category: "Electronics", price: 35000, stock: 75},
    {name: "Headphones", category: "Accessories", price: 5000, stock: 200},
    {name: "Keyboard", category: "Accessories", price: 2000, stock: 150}
])
```

1) Insert 5 documents into the products collection:

Same as above

2) Retrieve all products priced above 10,000:

```
db.products.find({price: {$gt: 10000}})
```

3) Find the product with the name "Laptop":

```
db.products.find({name: "Laptop"})
```

4) Delete the product "Mobile":

```
db.products.deleteOne({name: "Mobile"})
```

Q.3) Project-Employee Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Project (
    P_No INT PRIMARY KEY,
    P_Name VARCHAR(50),
    P_Type VARCHAR(30),
    Duration INT
);

CREATE TABLE Employee (
    E_no INT PRIMARY KEY,
    E_Name VARCHAR(50),
    Qualification VARCHAR(30),
    JoinDate DATE
);

CREATE TABLE Project_Employee (
    P_No INT,
    E_no INT,
    PRIMARY KEY (P_No, E_no),
    FOREIGN KEY (P_No) REFERENCES Project(P_No),
    FOREIGN KEY (E_no) REFERENCES Employee(E_no)
);

INSERT INTO Project VALUES
(1, 'Robotics', 'Research', 12),
(2, 'AI Development', 'Development', 18),
(3, 'Web Portal', 'Development', 6);

INSERT INTO Employee VALUES
```

```
(101, 'Nitin', 'M.Tech', '2020-01-15'),
(102, 'Suresh', 'B.E.', '2019-06-20'),
(103, 'Priya', 'MCA', '2021-03-10'),
(104, 'Neeraj', 'B.E.', '2018-09-05');

INSERT INTO Project_Employee VALUES
(1, 101), (1, 102), (2, 103), (2, 104), (3, 101);
```

a) Find the employee details whose name starts with 'N':

```
SELECT * FROM Employee WHERE E_Name LIKE 'N%';
```

b) List the names of the employees in alphabetical order:

```
SELECT E_Name FROM Employee ORDER BY E_Name ASC;
```

c) Delete the information of employee whose qualification is B.E.:

```
DELETE FROM Project_Employee WHERE E_no IN (SELECT E_no FROM Employee WHERE Qualificatior
DELETE FROM Employee WHERE Qualification = 'B.E.';
```

d) Find the employee numbers of the employees, who do not work on project "Robotics":

```
SELECT E_no FROM Employee
WHERE E_no NOT IN (
    SELECT PE.E_no FROM Project_Employee PE
    JOIN Project P ON PE.P_No = P.P_No
    WHERE P.P_Name = 'Robotics'
);
```

SLIP NO. 9

Q.1) Music Streaming Service in Neo4j (10 Marks)

Data Creation:

```
// Create Users
CREATE (u1:User {name: 'Anjali', user_id: 'U001'})
CREATE (u2:User {name: 'Rohit', user_id: 'U002'})
CREATE (u3:User {name: 'Priya', user_id: 'U003'})

// Create Songs
CREATE (s1:Song {name: 'Tum Hi Ho', duration: '4:22'})
```

```
CREATE (s2:Song {name: 'SongX', duration: '3:45'})
CREATE (s3:Song {name: 'Kesariya', duration: '4:28'})
CREATE (s4:Song {name: 'Raabta', duration: '5:01'})

// Create Artists
CREATE (ar1:Artist {name: 'Arijit Singh'})
CREATE (ar2:Artist {name: 'Shreya Ghoshal'})
CREATE (ar3:Artist {name: 'Pritam'})

// Create Playlists
CREATE (pl1:Playlist {name: 'BollywoodHits', songs_count: 25})
CREATE (pl2:Playlist {name: 'Romantic Songs', songs_count: 30})

// Relationships
CREATE (u1)-[:LISTENS_TO]->(s1)
CREATE (u2)-[:LISTENS_TO]->(s2)

CREATE (s1)-[:SUNG_BY]->(ar1)
CREATE (s2)-[:SUNG_BY]->(ar1)
CREATE (s3)-[:SUNG_BY]->(ar1)
CREATE (s4)-[:SUNG_BY]->(ar2)

CREATE (s1)-[:ADDED_TO]->(pl1)
CREATE (s2)-[:ADDED_TO]->(pl1)
CREATE (s3)-[:ADDED_TO]->(pl2)

CREATE (pl1)-[:CREATED_BY]->(u1)
CREATE (pl2)-[:CREATED_BY]->(u2)
```

a. List all songs sung by "Arijit Singh".

```
MATCH (s:Song)-[:SUNG_BY]->(ar:Artist {name: 'Arijit Singh'})
RETURN s.name AS SongName
```

b. Find playlists created by user "Anjali".

```
MATCH (pl:Playlist)-[:CREATED_BY]->(u:User {name: 'Anjali'})
RETURN pl.name AS PlaylistName
```

c. Count the number of songs in each playlist.

```
MATCH (s:Song)-[:ADDED_TO]->(pl:Playlist)
RETURN pl.name AS Playlist, COUNT(s) AS SongCount
```

d. Delete all songs in playlist "BollywoodHits".

```
MATCH (s:Song)-[r:ADDED_TO]->(pl:Playlist {name: 'BollywoodHits'})
DELETE r
```

e. Update artist of song "SongX" to "Shreya Ghoshal".

```
MATCH (s:Song {name: 'SongX'})-[r:SUNG_BY]->(ar:Artist)
DELETE r
WITH s
MATCH (newArtist:Artist {name: 'Shreya Ghoshal'})
CREATE (s)-[:SUNG_BY]->(newArtist)
```

Q.2) LibraryDB in MongoDB (10 Marks)

Data Creation:

```
use LibraryDB

db.books.insertMany([
    {title: "The Great Gatsby", author: "F. Scott Fitzgerald", price: 350},
    {title: "To Kill a Mockingbird", author: "Harper Lee", price: 400},
    {title: "1984", author: "George Orwell", price: 300},
    {title: "Pride and Prejudice", author: "Jane Austen", price: 380},
    {title: "Brave New World", author: "Aldous Huxley", price: 360}
])
```

1) Insert 5 documents into the books collection:

Same as above

2) Retrieve all books from the collection:

```
db.books.find()
```

3) Find books that cost more than 350:

```
db.books.find({price: {$gt: 350}})
```

4) Update the price of "1984" to 320:

```
db.books.updateOne({title: "1984"}, {$set: {price: 320}})
```

Q.3) Student-Competition Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Student (
    sreg_no INT PRIMARY KEY,
    s_name VARCHAR(50),
    class VARCHAR(20) CHECK (class IN ('FYBSc_IT', 'SYBSc_IT', 'TYBSc_IT'))
);

CREATE TABLE Competition (
    c_no INT PRIMARY KEY,
    c_name VARCHAR(50),
    c_type VARCHAR(30)
);

CREATE TABLE Student_Competition (
    sreg_no INT,
    c_no INT,
    rank INT,
    year INT,
    PRIMARY KEY (sreg_no, c_no, year),
    FOREIGN KEY (sreg_no) REFERENCES Student(sreg_no),
    FOREIGN KEY (c_no) REFERENCES Competition(c_no)
);

