Slip -11

Q.1) Consider the Book table

a)List the number of books published in each year sorted by publication year in ascending order

b)Find the authors who have written more than one book along with the total number of books they have written sorted by the number of books in descending order

CREATE TABLE Book (

book\_id INT PRIMARY KEY, title VARCHAR(100),

author VARCHAR(100), publication year INT, language VARCHAR(50), avaiIabIe\_copies INT, totaI\_copies INT

)

INSERT INTO Book (book\_id, title, author, pubIication\_year, language, avaiIabIe\_copies, totaI\_copies)

VALUES

(1, 'Book1', 'Author1', 2000, 'English', 10, 100),

(2, 'Book2', 'Author2', 2005, 'English', 15, 150),

(3, 'Book3', 'Author1', 2000, 'English', 20, 200),

(4, 'Book4', 'Author3', 2010, 'English', 25, 250),

(5, 'Book5', 'Author4', 2015, 'English', 30, 300);

SELECT publication year, COUNT(\*) AS num books published FROM Book

GROUP BY publication year ORDER BY publication year ASC;

SELECT author, COUNT(\*) AS num\_books\_written FROM Book

GROUP BY author HAVING COUNT(\*) > 1

ORDER BY num books written DESC;

Q.2) Model the following sales system as a document database

// Create Products collection

db.createCollection("Products")

db.Products.insertMany([

{product\_id: 1, name: "Product A", price: 100, quantity: 50},

{product\_id: 2, name: "Product B", price: 150, quantity: 30},

{product\_id: 3, name: "Product C", price: 200, quantity: 20},

{product\_id: 4, name: "Product D", price: 250, quantity: 10},

{product\_id: 5, name: "Product E", price: 300, quantity: 25}

])

// Create Customers collection

db.createCollection("Customers")

db.Customers.insertMany([

{customer\_id: 1, name: "Mr. X", email: "x@example.com", address: "Address 1"},

{customer\_id: 2, name: "Mr. Y", email: "y@example.com", address: "Address 2"},

{customer\_id: 3, name: "Mr. Z", email: "z@example.com", address: "Address 3"},

{customer\_id: 4, name: "Mr. A", email: "a@example.com", address: "Address 4"},

{customer\_id: 5, name: "Mr. B", email: "b@example.com", address: "Address 5"}

])

// Create Orders collection

db.createCollection("Orders")

db.Orders.insertMany([

{order\_id: 1, customer\_id: 1, product\_id: 1, quantity: 2, total\_value: 200, processed: true},

{order\_id: 2, customer\_id: 2, product\_id: 3, quantity: 3, total\_value: 600, processed: false},

{order\_id: 3, customer\_id: 3, product\_id: 2, quantity: 1, total\_value: 150, processed: true},

{order\_id: 4, customer\_id: 4, product\_id: 4, quantity: 2, total\_value: 500, processed: false},

{order\_id: 5, customer\_id: 5, product\_id: 5, quantity: 2, total\_value: 600, processed: true}

])

// Create Invoices collection

db.createCollection("Invoices")

db.Invoices.insertMany([

{invoice\_id: 1, order\_id: 1, invoice\_date: ISODate("2024-03-25"), amount: 200},

{invoice\_id: 2, order\_id: 3, invoice\_date: ISODate("2024-03-27"), amount: 150},

{invoice\_id: 3, order\_id: 5, invoice\_date: ISODate("2024-03-28"), amount: 600}

])

// A. List all products in the inventory

db.Products.find({})

// B. List the details of orders with a value >20000

db.Orders.find({total\_value: {$gt: 20000}})

// C. List all the orders which have not been processed (invoice not generated)

db.Orders.find({processed: false})

// D. List all the orders along with their invoice for “Mr. Rajiv”

db.Orders.aggregate([

{

$lookup: {

from: "Invoices",

localField: "order\_id",

foreignField: "order\_id",

as: "invoice"

}

},

{

$match: {

"invoice": { $ne: [] }

}

},

{

$lookup: {

from: "Customers",

localField: "customer\_id",

foreignField: "customer\_id",

as: "customer"

}

},

{

$match: {

"customer.name": "Mr. Rajiv"

}

}

])

Slip-12

Q.1 Consider the Worker table.

a) Find the departments where the number of workers is equal to the number of distinct first names, sorted by department name

b) List the number of workers in each department who joined after January 1, 2021, sorted by department name

CREATE TABLE Worker ( WORKER\_ID INT PRIMARY KEY, FIRST\_NAME VARCHAR(50), LAST\_NAME VARCHAR(50), SALARY DECIMAL(10, 2), JOINING\_DATE DATE, DEPARTMENT VARCHAR(50)