INSERT INTO Student VALUES
(1, 'Amit', 'FYBSc_IT'),
(2, 'Priya', 'SYBSc_IT'),
(3, 'Rahul', 'TYBSc_IT'),
(4, 'Neha', 'FYBSc_IT'),
(5, 'Arun', 'SYBSc_IT');

INSERT INTO Competition VALUES
(101, 'Quiz', 'Academic'),
(102, 'Debate', 'Literary'),
(103, 'Coding', 'Technical'),
(104, 'Sports', 'Athletics');

INSERT INTO Student_Competition VALUES
(1, 101, 1, 2023), (2, 102, 1, 2020), (3, 103, 2, 2020),
(1, 102, 3, 2020), (4, 101, 2, 2023), (2, 103, 1, 2020);
```

a) List the names of students scoring 1st rank in all different competition:

```
SELECT DISTINCT S.s_name
FROM Student S
JOIN Student_Competition SC ON S.sreg_no = SC.sreg_no
WHERE SC.rank = 1;
```

b) Delete all students of class FYBSc_IT participated in Quiz competition in year 2023:

```
DELETE FROM Student_Competition
WHERE sreg_no IN (SELECT sreg_no FROM Student WHERE class = 'FYBSc_IT')
AND c_no = (SELECT c_no FROM Competition WHERE c_name = 'Quiz')
AND year = 2023;
```

c) List the names of students participated in more than one competitions:

```
SELECT S.s_name
FROM Student S
JOIN Student_Competition SC ON S.sreg_no = SC.sreg_no
GROUP BY S.sreg_no, S.s_name
HAVING COUNT(DISTINCT SC.c_no) > 1;
```

d) List the name of all competitions held in year 2020:

```
SELECT DISTINCT C.c_name
FROM Competition C
JOIN Student_Competition SC ON C.c_no = SC.c_no
WHERE SC.year = 2020;
```

SLIP NO. 10

Q.1) E-learning Platform in Neo4j (10 Marks)

Data Creation:

```
// Create Students
CREATE (s1:Student {name: 'Aman', id: 'STU001'})
CREATE (s2:Student {name: 'Kavita', id: 'STU002'})
CREATE (s3:Student {name: 'Rahul', id: 'STU003'})

// Create Courses
CREATE (c1:Course {code: 'AI Basics', name: 'Introduction to AI'})
CREATE (c2:Course {code: 'ML101', name: 'Machine Learning'})
CREATE (c3:Course {code: 'DS101', name: 'Data Science'})
```

```
// Create Instructors
CREATE (i1:Instructor {name: 'Prof. Sharma'})
CREATE (i2:Instructor {name: 'Prof. Sen'})
CREATE (i3:Instructor {name: 'Prof. Gupta'})

// Create Assignments
CREATE (a1:Assignment {name: 'Assignment 1', deadline: '2024-02-15'})
CREATE (a2:Assignment {name: 'Assignment 2', deadline: '2024-02-20'})
CREATE (a3:Assignment {name: 'Project', deadline: '2024-03-01'})

// Relationships
CREATE (s1)-[:ENROLLED]->(c1)
CREATE (s1)-[:ENROLLED]->(c2)
CREATE (s2)-[:ENROLLED]->(c1)
CREATE (s3)-[:ENROLLED]->(c3)

CREATE (c1)-[:TAUGHT_BY]->(i1)
CREATE (c2)-[:TAUGHT_BY]->(i2)
CREATE (c3)-[:TAUGHT_BY]->(i3)

CREATE (s1)-[:SUBMITTED]->(a1)
CREATE (s2)-[:SUBMITTED]->(a1)
CREATE (s2)-[:SUBMITTED]->(a2)

CREATE (a1)-[:ASSIGNED_IN]->(c1)
CREATE (a2)-[:ASSIGNED_IN]->(c1)
CREATE (a3)-[:ASSIGNED_IN]->(c2)
```

a. List all courses taken by "Aman".

```
MATCH (s:Student {name: 'Aman'})-[:ENROLLED]->(c:Course)
RETURN c.name AS CourseName
```

b. Find all assignments submitted in course "AI Basics".

```
MATCH (a:Assignment)-[:ASSIGNED_IN]->(c:Course {code: 'AI Basics'})
RETURN a.name AS AssignmentName
```

c. Count the number of students enrolled in each course.

```
MATCH (s:Student)-[:ENROLLED]->(c:Course)
RETURN c.name AS Course, COUNT(s) AS StudentCount
```

d. Delete all assignments submitted by student "Kavita".

```
MATCH (s:Student {name: 'Kavita'})-[r:SUBMITTED]->(a:Assignment)
DELETE r
```

e. Update instructor of course "ML101" to "Prof. Sen".

```
MATCH (c:Course {code: 'ML101'})-[r:TAUGHT_BY]->(i:Instructor)
DELETE r
WITH c
MATCH (newInstructor:Instructor {name: 'Prof. Sen'})
CREATE (c)-[:TAUGHT_BY]->(newInstructor)
```

Q.2) LibraryDB in MongoDB (10 Marks)

Same as Slip 9 Q.2

Q.3) Person-Area Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Area (
    aname VARCHAR(50) PRIMARY KEY,
    area_type VARCHAR(10) CHECK (area_type IN ('urban', 'rural'))
);

CREATE TABLE Person (
    pnumber INT PRIMARY KEY,
    pname VARCHAR(50),
    birthdate DATE,
    income DECIMAL(10,2),
    aname VARCHAR(50),
    FOREIGN KEY (aname) REFERENCES Area(aname)
);

INSERT INTO Area VALUES
('Pune', 'urban'),
('Mumbai', 'urban'),
('Solapur', 'rural'),
('Nashik', 'rural'),
('Nagpur', 'urban');

INSERT INTO Person VALUES
(1, 'Ayush', '1990-05-15', 50000, 'Pune'),
(2, 'Ayansh', '1985-08-20', 75000, 'Mumbai'),
(3, 'Ramesh', '1992-03-10', 25000, 'Solapur'),
(4, 'Ayesha', '1988-11-25', 60000, 'Nashik'),
(5, 'Suresh', '1995-07-30', 95000, 'Nagpur');
```

a) List the names of all people whose name starts with 'Ay':

```
SELECT pname FROM Person WHERE pname LIKE 'Ay%';
```

b) Delete information of all people staying in 'rural' area:

```
DELETE FROM Person WHERE aname IN (SELECT aname FROM Area WHERE area_type = 'rural');
```

c) Find the count of people area wise:

```
SELECT aname, COUNT(*) AS PersonCount FROM Person GROUP BY aname;
```

d) List the names of all people whose income is between Rs.30,000 and Rs.90,000:

```
SELECT pname FROM Person WHERE income BETWEEN 30000 AND 90000;
```

SLIP NO. 11

Q.1) Supply Chain Management in Neo4j (10 Marks)

Data Creation:

```
// Create Suppliers
CREATE (sup1:Supplier {name: 'Supplier A', location: 'Delhi'})
CREATE (sup2:Supplier {name: 'Supplier B', location: 'Mumbai'})
CREATE (sup3:Supplier {name: 'Supplier C', location: 'Pune'})

// Create Products
CREATE (p1:Product {name: 'Laptop', sku: 'LAP001', price: 75000})
CREATE (p2:Product {name: 'Mobile', sku: 'MOB001', price: 25000})
CREATE (p3:Product {name: 'Tablet', sku: 'TAB001', price: 35000})

// Create Warehouses
CREATE (w1:Warehouse {name: 'Warehouse Delhi', capacity: 1000})
CREATE (w2:Warehouse {name: 'Warehouse Mumbai', capacity: 1500})
CREATE (w3:Warehouse {name: 'Warehouse Pune', capacity: 800})

// Create Orders
CREATE (o1:Order {order_id: 'O456', date: '2024-01-15', value: 100000})
CREATE (o2:Order {order_id: 'O789', date: '2024-01-16', value: 50000})

// Relationships
CREATE (sup1)-[:SUPPLIES]->(p1)
CREATE (sup1)-[:SUPPLIES]->(p2)
```

```
CREATE (sup2)-[:SUPPLIES]->(p3)

CREATE (p1)-[:STORED_IN]->(w1)
CREATE (p1)-[:STORED_IN]->(w2)
CREATE (p2)-[:STORED_IN]->(w1)
CREATE (p3)-[:STORED_IN]->(w3)

CREATE (o1)-[:CONTAINS]->(p1)
CREATE (o1)-[:CONTAINS]->(p2)
CREATE (o2)-[:CONTAINS]->(p3)
```

a. List all products supplied by "Supplier A".

```
MATCH (sup:Supplier {name: 'Supplier A'})-[:SUPPLIES]->(p:Product)
RETURN p.name AS ProductName
```

b. Find warehouses storing "Laptop".

```
MATCH (p:Product {name: 'Laptop'})-[:STORED_IN]->(w:Warehouse)
RETURN w.name AS WarehouseName
```

c. Count the number of products in each warehouse.

```
MATCH (p:Product)-[:STORED_IN]->(w:Warehouse)
RETURN w.name AS Warehouse, COUNT(p) AS ProductCount
```

d. Delete all products in order "O456".

```
MATCH (o:Order {order_id: 'O456'})-[r:CONTAINS]->(p:Product)
DELETE r
```

e. Update supplier of product "Mobile" to "Supplier B".

```
MATCH (p:Product {name: 'Mobile'})<-[r:SUPPLIES]-(sup:Supplier)
DELETE r
WITH p
MATCH (newSup:Supplier {name: 'Supplier B'})
CREATE (newSup)-[:SUPPLIES]->(p)
```

Q.2) StockDB in MongoDB (10 Marks)

Data Creation:

```
use StockDB

db.Product.insertMany([
    {name: "Laptop", category: "Electronics", price: 75000, stock: 50},
    {name: "Notebook", category: "Stationery", price: 50, stock: 500},
    {name: "Pen", category: "Stationery", price: 20, stock: 1000},
    {name: "Mobile", category: "Electronics", price: 25000, stock: 100},
    {name: "Tablet", category: "Electronics", price: 3500, stock: 15}
])
```

1. Find products with price greater than 1,000:

```
db.Product.find({price: {$gt: 1000}})
```

2. Show products not in the "Stationery" category:

```
db.Product.find({category: {$ne: "Stationery"}})
```

3. List products with stock less than 20:

```
db.Product.find({stock: {$lt: 20}})
```

4. Find products priced between 100 and 5000:

```
db.Product.find({price: {$gte: 100, $lte: 5000}})
```

Q.3) Project-Employee Database in SQL (10 Marks)

Same as Slip 8 Q.3

SLIP NO. 12

Q.1) Plan-Customer Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Plan (
    plan_no INT PRIMARY KEY,
    plan_name VARCHAR(50),
    fix_amt DECIMAL(10,2) CHECK (fix_amt > 0)
);

CREATE TABLE Customer (
    cust_no INT PRIMARY KEY,
    cust_name VARCHAR(50),
    mobile_no VARCHAR(15),
    plan_no INT,
    FOREIGN KEY (plan_no) REFERENCES Plan(plan_no)
);

INSERT INTO Plan VALUES
(1, 'Basic', 199),
(2, 'Silver', 399),
(3, 'Gold', 599),
(4, 'Platinum', 999);

INSERT INTO Customer VALUES
(101, 'Amit', '9876543210', 1),
(102, 'Priya', '9876543211', 2),
(103, 'Rahul', '9876543212', 1),
(104, 'Neha', '9876543213', 3),
(105, 'Arun', '9876543214', 4);
```

1. Find all plans with fix_amt greater than 300:

```
SELECT * FROM Plan WHERE fix_amt > 300;
```

2. Change a customer's plan from 'Basic' to 'Silver':

```
UPDATE Customer
SET plan_no = (SELECT plan_no FROM Plan WHERE plan_name = 'Silver')
WHERE plan_no = (SELECT plan_no FROM Plan WHERE plan_name = 'Basic');
```

3. Delete a customer record by name:

```
DELETE FROM Customer WHERE cust_name = 'Amit';
```

Q.2) Sales System in MongoDB (10 Marks)

Same as Slip 4 Q.2

Q.3) Person-Movie Database in Neo4j (10 Marks)

Data Creation:

```
// Create Persons
CREATE (p1:Person {name: 'Alice'})
CREATE (p2:Person {name: 'Bob'})
CREATE (p3:Person {name: 'Charlie'})

// Create Movies
CREATE (m1:Movie {title: 'Inception', year: 2010})
CREATE (m2:Movie {title: 'The Matrix', year: 1999})
CREATE (m3:Movie {title: 'Interstellar', year: 2014})

// Create LIKES relationships
CREATE (p1)-[:LIKES]->(m1)
CREATE (p1)-[:LIKES]->(m3)
CREATE (p2)-[:LIKES]->(m1)
CREATE (p2)-[:LIKES]->(m2)
CREATE (p3)-[:LIKES]->(m2)
```

1) Retrieve all Person nodes with the name starting with "A":

```
MATCH (p:Person)
WHERE p.name STARTS WITH 'A'
RETURN p
```

2) Retrieve and display all nodes in the database:

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

3) Find all movies liked by Bob:

```
MATCH (p:Person {name: 'Bob'})-[:LIKES]->(m:Movie)
RETURN m.title AS MovieTitle
```

SLIP NO. 13

Q.1) Project-Department Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Department (
    dno INT PRIMARY KEY,
    dname VARCHAR(50),
    HOD VARCHAR(50),
    loc VARCHAR(50)
);

CREATE TABLE Project (
    pno INT PRIMARY KEY,
    pname VARCHAR(50),
    start_date DATE,
    budget DECIMAL(15,2),
    status CHAR(1) CHECK (status IN ('C', 'P', 'I')),
    dno INT,
    FOREIGN KEY (dno) REFERENCES Department(dno)
);

INSERT INTO Department VALUES
(1, 'IT', 'Dr. Sharma', 'Pune'),
(2, 'HR', 'Dr. Mehta', 'Mumbai'),
(3, 'Finance', 'Dr. Gupta', 'Delhi');

INSERT INTO Project VALUES
(101, 'Web Portal', '2023-01-15', 45000, 'P', 1),
(102, 'ERP System', '2022-06-01', 150000, 'C', 1),
(103, 'HR Automation', '2024-01-10', 30000, 'I', 2),
(104, 'Budget Tracker', '2023-08-20', 25000, 'P', 3),
(105, 'AI Research', '2023-03-15', 80000, 'P', 1);
```