)

INSERT INTO Worker (WORKER\_ID, FIRST\_NAME, LAST\_NAME, SALARY, JOINING\_DATE, DEPARTMENT)

VALUES

(1, 'John', 'Doe', 50000.00, '2021-03-15', 'Finance’),

(2, 'Jane', 'Smith', 60000.00, '2021-02-20', 'HR'),

(3, 'Michael', 'Johnson', 55000.00, '2020-12-10', 'IT’),

(4, 'Emily', 'Brown', 52000.00, '2021-05-05', 'Finance’),

(5, 'David', 'Jones', 48000.00, '2021-06-30', 'IT');

SELECT DEPARTMENT

FROM Worker

GROUP BY DEPARTMENT

HAVING COUNT(\*) = COUNT(DISTINCT FIRST\_NAME) ORDER BY DEPARTMENT;

SELECT DEPARTMENT, COUNT(\*) AS NumOfWorkers

FROM Worker

WHERE JOINING\_DATE > '2021-01-01' GROUP BY DEPARTMENT

ORDER BY DEPARTMENT;

Q.2 Model the following University information system as a graph model, and answer the following queries using Cypher.

CREATE

(Physics:Department {name: "Physics"}),

(Geography:Department {name: "Geography"}),

(Computer:Department {name: "Computer"}),

(Mathematics:Department {name: "Mathematics"}),

(course1:Course {name: "Physics 101"}),

(course2:Course {name: "Physics 201"}),

(course3:Course {name: "Geography 101"}),

(course4:Course {name: "Computer Science 101"}),

(course5:Course {name: "Mathematics 101"}),

(person1:Person {name: "John"}),

(person2:Person {name: "Alice"}),

(Physics)-[:DEPARTMENT\_CONDUCTS\_COURSE]->(course1),

(Physics)-[:DEPARTMENT\_CONDUCTS\_COURSE]->(course2),

(Geography)-[:DEPARTMENT\_CONDUCTS\_COURSE]->(course3),

(Computer)-[:DEPARTMENT\_CONDUCTS\_COURSE]->(course4),

(Mathematics)-[:DEPARTMENT\_CONDUCTS\_COURSE]->(course5),

(course1)-[:COURSE\_HAS\_RECOMMENDATION]->(person1),

(course1)-[:COURSE\_HAS\_RECOMMENDATION]->(person2),

(course2)-[:COURSE\_HAS\_RECOMMENDATION]->(person1),

(course3)-[:COURSE\_HAS\_RECOMMENDATION]->(person2),

(course4)-[:COURSE\_HAS\_RECOMMENDATION]->(person1),

(course5)-[:COURSE\_HAS\_RECOMMENDATION]->(person2)

Q1) Retrieve all departments:

MATCH (d:Department)

RETURN d;

Q2) Retrieve courses conducted by the Physics department:

MATCH (d:Department {name: "Physics"})-[:DEPARTMENT\_CONDUCTS\_COURSE]->(c:Course)

RETURN c.name;

Q3) Retrieve the most recommended course in the Geography department:

MATCH (d:Department {name: "Geography"})-[:DEPARTMENT\_CONDUCTS\_COURSE]->(c:Course)-[:COURSE\_HAS\_RECOMMENDATION]->(p:Person)

RETURN c.name, COUNT(p) AS recommendations

ORDER BY recommendations DESC

LIMIT 1;

Q4) Retrieve common courses between Mathematics and Computer departments:

MATCH (math:Department {name: "Mathematics"})-[:DEPARTMENT\_CONDUCTS\_COURSE]->(math\_course:Course),

(comp:Department {name: "Computer"})-[:DEPARTMENT\_CONDUCTS\_COURSE]->(comp\_course:Course)

WHERE math\_course.name = comp\_course.name

RETURN math\_course.name;

Slip-13

Q.1 Consider the Book table

a) book (book id,titIe,author,pubIication\_year,Ianguage,avaiIabIe\_copies,totaI\_copies) Write a query to insert 5 rows in it that as per the query requirements.

b) Find the authors who have written books in more than one language, along with the total number of languages they have written books in, sorted by author name:

List the titles of books along with the average number of available copies per book, and display only those books where the average number of available copies is less than 5, sorted by average available copies in descending order

CREATE TABLE Book (

book\_id INT PRIMARY KEY, title VARCHAR(100),

author VARCHAR(100), publication year INT, language VARCHAR(50), avaiIabIe\_copies INT, totaI\_copies INT

)

INSERT INTO Book (book\_id, title, author, pubIication\_year, language, avaiIabIe\_copies, totaI\_copies)