1. Display total number of projects whose status is "P":

```
SELECT COUNT(*) AS TotalProjects FROM Project WHERE status = 'P';
```

2. Select the project with budget less than 50,000:

```
SELECT * FROM Project WHERE budget < 50000;
```

3. Delete the employees whose status is Incomplete:

```
DELETE FROM Project WHERE status = 'I';
```

Q.2) Olympic Information System in MongoDB (10 Marks)

Same as Slip 3 Q.2

Q.3) Supply Chain Management in Neo4j (10 Marks)

Same as Slip 11 Q.1 (Any 3 questions)

SLIP NO. 14

Q.1) Course-Student Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Course (
    course_id VARCHAR(10) PRIMARY KEY,
    course_name VARCHAR(50),
    fees DECIMAL(10,2) CHECK (fees > 0)
);

CREATE TABLE Student (
    roll_no VARCHAR(10) PRIMARY KEY,
    stud_name VARCHAR(50),
    course_id VARCHAR(10),
    FOREIGN KEY (course_id) REFERENCES Course(course_id)
);

INSERT INTO Course VALUES
('C001', 'BCA', 35000),
('C002', 'B.Com', 25000),
('C003', 'BBA', 45000),
('C004', 'MCA', 55000);

INSERT INTO Student VALUES
('S001', 'Amit', 'C001'),
('S002', 'Priya', 'C002'),
('S003', 'Rahul', 'C002'),
('S004', 'Neha', 'C003'),
('S005', 'Arun', 'C002'),
('S006', 'Kavita', 'C002');
```

1. Find courses with fees greater than 40,000:

```
SELECT * FROM Course WHERE fees > 40000;
```

2. Update all students of 'B.Com' course to 'BBA':

```
UPDATE Student
SET course_id = (SELECT course_id FROM Course WHERE course_name = 'BBA')
WHERE course_id = (SELECT course_id FROM Course WHERE course_name = 'B.Com');
```

3. Delete a student by name:

```
DELETE FROM Student WHERE stud_name = 'Amit';
```

4. Find course names HAVING more than 3 students:

```
SELECT C.course_name, COUNT(S.roll_no) AS StudentCount
FROM Course C
JOIN Student S ON C.course_id = S.course_id
GROUP BY C.course_id, C.course_name
HAVING COUNT(S.roll_no) > 3;
```

Q.2) BookDB in MongoDB (10 Marks)

Same as Slip 2 Q.2

Q.3) Music Streaming Service in Neo4j (10 Marks)

Data Creation:

```
// Create Users
CREATE (u1:User {name: 'Chetak', user_id: 'U001'})
CREATE (u2:User {name: 'Rohit', user_id: 'U002'})

// Create Songs
CREATE (s1:Song {name: 'Song1', duration: '4:22'})
CREATE (s2:Song {name: 'Song2', duration: '3:45'})

// Create Artists
CREATE (ar1:Artist {name: 'Pritam'})
CREATE (ar2:Artist {name: 'Arijit Singh'})

// Create Playlists
CREATE (pl1:Playlist {name: 'My Playlist 1'})
CREATE (pl2:Playlist {name: 'My Playlist 2'})
```

```
// Relationships
CREATE (s1)-[:SUNG_BY]->(ar1)
CREATE (s2)-[:SUNG_BY]->(ar1)
CREATE (pl1)-[:CREATED_BY]->(u1)
CREATE (pl2)-[:CREATED_BY]->(u2)
```

a. List all songs sung by "Pritam":

```
MATCH (s:Song)-[:SUNG_BY]->(ar:Artist {name: 'Pritam'})
RETURN s.name AS SongName
```

b. Delete playlists created by user "Chetak":

```
MATCH (pl:Playlist)-[:CREATED_BY]->(u:User {name: 'Chetak'})
DETACH DELETE pl
```

c. Count the number of artists:

```
MATCH (ar:Artist)
RETURN COUNT(ar) AS ArtistCount
```

SLIP NO. 15

Q.1) Branch-Account Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Branch (
    branch_id VARCHAR(10) PRIMARY KEY,
    branch_name VARCHAR(50),
    city VARCHAR(50)
);

CREATE TABLE Account (
    acc_no VARCHAR(15) PRIMARY KEY,
    cust_name VARCHAR(50),
    balance DECIMAL(15,2) CHECK (balance >= 0),
    branch_id VARCHAR(10),
    FOREIGN KEY (branch_id) REFERENCES Branch(branch_id)
);

INSERT INTO Branch VALUES
('B001', 'Main Branch', 'Delhi'),
('B002', 'City Branch', 'Mumbai'),
('B003', 'Sub Branch', 'Pune');
```

```
INSERT INTO Account VALUES
('ACC001', 'Amit', 25000, 'B001'),
('ACC002', 'Priya', 45000, 'B001'),
('ACC003', 'Rahul', 15000, 'B002'),
('ACC004', 'Neha', 0, 'B002'),
('ACC005', 'Arun', 55000, 'B003');
```

1. Find all accounts where balance range 10,000 and 50,000:

```
SELECT * FROM Account WHERE balance BETWEEN 10000 AND 50000;
```

2. Update balance by adding ₹1,000 to customers in 'Delhi' branch:

```
UPDATE Account
SET balance = balance + 1000
WHERE branch_id IN (SELECT branch_id FROM Branch WHERE city = 'Delhi');
```

3. Delete accounts with balance = 0:

```
DELETE FROM Account WHERE balance = 0;
```

4. Display branch names HAVING average balance greater than 25,000:

```
SELECT B.branch_name, AVG(A.balance) AS AvgBalance
FROM Branch B
JOIN Account A ON B.branch_id = A.branch_id
GROUP BY B.branch_id, B.branch_name
HAVING AVG(A.balance) > 25000;
```

Q.2) StudentDB in MongoDB (10 Marks)

Same as Slip 1 Q.2

Q.3) Supply Chain Management in Neo4j (10 Marks)

Same as Slip 11 Q.1 (a, b, c only)

SLIP NO. 16

Q.1) Student-Marks Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Student (
    stud_id VARCHAR(10) PRIMARY KEY,
    stud_name VARCHAR(50),
    class VARCHAR(20)
);

CREATE TABLE Marks (
    stud_id VARCHAR(10),
    subject VARCHAR(50),
    marks INT CHECK (marks BETWEEN 0 AND 100),
    PRIMARY KEY (stud_id, subject),
    FOREIGN KEY (stud_id) REFERENCES Student(stud_id)
);

INSERT INTO Student VALUES
('S001', 'Amit', '10th'),
('S002', 'Priya', '10th'),
('S003', 'Rahul', '11th'),
('S004', 'Neha', '10th');

INSERT INTO Marks VALUES
('S001', 'Maths', 85),
('S001', 'Science', 78),
('S002', 'Maths', 92),
('S002', 'Science', 88),
('S003', 'Maths', 75),
('S003', 'Science', 82),
('S004', 'Maths', 30),
('S004', 'Science', 25);
```

1. Display all students whose marks range 80 and 100 in 'Maths':

```
SELECT S.stud_id, S.stud_name, M.marks
FROM Student S
JOIN Marks M ON S.stud_id = M.stud_id
WHERE M.subject = 'Maths' AND M.marks BETWEEN 80 AND 100;
```

2. Update marks by +5 for 'Science':

```
UPDATE Marks SET marks = marks + 5 WHERE subject = 'Science';
```

3. Delete students who failed (marks < 35):

```
DELETE FROM Marks WHERE marks < 35;
```

4. Display subjects HAVING average marks > 70:

```
SELECT subject, AVG(marks) AS AvgMarks
FROM Marks
GROUP BY subject
HAVING AVG(marks) > 70;
```

Q.2) LibraryDB in MongoDB (10 Marks)

Same as Slip 9 Q.2

Q.3) Social Network Graph in Neo4j (10 Marks)

Data Creation:

```
CREATE (a:Person {name: 'Alice'})
CREATE (b:Person {name: 'Bob'})
CREATE (c:Person {name: 'Charlie'})
CREATE (e:Person {name: 'Eve'})

CREATE (a)-[:FRIENDS]->(b)
CREATE (a)-[:FRIENDS]->(c)
CREATE (b)-[:FRIENDS]->(a)
CREATE (c)-[:FRIENDS]->(a)
CREATE (a)-[:KNOWS]->(e)
```

1) Find all nodes in the graph:

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

2) Retrieve all friends of Alice:

```
MATCH (a:Person {name: 'Alice'})-[:FRIENDS]->(f:Person)
RETURN f.name AS FriendName
```

SLIP NO. 17

Q.1) Flight-Passenger Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Flight (
    flight_no VARCHAR(10) PRIMARY KEY,
    source VARCHAR(50),
    destination VARCHAR(50),
    fare DECIMAL(10,2) CHECK (fare > 0)
);

CREATE TABLE Passenger (
    pid VARCHAR(10) PRIMARY KEY,
    pname VARCHAR(50),
    flight_no VARCHAR(10),
    seat_no VARCHAR(10),
    FOREIGN KEY (flight_no) REFERENCES Flight(flight_no)
);

INSERT INTO Flight VALUES
('F001', 'Mumbai', 'Delhi', 5500),
('F002', 'Delhi', 'Dubai', 15000),
('F003', 'Mumbai', 'Dubai', 12000),
('F004', 'Chennai', 'Singapore', 8000);

INSERT INTO Passenger VALUES
('P001', 'Amit', 'F001', '12A'),
('P002', 'Priya', 'F002', '8B'),
('P003', 'Rahul', 'F002', '10C'),
('P004', 'Neha', 'F002', '15A'),
('P005', 'Arun', 'F002', '20B'),
('P006', 'Kavita', 'F002', '22C'),
('P007', 'Sita', 'F002', '25A');
```

1. Find all flights where fare in range 5000 and 10000:

```
SELECT * FROM Flight WHERE fare BETWEEN 5000 AND 10000;
```

2. Update fare by 10% for destination = 'Dubai':

```
UPDATE Flight SET fare = fare * 1.10 WHERE destination = 'Dubai';
```

3. Delete passengers traveling to 'Canceled' flights:

```
DELETE FROM Passenger WHERE flight_no IN (SELECT flight_no FROM Flight WHERE destination
```

4. Display destination HAVING more than 5 passengers:

```
SELECT F.destination, COUNT(P.pid) AS PassengerCount
FROM Flight F
JOIN Passenger P ON F.flight_no = P.flight_no
GROUP BY F.destination
HAVING COUNT(P.pid) > 5;
```

Q.2) LibraryDB in MongoDB (10 Marks)

Same as Slip 9 Q.2

Q.3) Movie Database Graph in Neo4j (10 Marks)

Data Creation:

```
CREATE (tom:Actor {name: 'Tom Hanks'})
CREATE (robert:Director {name: 'Robert Zemeckis'})
CREATE (fg:Movie {title: 'Forrest Gump', year: 1994})
CREATE (ca:Movie {title: 'Cast Away', year: 2000})

CREATE (tom)-[:ACTED_IN]->(fg)
CREATE (tom)-[:ACTED_IN]->(ca)
CREATE (robert)-[:DIRECTED]->(fg)
```

1) Retrieve all movies directed by Robert Zemeckis:

```
MATCH (d:Director {name: 'Robert Zemeckis'})-[:DIRECTED]->(m:Movie)
RETURN m.title AS MovieTitle
```

2) Find all actors in the graph:

```
MATCH (a:Actor)
RETURN a.name AS ActorName
```

3) Delete Tom Hanks:

```
MATCH (a:Actor {name: 'Tom Hanks'})
DETACH DELETE a
```

SLIP NO. 18

Q.1) College-Teacher Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE College (
    code VARCHAR(10) PRIMARY KEY,
    college_name VARCHAR(100),
    address VARCHAR(200)
);

CREATE TABLE Teacher (
    teacher_id INT PRIMARY KEY,
    teacher_name VARCHAR(50),
    Qualification VARCHAR(30) NOT NULL,
    specialization VARCHAR(50),
    salary DECIMAL(10,2),
    Desg VARCHAR(30),
    college_code VARCHAR(10),
    FOREIGN KEY (college_code) REFERENCES College(code)
);

INSERT INTO College VALUES
('C001', 'Abhinav College', 'Pune'),
('C002', 'Modern College', 'Mumbai'),
('C003', 'National College', 'Delhi');

INSERT INTO Teacher VALUES
(101, 'Prof. Sharma', 'PhD', 'Computer Science', 75000, 'Professor', 'C001'),
(102, 'Prof. Mehta', 'M.Tech', 'Electronics', 65000, 'Asst. Professor', 'C002'),
(103, 'Dr. Kirti', 'PhD', 'Computer Science', 80000, 'HOD', 'C001'),
(104, 'Prof. Gupta', 'M.Sc', 'Mathematics', 55000, 'Lecturer', 'C003');
```

1. Display College details for Computer Science teacher:

```
SELECT C.*
FROM College C
JOIN Teacher T ON C.code = T.college_code
WHERE T.specialization = 'Computer Science';
```

2. Find the details of college "Abhinav" and "Modern":

```
SELECT * FROM College WHERE college_name IN ('Abhinav College', 'Modern College');
```

3. Change the name of teacher to Dr.Kirti whose Id=103:

```
UPDATE Teacher SET teacher_name = 'Dr.Kirti' WHERE teacher_id = 103;
```

Q.2) ShopDB in MongoDB (10 Marks)

Same as Slip 8 Q.2

Q.3) Company Organizational Hierarchy in Neo4j (10 Marks)

Data Creation:

```
CREATE (john:Employee {name: 'John'})
CREATE (mary:Employee {name: 'Mary'})
CREATE (steve:Employee {name: 'Steve'})
CREATE (sam:Employee {name: 'Sam'})

CREATE (john)-[:MANAGES]->(mary)
CREATE (john)-[:MANAGES]->(steve)
CREATE (mary)-[:MANAGES]->(sam)
```

1) Find all subordinates of John:

```
MATCH (john:Employee {name: 'John'})-[:MANAGES*]->(sub:Employee)
RETURN sub.name AS Subordinate
```

2) Retrieve all employees managed directly by Mary:

```
MATCH (mary:Employee {name: 'Mary'})-[:MANAGES]->(emp:Employee)
RETURN emp.name AS Employee
```

3) Delete the node Steve:

```
MATCH (steve:Employee {name: 'Steve'})
DETACH DELETE steve
```

SLIP NO. 19

Q.1) Driver-Car Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Driver (
    driver_id INT PRIMARY KEY,
    driver_name VARCHAR(50) NOT NULL,
    address VARCHAR(100)
);

CREATE TABLE Car (
    license_no VARCHAR(20) PRIMARY KEY,
    model VARCHAR(50),
    year INT
);

CREATE TABLE Driver_Car (
    driver_id INT,
    license_no VARCHAR(20),
    PRIMARY KEY (driver_id, license_no),
    FOREIGN KEY (driver_id) REFERENCES Driver(driver_id),
    FOREIGN KEY (license_no) REFERENCES Car(license_no)
);

INSERT INTO Driver VALUES
(100, 'KARAN', 'Pune'),
(101, 'Amit', 'Mumbai'),
(102, 'Priya', 'Delhi'),
(103, 'Rahul', 'Bangalore');

INSERT INTO Car VALUES
('MH12AB1234', 'Swift', 2006),
('MH14CD5678', 'Swift', 2010),
('DL01EF9012', 'i20', 2006),
('KA03GH3456', 'Creta', 2020);

INSERT INTO Driver_Car VALUES
(100, 'MH12AB1234'),
(101, 'MH14CD5678'),
(102, 'DL01EF9012'),
(103, 'MH12AB1234'),
(101, 'MH12AB1234');
```

1. Display the total number of person who are using "Swift" car:

```
SELECT COUNT(DISTINCT DC.driver_id) AS TotalUsers
FROM Driver_Car DC
```

```
JOIN Car C ON DC.license_no = C.license_no
WHERE C.model = 'Swift';
```

2. Change the driverID to 101 whose name is KARAN:

```
UPDATE Driver SET driver_id = 101 WHERE driver_name = 'KARAN';
```

3. Display all car launched in 2006:

```
SELECT * FROM Car WHERE year = 2006;
```

Q.2) SchoolDB in MongoDB (10 Marks)

Same as Slip 7 Q.2

Q.3) Social Media Platform Graph in Neo4j (10 Marks)

Data Creation:

```
CREATE (u1:User {name: 'User1'})
CREATE (u2:User {name: 'User2'})
CREATE (u3:User {name: 'User3'})

CREATE (u1)-[:FOLLOWS]->(u2)
CREATE (u1)-[:FOLLOWS]->(u3)
CREATE (u3)-[:FOLLOWS]->(u1)
```

1) Retrieve all users followed by User1:

```
MATCH (u1:User {name: 'User1'})-[:FOLLOWS]->(followed:User)
RETURN followed.name AS FollowedUser
```

2) Find users who follow back User1:

```
MATCH (u1:User {name: 'User1'})-[:FOLLOWS]->(other:User)-[:FOLLOWS]->(u1)
RETURN other.name AS FollowsBack
```

3) Delete user3:

```
MATCH (u3:User {name: 'User3'})
DETACH DELETE u3
```

SLIP NO. 20

Q.1) Game-Player Database in SQL (10 Marks)

Data Creation:

```
CREATE TABLE Game (
    game_name VARCHAR(50) PRIMARY KEY,
    no_of_players INT CHECK (no_of_players > 0),
    coach_name VARCHAR(50)
);

CREATE TABLE Player (
    pid INT PRIMARY KEY,
    pname VARCHAR(50),
    address VARCHAR(100),
    club_name VARCHAR(50)
);

CREATE TABLE Game_Player (
    game_name VARCHAR(50),
    pid INT,
    PRIMARY KEY (game_name, pid),
    FOREIGN KEY (game_name) REFERENCES Game(game_name),
    FOREIGN KEY (pid) REFERENCES Player(pid)
);

INSERT INTO Game VALUES
('Football', 11, 'Coach A'),
('Cricket', 15, 'Coach B'),
('Hockey', 11, 'Coach C'),
('Basketball', 5, 'Coach D');

INSERT INTO Player VALUES
(1, 'Suresh', 'Pune', 'Club X'),
(2, 'Ramesh', 'Mumbai', 'Club Y'),
(3, 'Sita', 'Delhi', 'Club X'),
(4, 'Geeta', 'Bangalore', 'Club Z'),
(5, 'Mohan', 'Chennai', 'Club X');

INSERT INTO Game_Player VALUES
('Football', 1), ('Football', 2), ('Cricket', 3),
('Cricket', 4), ('Hockey', 1), ('Hockey', 5);
```

1. Display games details having number of players more than 5:

```
SELECT * FROM Game WHERE no_of_players > 5;
```

2. Count the number of players in each club:

```
SELECT club_name, COUNT(*) AS PlayerCount FROM Player GROUP BY club_name;
```

3. Find the player details whose name starts with 'S':

```
SELECT * FROM Player WHERE pname LIKE 'S%';
```

Q.2) CompanyDB in MongoDB (10 Marks)

Same as Slip 6 Q.2

Q.3) Library System in Neo4j (10 Marks)

Data Creation:

```
CREATE (r1:Reader {name: 'Reader1'})
CREATE (r2:Reader {name: 'Reader2'})
CREATE (b1:Book {title: 'BookA'})
CREATE (b2:Book {title: 'BookB'})
CREATE (a1:Author {name: 'Author1'})

CREATE (r1)-[:BORROWED]->(b1)
CREATE (r2)-[:BORROWED]->(b2)
CREATE (b1)-[:WRITTEN_BY]->(a1)
```

1) Find all books borrowed by Reader1:

```
MATCH (r:Reader {name: 'Reader1'})-[:BORROWED]->(b:Book)
RETURN b.title AS BookTitle
```

2) Retrieve all books written by Author1:

```
MATCH (b:Book)-[:WRITTEN_BY]->(a:Author {name: 'Author1'})
RETURN b.title AS BookTitle
```

3) Delete the book BOOKB:

```
MATCH (b:Book {title: 'BookB'})
DETACH DELETE b
```

SLIP NO. 21

Q.1) Insurance System in Neo4j (10 Marks)

Data Creation:

```
// Create Policies
CREATE (pol1:Policy {policy_no: 'P202', type: 'Life', premium: 25000})
CREATE (pol2:Policy {policy_no: 'P303', type: 'Health', premium: 15000})
CREATE (pol3:Policy {policy_no: 'P404', type: 'Vehicle', premium: 10000})

// Create Customers
CREATE (c1:Customer {name: 'Ajay', id: 'C001'})
CREATE (c2:Customer {name: 'Priya', id: 'C002'})
CREATE (c3:Customer {name: 'Rahul', id: 'C003'})

// Create Agents
CREATE (a1:Agent {name: 'Agent Suresh', id: 'A001'})
CREATE (a2:Agent {name: 'Agent Meera', id: 'A002'})

// Create Claims
CREATE (cl1:Claim {claim_no: 'CL001', amount: 50000, status: 'Pending'})
CREATE (cl2:Claim {claim_no: 'CL002', amount: 30000, status: 'Approved'})
CREATE (cl3:Claim {claim_no: 'CL003', amount: 25000, status: 'Pending'})

// Relationships
CREATE (pol1)-[:OWNED_BY]->(c1)
CREATE (pol2)-[:OWNED_BY]->(c1)
CREATE (pol3)-[:OWNED_BY]->(c2)

CREATE (pol1)-[:SOLD_BY]->(a1)
CREATE (pol2)-[:SOLD_BY]->(a1)
CREATE (pol3)-[:SOLD_BY]->(a2)

CREATE (pol1)-[:HAS_CLAIM]->(cl1)
CREATE (pol2)-[:HAS_CLAIM]->(cl2)
CREATE (pol2)-[:HAS_CLAIM]->(cl3)
```