VALUES

(1, 'Book1', 'Author1', 2010, 'English', 10, 20),

(2, 'Book2', 'Author2', 2015, 'Spanish', 5, 10),

(3, 'Book3', 'Author1', 2018, 'French', 3, 8),

(4, 'Book4', 'Author3', 2020, 'English', 8, 15),

(5, 'Book5', 'Author2', 2012, 'German', 6, 12);

SELECT author, COUNT(DISTINCT language) AS num languages FROM Book

GROUP BY author

HAVING COUNT(DISTINCT language) > 1

ORDER BY author;

SELECT title, AVG(avaiIabIe\_copies) AS avg\_avaiIabIe\_copies FROM Book

GROUP BY title

HAVING AVG(available copies) < 5 ORDER BY avg available copies DESC;

Q.2 Model the following Library information system as a graph model,and answer the following queries using Cypher.

CREATE

(student1:Student {name: "Alice"}),

(student2:Student {name: "Bob"}),

(book1:Book {name: "Introduction to Algorithms"}),

(book2:Book {name: "Database Systems"}),

(book3:Book {name: "The Art of Computer Programming"}),

(type1:Type {name: "Text"}),

(type2:Type {name: "Reference"}),

(type3:Type {name: "Bibliography"}),

(student1)-[:STUDENT\_BOUGHT\_BOOK]->(book1)-[:BOOK\_OF\_TYPE]->(type1),

(student1)-[:STUDENT\_BOUGHT\_BOOK]->(book2)-[:BOOK\_OF\_TYPE]->(type2),

(student2)-[:STUDENT\_BOUGHT\_BOOK]->(book3)-[:BOOK\_OF\_TYPE]->(type1),

(student1)-[:STUDENT\_RECOMMENDED\_BOOK]->(book1),

(student1)-[:STUDENT\_RECOMMENDED\_BOOK]->(book2),

(student2)-[:STUDENT\_RECOMMENDED\_BOOK]->(book3);

a) MATCH (b:Book)-[:BOOK\_OF\_TYPE]->(t:Type {name: "Text"}) RETURN b.name;

b) MATCH (s:Student)-[:STUDENT\_BOUGHT\_BOOK]->(:Book)-[:BOOK\_OF\_TYPE]->(t:Type)

WHERE t.name IN ["Text", "Reference"]

RETURN DISTINCT s.name;

c) MATCH (:Student)-[:STUDENT\_RECOMMENDED\_BOOK]->(b:Book)-[:BOOK\_OF\_TYPE]->(t:Type)

RETURN t.name, COUNT(b) AS recommendations ORDER BY recommendations DESC LIMIT 1;

d) MATCH (s:Student)-[:STUDENT\_BOUGHT\_BOOK]->(b:Book)-[:BOOK\_OF\_TYPE]->(t:Type)

WITH s, COUNT(DISTINCT t) AS numTypes WHERE numTypes > 1 RETURN s.name;

Slip-14

**Q.1 consider a book table**

a) Write a query to insert 5 rows in it that as per the query requirements

b) Find the most common publication language among books along with the total number of books published in each language sorted by'the number of books in descending order

Find the authors who have written books with the highest average number of total copies available and display the top 3 authors with the highest average sorted by average copies in descending order

CREATE TABLE Book (

book\_id INT PRIMARY KEY, title VARCHAR(100),

author VARCHAR(100), publication year INT, language VARCHAR(50), avaiIabIe\_copies INT, totaI\_copies INT

)

INSERT INTO Book (book\_id, title, author, pubIication\_year, language, avaiIabIe\_copies, totaI\_copies)

VALUES

(1, 'Book1', 'Author1', 2010, 'English', 10, 20),

(2, 'Book2', 'Author2', 2015, 'Spanish', 5, 10),

(3, 'Book3', 'Author1', 2018, 'French', 3, 8),

(4, 'Book4', 'Author3', 2020, 'English', 8, 15),

(5, 'Book5', 'Author2', 2012, 'German', 6, 12);

SELECT language, COUNT(\*) AS num books FROM Book

GROUP BY language

ORDER BY num\_books DESC;

SELECT author, AVG(totaI\_copies) AS avg\_copies FROM Book

GROUP BY author

ORDER BY avg copies DESC LIMIT 3;

**Q.2 Model the following Automobile information system as a graph model,and answer the following queries using Cypher.**

CREATE (:Vehicle {type: 'Two-Wheeler', characteristics: 'Small, agile'})

CREATE (:Vehicle {type: 'Four-Wheeler', characteristics: 'Spacious, stable'})