a. List all policies owned by "Ajay":

```
MATCH (pol:Policy)-[:OWNED_BY]->(c:Customer {name: 'Ajay'})
RETURN pol.policy_no AS PolicyNo, pol.type AS Type
```

b. Find claims under policy "P202":

```
MATCH (pol:Policy {policy_no: 'P202'})-[:HAS_CLAIM]->(cl:Claim)
RETURN cl.claim_no AS ClaimNo, cl.amount AS Amount, cl.status AS Status
```

c. Count the number of policies sold by each agent:

```
MATCH (pol:Policy)-[:SOLD_BY]->(a:Agent)
RETURN a.name AS Agent, COUNT(pol) AS PolicyCount
```

d. Delete all claims of policy "P303":

```
MATCH (pol:Policy {policy_no: 'P303'})-[r:HAS_CLAIM]->(cl:Claim)
DELETE r, cl
```

e. Update agent of policy "P404" to "Agent Suresh":

```
MATCH (pol:Policy {policy_no: 'P404'})-[r:SOLD_BY]->(a:Agent)
DELETE r
WITH pol
MATCH (newAgent:Agent {name: 'Agent Suresh'})
CREATE (pol)-[:SOLD_BY]->(newAgent)
```

Q.2) BookDB in MongoDB (10 Marks)

Same as Slip 2 Q.2

Q.3) Student-Teacher Database in SQL (10 Marks)

Same as Slip 2 Q.3

SLIP NO. 22

Q.1) Real Estate Management in Neo4j (10 Marks)

Data Creation:

```
// Create Properties
CREATE (pr1:Property {prop_id: 'PLOT101', type: '3BHK', price: 7500000})
CREATE (pr2:Property {prop_id: 'PLOT102', type: '2BHK', price: 5000000})
CREATE (pr3:Property {prop_id: 'PLOT103', type: 'Villa', price: 15000000})

// Create Owners
```

```
CREATE (o1:Owner {name: 'Ravi', id: 'O001'})
CREATE (o2:Owner {name: 'Sunita', id: 'O002'})
CREATE (o3:Owner {name: 'Arun', id: 'O003'})

// Create Agents
CREATE (a1:Agent {name: 'Agent Meera', id: 'A001'})
CREATE (a2:Agent {name: 'Agent Suresh', id: 'A002'})

// Create Locations
CREATE (l1:Location {name: 'Delhi', zone: 'North'})
CREATE (l2:Location {name: 'Mumbai', zone: 'West'})
CREATE (l3:Location {name: 'Pune', zone: 'West'})

// Relationships
CREATE (pr1)-[:OWNED_BY]->(o1)
CREATE (pr2)-[:OWNED_BY]->(o1)
CREATE (pr3)-[:OWNED_BY]->(o2)

CREATE (pr1)-[:MANAGED_BY]->(a1)
CREATE (pr2)-[:MANAGED_BY]->(a1)
CREATE (pr3)-[:MANAGED_BY]->(a2)

CREATE (pr1)-[:LOCATED_IN]->(l1)
CREATE (pr2)-[:LOCATED_IN]->(l2)
CREATE (pr3)-[:LOCATED_IN]->(l1)
```

a. List all properties owned by "Ravi":

```
MATCH (pr:Property)-[:OWNED_BY]->(o:Owner {name: 'Ravi'})
RETURN pr.prop_id AS PropertyID, pr.type AS Type, pr.price AS Price
```

b. Find properties located in "Delhi":

```
MATCH (pr:Property)-[:LOCATED_IN]->(l:Location {name: 'Delhi'})
RETURN pr.prop_id AS PropertyID, pr.type AS Type
```

c. Count the number of properties per location:

```
MATCH (pr:Property)-[:LOCATED_IN]->(l:Location)
RETURN l.name AS Location, COUNT(pr) AS PropertyCount
```

d. Delete all properties managed by "Agent Meera":

```
MATCH (pr:Property)-[:MANAGED_BY]->(a:Agent {name: 'Agent Meera'})
DETACH DELETE pr
```

e. Update location of property "PLOT101" to "Mumbai":

```
MATCH (pr:Property {prop_id: 'PLOT101'})-[r:LOCATED_IN]->(l:Location)
DELETE r
WITH pr
MATCH (newLoc:Location {name: 'Mumbai'})
CREATE (pr)-[:LOCATED_IN]->(newLoc)
```

Q.2) Olympic Information System in MongoDB (10 Marks)

Same as Slip 3 Q.2

Q.3) Property-Owner Database in SQL (10 Marks)

Same as Slip 3 Q.3

SLIP NO. 23

Q.1) Online Shopping System in Neo4j (10 Marks)

Same as Slip 4 Q.1

Q.2) Sales System in MongoDB (10 Marks)

Same as Slip 4 Q.2

Q.3) Game-Player Database in SQL (10 Marks)

Same as Slip 4 Q.3

SLIP NO. 24

Q.1) Logistics Management in Neo4j (10 Marks)

Data Creation:

```
// Create Packages
CREATE (pkg1:Package {pkg_id: 'PKG001', weight: 5, destination: 'Delhi'})
CREATE (pkg2:Package {pkg_id: 'PKG002', weight: 10, destination: 'Mumbai'})
CREATE (pkg3:Package {pkg_id: 'PKG003', weight: 3, destination: 'Pune'})

// Create Deliveries
CREATE (d1:Delivery {delivery_id: 'D123', date: '2024-01-15', status: 'Delivered'})
```

```
CREATE (d2:Delivery {delivery_id: 'D202', date: '2024-01-16', status: 'In Transit'})
CREATE (d3:Delivery {delivery_id: 'D303', date: '2024-01-17', status: 'Pending'})

// Create Vehicles
CREATE (v1:Vehicle {vehicle_id: 'V001', type: 'Truck T1', capacity: 1000})
CREATE (v2:Vehicle {vehicle_id: 'V002', type: 'Van V1', capacity: 500})

// Create Drivers
CREATE (dr1:Driver {name: 'Suresh', id: 'DR001'})
CREATE (dr2:Driver {name: 'Ajay', id: 'DR002'})
CREATE (dr3:Driver {name: 'Ramesh', id: 'DR003'})

// Relationships
CREATE (pkg1)-[:DELIVERED_BY]->(d1)
CREATE (pkg2)-[:DELIVERED_BY]->(d1)
CREATE (pkg3)-[:DELIVERED_BY]->(d2)

CREATE (d1)-[:CARRIED_IN]->(v1)
CREATE (d2)-[:CARRIED_IN]->(v2)
CREATE (d3)-[:CARRIED_IN]->(v1)

CREATE (v1)-[:DRIVEN_BY]->(dr1)
CREATE (v2)-[:DRIVEN_BY]->(dr2)
```