CREATE (:Customer {name: 'John'})

CREATE (:Customer {name: 'Alice'})

MATCH (c:Customer {name: 'John'}), (v:Vehicle {type: 'Two-Wheeler'})

CREATE (c)-[:PURCHASED\_BY]->(v)

MATCH (c:Customer {name: 'Alice'}), (v:Vehicle {type: 'Four-Wheeler'})

CREATE (c)-[:PURCHASED\_BY]->(v)

MATCH (c:Customer {name: 'John'}), (v:Vehicle {type: 'Two-Wheeler'})

CREATE (c)-[:RECOMMENDED\_BY]->(v)

MATCH (c:Customer {name: 'Alice'}), (v:Vehicle {type: 'Four-Wheeler'})

CREATE (c)-[:RECOMMENDED\_BY]->(v)// A. List the characteristics of four-wheeler types

MATCH (v:Vehicle {type: 'Four-Wheeler'})

RETURN v.characteristics;

// B. List the name of customers who bought a two-wheeler vehicle

MATCH (c:Customer)-[:PURCHASED\_BY]->(v:Vehicle {type: 'Two-Wheeler'})

RETURN c.name;

// C. List the customers who bought more than one type of vehicle

MATCH (c:Customer)-[:PURCHASED\_BY]->(v:Vehicle)

WITH c, COUNT(DISTINCT v) AS numVehicles

WHERE numVehicles > 1

RETURN c.name;

// D. List the most recommended vehicle type

MATCH (c:Customer)-[:RECOMMENDED\_BY]->(v:Vehicle)

WITH v, COUNT(c) AS numRecommendations

ORDER BY numRecommendations DESC

LIMIT 1

RETURN v.type;

Slip-15

**Q.1 Consider the book table**

a) List the titles of books along with the total number of available copies for each book and display only those books where the total number of available copies is less than the total number of copies sorted by total available copies in ascending order

b) Find the authors who have written books with the highest difference between available copies and total copies and display the top 3 authors with the highest difference, sorted by difference in descending order

CREATE TABLE Book (

book\_id INT PRIMARY KEY, title VARCHAR(100),

author VARCHAR(100), pubIication\_year INT, language VARCHAR(50), avaiIabIe\_copies INT, totaI\_copies INT

)

INSERT INTO Book (book\_id, title, author, pubIication\_year, language, avaiIabIe\_copies, total copies)

VALUES

(1, 'Book1', 'Author1', 2010, 'English', 10, 20),

(2, 'Book2', 'Author2', 2015, 'Spanish', 5, 10),

(3, 'Book3', 'Author1', 2018, 'French', 3, 8),

(4, 'Book4', 'Author3', 2020, 'English', 8, 15),

(5, 'Book5', 'Author2', 2012, 'German', 6, 12);

SELECT title, available copies FROM Book

WHERE available copies < total copies ORDER BY available copies ASC;

SELECT author, (totaI\_copies - avaiIabIe\_copies) AS difference FROM Book

GROUP BY author

ORDER BY difference DESC LIMIT 3;

**Q.2 Model the following Car Showroom information as a graph model,and**

**answer the queries using Cypher.**

// Creating CarModel nodes

CREATE (:CarModel {modelName: 'Honda City'}),

(:CarModel {modelName: 'Skoda'}),

(:CarModel {modelName: 'Creta'}),

(:CarModel {modelName: 'Swift'}),

(:CarModel {modelName: 'Ertiga'});

// Creating SalesStaff nodes

CREATE (:SalesStaff {staffName: 'Mr. Narayan'}),

(:SalesStaff {staffName: 'Ms. Smith'}),

(:SalesStaff {staffName: 'Mr. Patel'});

// Creating Relationships: BELONGS\_TO

MATCH (s:SalesStaff), (c:CarModel)

WHERE s.staffName = 'Mr. Narayan' AND c.modelName IN ['Honda City', 'Skoda']

CREATE (s)-[:BELONGS\_TO]->(c);

MATCH (s:SalesStaff), (c:CarModel)

WHERE s.staffName = 'Ms. Smith' AND c.modelName IN ['Creta', 'Swift']

CREATE (s)-[:BELONGS\_TO]->(c);

MATCH (s:SalesStaff), (c:CarModel)

WHERE s.staffName = 'Mr. Patel' AND c.modelName IN ['Ertiga']

CREATE (s)-[:BELONGS\_TO]->(c);

// Creating Customer nodes and relationships

CREATE (:Customer {customerName: 'John'}),

(:Customer {customerName: 'Alice'}),

(:Customer {customerName: 'Bob'});