a. List all packages delivered by driver "Suresh":

```
MATCH (v:Vehicle)-[:DRIVEN_BY]->(dr:Driver {name: 'Suresh'})
MATCH (d:Delivery)-[:CARRIED_IN]->(v)
MATCH (pkg:Package)-[:DELIVERED_BY]->(d)
RETURN pkg.pkg_id AS PackageID, pkg.destination AS Destination
```

b. Find vehicles used for delivery "D123":

```
MATCH (d:Delivery {delivery_id: 'D123'})-[:CARRIED_IN]->(v:Vehicle)
RETURN v.vehicle_id AS VehicleID, v.type AS Type
```

c. Count the number of packages delivered by each driver:

```
MATCH (v:Vehicle)-[:DRIVEN_BY]->(dr:Driver)
MATCH (d:Delivery)-[:CARRIED_IN]->(v)
MATCH (pkg:Package)-[:DELIVERED_BY]->(d)
RETURN dr.name AS Driver, COUNT(pkg) AS PackageCount
```

d. Delete all deliveries made by driver "Ajay":

```
MATCH (v:Vehicle)-[:DRIVEN_BY]->(dr:Driver {name: 'Ajay'})
MATCH (d:Delivery)-[:CARRIED_IN]->(v)
DETACH DELETE d
```

e. Update vehicle of delivery "D202" to "Truck T1":

```
MATCH (d:Delivery {delivery_id: 'D202'})-[r:CARRIED_IN]->(v:Vehicle)
DELETE r
WITH d
MATCH (newVehicle:Vehicle {type: 'Truck T1'})
CREATE (d)-[:CARRIED_IN]->(newVehicle)
```

Q.2) LibraryDB in MongoDB (10 Marks)

Same as Slip 5 Q.2

Q.3) Banking System in SQL (10 Marks)

Same as Slip 5 Q.3

SLIP NO. 25

Q.1) Recruitment Management in Neo4j (10 Marks)

Data Creation:

```
// Create Recruiters
CREATE (r1:Recruiter {name: 'Rahul', id: 'R001'})
CREATE (r2:Recruiter {name: 'Recruiter Meena', id: 'R002'})
CREATE (r3:Recruiter {name: 'Amit', id: 'R003'})

// Create Jobs
CREATE (j1:Job {job_id: 'J101', title: 'Software Developer', salary: 800000})
CREATE (j2:Job {job_id: 'J202', title: 'Data Analyst', salary: 600000})
CREATE (j3:Job {job_id: 'J303', title: 'Project Manager', salary: 1200000})

// Create Candidates
CREATE (c1:Candidate {name: 'Sita', id: 'C001'})
CREATE (c2:Candidate {name: 'Mohan', id: 'C002'})
CREATE (c3:Candidate {name: 'Geeta', id: 'C003'})

// Create Interviews
CREATE (i1:Interview {int_id: 'INT001', date: '2024-01-20', result: 'Selected'})
CREATE (i2:Interview {int_id: 'INT002', date: '2024-01-21', result: 'Pending'})
```

```
CREATE (i3:Interview {int_id: 'INT003', date: '2024-01-22', result: 'Rejected'})
CREATE (i4:Interview {int_id: 'INT004', date: '2024-01-23', result: 'Pending'})

// Relationships
CREATE (r1)-[:SCHEDULED]->(i1)
CREATE (r1)-[:SCHEDULED]->(i2)
CREATE (r2)-[:SCHEDULED]->(i3)
CREATE (r3)-[:SCHEDULED]->(i4)

CREATE (c1)-[:INTERVIEWED]->(i1)
CREATE (c1)-[:INTERVIEWED]->(i3)
CREATE (c2)-[:INTERVIEWED]->(i2)
CREATE (c3)-[:INTERVIEWED]->(i4)

CREATE (i1)-[:FOR_JOB]->(j1)
CREATE (i2)-[:FOR_JOB]->(j2)
CREATE (i3)-[:FOR_JOB]->(j1)
CREATE (i4)-[:FOR_JOB]->(j3)
```

a. List all candidates interviewed by recruiter "Rahul":

```
MATCH (r:Recruiter {name: 'Rahul'})-[:SCHEDULED]->(i:Interview)<-[:INTERVIEWED]-(c:Candid
RETURN DISTINCT c.name AS CandidateName
```

b. Find all jobs for which interviews were scheduled:

```
MATCH (i:Interview)-[:FOR_JOB]->(j:Job)
RETURN DISTINCT j.job_id AS JobID, j.title AS JobTitle
```

c. Count the number of interviews conducted by each recruiter:

```
MATCH (r:Recruiter)-[:SCHEDULED]->(i:Interview)
RETURN r.name AS Recruiter, COUNT(i) AS InterviewCount
```

d. Delete all interviews of candidate "Sita":

```
MATCH (c:Candidate {name: 'Sita'})-[r:INTERVIEWED]->(i:Interview)
DETACH DELETE i
```

e. Update recruiter of job "J202" to "Recruiter Meena":

```
MATCH (i:Interview)-[:FOR_JOB]->(j:Job {job_id: 'J202'})
MATCH (r:Recruiter)-[rel:SCHEDULED]->(i)
DELETE rel
WITH i
```

```
MATCH (newRecruiter:Recruiter {name: 'Recruiter Meena'})
CREATE (newRecruiter)-[:SCHEDULED]->(i)
```

Q.2) BookDB in MongoDB (10 Marks)

Same as Slip 2 Q.2

Q.3) Student-Teacher Database in SQL (10 Marks)

Same as Slip 2 Q.3

Summary of Repeated Questions

Question	Same As
Slip 9 Q.2 (LibraryDB MongoDB)	Slip 10, 16, 17 Q.2
Slip 2 Q.2 (BookDB MongoDB)	Slip 14, 21, 25 Q.2
Slip 1 Q.2 (StudentDB MongoDB)	Slip 15 Q.2
Slip 3 Q.2 (Olympic MongoDB)	Slip 13, 22 Q.2
Slip 4 Q.2 (Sales MongoDB)	Slip 12, 23 Q.2
Slip 5 Q.2 (LibraryDB MongoDB)	Slip 24 Q.2
Slip 6 Q.2 (CompanyDB MongoDB)	Slip 20 Q.2
Slip 7 Q.2 (SchoolDB MongoDB)	Slip 19 Q.2
Slip 8 Q.2 (ShopDB MongoDB)	Slip 18 Q.2
Slip 2 Q.3 (Student-Teacher SQL)	Slip 21, 25 Q.3
Slip 3 Q.3 (Property-Owner SQL)	Slip 22 Q.3
Slip 4 Q.1 (Online Shopping Neo4j)	Slip 23 Q.1
Slip 4 Q.3 (Game-Player SQL)	Slip 23 Q.3
Slip 5 Q.3 (Banking SQL)	Slip 24 Q.3
Slip 8 Q.3 (Project-Employee SQL)	Slip 11 Q.3
Slip 11 Q.1 (Supply Chain Neo4j)	Slip 13 Q.3, 15 Q.3

This completes all 25 slips with comprehensive answers including data creation/insertion, questions, and their solutions for Neo4j (Cypher), MongoDB, and SQL. Good luck with your ADT practicals!

⁂

1. 2025MSC-CS-ADT-Practical-Slips-1.pdf