// Creating Relationships: ENQUIRED\_ABOUT and PURCHASED

MATCH (c:Customer), (cm:CarModel)

WHERE c.customerName = 'John' AND cm.modelName IN ['Skoda']

CREATE (c)-[:ENQUIRED\_ABOUT {enquiryDate: date('2024-03-29')}]->(cm);

MATCH (c:Customer), (cm:CarModel)

WHERE c.customerName = 'Alice' AND cm.modelName IN ['Swift']

CREATE (c)-[:ENQUIRED\_ABOUT {enquiryDate: date('2024-03-28')}]->(cm);

MATCH (c:Customer), (cm:CarModel)

WHERE c.customerName = 'Bob' AND cm.modelName IN ['Creta']

CREATE (c)-[:ENQUIRED\_ABOUT {enquiryDate: date('2024-03-27')}]->(cm),

(c)-[:PURCHASED {purchaseDate: date('2024-03-28')}]->(cm);

A.] MATCH (c:CarModel)

RETURN c.modelName;

B.] MATCH (:SalesStaff {staffName: 'Mr. Narayan'})-[:BELONGS\_TO]->(c:CarModel)

RETURN c.modelName;

C.] MATCH (c:Customer)-[e:ENQUIRED\_ABOUT]->()

WHERE NOT exists((c)-[:PURCHASED]->())

RETURN c.customerName;

D.] MATCH (c:CarModel)<-[p:PURCHASED]-()

RETURN c.modelName, COUNT(p) AS sales

ORDER BY sales DESC

LIMIT 1;

Slip-16

**Q.1 Consider the course table**

a) List the number of courses in each category, sorted by category name:

b) Find the instructors who teach more than one course, along with the total number of courses they teach, sorted by the number of courses in descending order:

CREATE TABLE Courses ( CourselD INT PRIMARY KEY, Title VARCHAR(100),

Instructor VARCHAR(100), Category VARCHAR(50), Price DECIMAL(10, 2),

Duration INT, EnrollmentCount INT

)

INSERT INTO Courses (CourselD, Title, Instructor, Category, Price, Duration, EnrollmentCount) VALUES

(1, 'Course1', 'Instructor1', 'Programming', 99.99, 30, 100),

(2, 'Course2', 'Instructor2', 'Data Science', 149.99, 45, 80),

(3, 'Course3', 'Instructor1', 'Web Development', 79.99, 40, 120),

(4, 'Course4', 'Instructor3', 'Programming', 129.99, 35, 90),

(5, 'Course5', 'Instructor2', 'Data Science', 199.99, 50, 70);

SELECT Category, COUNT(\*) AS CourseCount FROM Courses

GROUP BY Category ORDER BY Category;

SELECT Instructor, COUNT(\*) AS CourseCount FROM Courses

GROUP BY Instructor HAVING COUNT(\*) > 1

ORDER BY CourseCount DESC;

**Q.2) Model the following Medical information as a graph model, and answer the following queries using Cypher.**

// Creating MedicineBrand nodes

CREATE (:MedicineBrand {brandName: 'Dr. Reddy'}),

(:MedicineBrand {brandName: 'Cipla'}),

(:MedicineBrand {brandName: 'SunPharma'});

// Creating State nodes

CREATE (:State {stateName: 'Rajasthan'}),

(:State {stateName: 'Gujarat'}),

(:State {stateName: 'Maharashtra'});

// Creating MedicineType nodes

CREATE (:MedicineType {typeName: 'Tablet'}),

(:MedicineType {typeName: 'Syrup'}),

(:MedicineType {typeName: 'Powder'});

// Creating Medicine nodes

CREATE (:Medicine {medicineName: 'Medicine A', usePercentage: 95}),

(:Medicine {medicineName: 'Medicine B', usePercentage: 85}),

(:Medicine {medicineName: 'Medicine C', usePercentage: 40});

// Creating Relationships: MANUFACTURES

MATCH (b:MedicineBrand), (m:Medicine)

WHERE b.brandName = 'Dr. Reddy' AND m.medicineName IN ['Medicine A', 'Medicine B']

CREATE (b)-[:MANUFACTURES]->(m);

MATCH (b:MedicineBrand), (m:Medicine)

WHERE b.brandName = 'Cipla' AND m.medicineName IN ['Medicine C']

CREATE (b)-[:MANUFACTURES]->(m);

// Creating Relationships: USES

MATCH (s:State), (m:Medicine)

WHERE s.stateName = 'Rajasthan' AND m.medicineName IN ['Medicine A', 'Medicine B']

CREATE (s)-[:USES]->(m);

MATCH (s:State), (m:Medicine)

WHERE s.stateName = 'Gujarat' AND m.medicineName IN ['Medicine A', 'Medicine C']

CREATE (s)-[:USES]->(m);

A.] MATCH (m:Medicine)

RETURN m.medicineName;

B.] MATCH (s:State {stateName: 'Rajasthan'})-[:USES]->(m:Medicine)

WHERE m.usePercentage >= 90

RETURN m.medicineName;

C.] MATCH (s:State {stateName: 'Gujarat'})-[:USES]->(m:Medicine)-[:BELONGS\_TO]->(:MedicineType {typeName: 'Tablet'})

WHERE m.usePercentage >= 90

RETURN m.medicineName;

D.] MATCH (m:Medicine)-[:BELONGS\_TO]->(:MedicineType {typeName: 'Powder'})

RETURN m.medicineName;

Slip-17

**Q.1)Consider the course table**

a) List the categories along with the total number of courses in each category, and display only those categories with more than 5 courses, sorted by the number of courses in descending order:

b)Find the categories where the average duration of courses is less than 8 weeks, sorted by category name:

CREATE TABLE Courses ( CourselD INT PRIMARY KEY, Title VARCHAR(255),

Instructor VARCHAR(255), Category VARCHAR(50), Price DECIMAL(10, 2),

Duration INT, -- Assuming duration is in weeks EnrollmentCount INT

)

INSERT INTO Courses (CourselD, Title, Instructor, Category, Price, Duration, EnrollmentCount) VALUES

(1, 'Introduction to Programming', 'John Smith', 'Programming', 99.99, 6, 100),

(2, 'Data Structures and Algorithms', 'Jane Doe', 'Programming', 129.99, 10, 85),

(3, 'Introduction to Statistics', 'Alice Johnson', 'Mathematics', 79.99, 8, 75),

(4, 'Calculus I', ’Michael Brown', 'Mathematics', 89.99, 12, 60),

(5, 'English Grammar Basics', 'Emily Wilson', 'Language', 49.99, 4, 120),

(6, 'Advanced French Conversation', 'Sophie Martin', 'Language', 79.99, 6, 55),

(7, 'Introduction to Physics', 'David Clark', 'Science', 69.99, 6, 90),

(8, 'Chemistry Fundamentals', 'Mark Thompson', 'Science', 79.99, 8, 70),

(9, 'Art History: Renaissance', 'Olivia Parker', 'Art', 59.99, 6, 40),

(10, 'Digital Photography Basics', 'Daniel White', 'Art', 69.99, 4, 65);

SELECT Category, COUNT(\*) AS TotalCourses FROM Courses

GROUP BY Category HAVING COUNT(\*) > 5

ORDER BY TotalCourses DESC;

SELECT Category, AVG(Duration) AS AverageDuration FROM Courses

GROUP BY Category

HAVING AVG(Duration) < 8 ORDER BY Category;

**Q.2) Model the following nursery management information as a graph model, and answer the following queries using Cypher.**

// Creating Plant nodes

CREATE (:Plant {plantName: 'Rose', plantType: 'Flowering', quantity: 1000}),

(:Plant {plantName: 'Tulip', plantType: 'Flowering', quantity: 800}),

(:Plant {plantName: 'Lavender', plantType: 'Non-flowering', quantity: 600}),

(:Plant {plantName: 'Creeper', plantType: 'Climber', quantity: 1200});

// Creating Fertilizer nodes

CREATE (:Fertilizer {fertilizerName: 'Nitrogen', quantity: 500}),

(:Fertilizer {fertilizerName: 'Phosphorus', quantity: 300}),

(:Fertilizer {fertilizerName: 'Potassium', quantity: 400});

// Creating Product nodes

CREATE (:Product {productName: 'Garden Shears', productType: 'Tool', quantity: 200}),

(:Product {productName: 'Pots', productType: 'Container', quantity: 300}),

(:Product {productName: 'Mulch', productType: 'Material', quantity: 600});

// Creating Customer nodes

CREATE (:Customer {customerName: 'Alice', visitDate: date('2024-03-27')}),

(:Customer {customerName: 'Bob', visitDate: date('2024-03-28')}),

(:Customer {customerName: 'Charlie', visitDate: date('2024-03-29')});

// Creating Supplier nodes

CREATE (:Supplier {supplierName: 'Supplier X'}),

(:Supplier {supplierName: 'Supplier Y'}),

(:Supplier {supplierName: 'Supplier Z'});

// Creating Relationships: PURCHASED

MATCH (c:Customer), (p:Plant)

WHERE c.customerName = 'Alice' AND p.plantName IN ['Rose', 'Tulip']

CREATE (c)-[:PURCHASED {purchaseDate: date('2024-03-27'), quantity: 600}]->(p);

MATCH (c:Customer), (p:Plant)

WHERE c.customerName = 'Bob' AND p.plantName IN ['Rose', 'Creeper']

CREATE (c)-[:PURCHASED {purchaseDate: date('2024-03-28'), quantity: 700}]->(p);

MATCH (c:Customer), (p:Plant)

WHERE c.customerName = 'Charlie' AND p.plantName IN ['Tulip', 'Creeper']

CREATE (c)-[:PURCHASED {purchaseDate: date('2024-03-29'), quantity: 800}]->(p);

// Creating Relationships: RECOMMENDED

MATCH (c:Customer)

WHERE c.customerName = 'Alice'

CREATE (c)-[:RECOMMENDED {rating: 5, recommendation: 'Great app!'}]->(:App);

MATCH (c:Customer)

WHERE c.customerName = 'Bob'

CREATE (c)-[:RECOMMENDED {rating: 4, recommendation: 'Good app.'}]->(:App);

MATCH (c:Customer)

WHERE c.customerName = 'Charlie'

CREATE (c)-[:RECOMMENDED {rating: 4, recommendation: 'Nice app.'}]->(:App);

// Creating Relationships: SUPPLIES

MATCH (s:Supplier), (p:Plant)

WHERE s.supplierName = 'Supplier X' AND p.plantName IN ['Rose', 'Tulip']

CREATE (s)-[:SUPPLIES {supplyDate: date('2024-03-27')}]->(p);

MATCH (s:Supplier), (p:Plant)

WHERE s.supplierName = 'Supplier Y' AND p.plantName IN ['Creeper']

CREATE (s)-[:SUPPLIES {supplyDate: date('2024-03-28')}]->(p);

MATCH (s:Supplier), (p:Plant)

WHERE s.supplierName = 'Supplier Z' AND p.plantName IN ['Lavender']

CREATE (s)-[:SUPPLIES {supplyDate: date('2024-03-29')}]->(p);

A.] MATCH (p:Plant)

RETURN DISTINCT p.plantType;

B.] MATCH (p:Plant)

WHERE p.plantType = 'Flowering'

RETURN p.plantName;

C.] MATCH (c:Customer)-[purchase:PURCHASED]->(p:Plant)

WHERE purchase.purchaseDate >= date('2024-03-28') AND purchase.quantity > 500

RETURN p.plantName;

D.] MATCH (s:Supplier)-[supplies:SUPPLIES]->(p:Plant)

WHERE p.plantName = 'Creeper'

RETURN s.supplierName

ORDER BY supplies.supplyDate DESC;

Slip-18

**Q.1)consider the course table**

a) List the instructors along with the total number of courses they teach, and display only those instructors who teach courses with more than 500 enrollments, sorted by the number of courses in descending order:

b) Find the categories where the average enrollment count of courses is greater than 700, sorted by average enrollment count in descending order:

CREATE TABLE Courses ( CourselD INT PRIMARY KEY, Title VARCHAR(255),

Instructor VARCHAR(255), Category VARCHAR(50), Price DECIMAL(10, 2),

Duration INT, -- Assuming duration is in weeks EnrollmentCount INT

)

INSERT INTO Courses (CourselD, Title, Instructor, Category, Price, Duration, EnrollmentCount) VALUES

(1, 'Introduction to Programming', 'John Smith', 'Programming', 99.99, 6, 600),

(2, 'Data Structures and Algorithms', 'Jane Doe', 'Programming', 129.99, 10, 700),

(3, 'Introduction to Statistics', 'Alice Johnson', 'Mathematics', 79.99, 8, 450),

(4, 'Calculus I', ’Michael Brown', 'Mathematics', 89.99, 12, 800),

(5, 'English Grammar Basics', 'Emily Wilson', 'Language', 49.99, 4, 300),

(6, 'Advanced French Conversation', 'Sophie Martin', 'Language', 79.99, 6, 550),

(7, 'Introduction to Physics', 'David Clark', 'Science', 69.99, 6, 900),

(8, 'Chemistry Fundamentals', 'Mark Thompson', 'Science', 79.99, 8, 750),

(9, 'Art History: Renaissance', 'Olivia Parker', 'Art', 59.99, 6, 400),

(10, 'Digital Photography Basics', 'Daniel White', 'Art', 69.99, 4, 650);

SELECT Instructor, COUNT(\*) AS TotalCourses FROM Courses

WHERE EnrollmentCount > 500 GROUP BY Instructor

HAVING COUNT(\*) > 0

ORDER BY TotalCourses DESC;

SELECT Category, AVG(EnroIImentCount) AS AvgEnrollment

FROM Courses GROUP BY Category HAVING AVG(EnroIImentCount) > 700 ORDER BY AvgEnrollment DESC;

**Q.2)Model the following Laptop manufacturing information system as a graph model, and answer the following queries using Cypher**

// Create Customer nodes

CREATE (:Customer {name: 'John'})

CREATE (:Customer {name: 'Alice'})

CREATE (:Customer {name: 'Bob'})

// Create Laptop nodes

CREATE (:Laptop {name: 'DELL XPS 13', company: 'DELL', characteristics: 'Thin and lightweight', purchase\_date: '26/01/2023'})

CREATE (:Laptop {name: 'MacBook Pro', company: 'Apple', characteristics: 'High performance and sleek design', purchase\_date: '27/01/2023'})

CREATE (:Laptop {name: 'HP Pavilion', company: 'HP', characteristics: 'Affordable and reliable', purchase\_date: '28/01/2023'})

// Create relationships between Customers and Laptops

MATCH (c:Customer {name: 'John'}), (l:Laptop {name: 'DELL XPS 13'})

CREATE (c)-[:PURCHASED\_BY]->(l)

MATCH (c:Customer {name: 'Alice'}), (l:Laptop {name: 'MacBook Pro'})

CREATE (c)-[:PURCHASED\_BY]->(l)

CREATE (c)-[:RECOMMENDED\_BY]->(l)

MATCH (c:Customer {name: 'Bob'}), (l:Laptop {name: 'HP Pavilion'})

CREATE (c)-[:PURCHASED\_BY]->(l)

CREATE (c)-[:RATED\_BY]->(l)

**List the characteristics of a specific laptop.** cypher

MATCH (l:Laptop {name: 'DELL XPS 13'}) RETURN l.characteristics

**List the names of customers who bought a “DELL” company laptop.** cypher

MATCH (c:Customer)-[:PURCHASED\_BY]->(l:Laptop) WHERE l.company = 'DELL' RETURN DISTINCT c.name

**List the customers who purchased a device on “26/01/2023”.** cypher

MATCH (c:Customer)-[:PURCHASED\_BY]->(l:Laptop) WHERE l.purchase\_date = '26/01/2023' RETURN DISTINCT c.name

**List the most recommended device.** cypher

MATCH (c:Customer)-[:RECOMMENDED\_BY]->(l:Laptop) RETURN l.name, COUNT(c) AS recommendations ORDER BY recommendations DESC LIMIT 1

Slip-19

**Q.1)Consider the course table**

a) List the categories where the maximum price of courses is less than $50, sorted by category name:

b) Find the instructors who teach courses with the highest average enrollment count, and display the top 3 instructors with the highest average, sorted by average enrollment count in descending order:

CREATE TABLE Courses ( CourselD INT PRIMARY KEY, Title VARCHAR(255),

Instructor VARCHAR(255), Category VARCHAR(50), Price DECIMAL(10, 2),

Duration INT, -- Assuming duration is in weeks EnrollmentCount INT

)

INSERT INTO Courses (CourselD, Title, Instructor, Category, Price, Duration, EnrollmentCount) VALUES

(1, 'Introduction to Programming', 'John Smith', 'Programming', 49.99, 6, 300),

(2, 'Data Structures and Algorithms', 'Jane Doe', 'Programming', 59.99, 10, 400),

(3, 'Introduction to Statistics', 'Alice Johnson', 'Mathematics', 39.99, 8, 250),

(4, 'Calculus I', ’Michael Brown', 'Mathematics', 49.99, 12, 350),

(5, 'English Grammar Basics', 'Emily Wilson', 'Language', 29.99, 4, 200),

(6, 'Advanced French Conversation', 'Sophie Martin', 'Language', 39.99, 6, 280),

(7, 'Introduction to Physics', 'David Clark', 'Science', 44.99, 6, 320),

(8, 'Chemistry Fundamentals', 'Mark Thompson', 'Science', 49.99, 8, 410),

(9, 'Art History: Renaissance', 'Olivia Parker', 'Art', 34.99, 6, 220),

(10, 'Digital Photography Basics', 'Daniel White', 'Art', 39.99, 4, 380);

SELECT Category FROM Courses GROUP BY Category

HAVING MAX(Price) < 50

ORDER BY Category;

SELECT Instructor, AVG(EnroIImentCount) AS AvgEnrollment FROM Courses

GROUP BY Instructor

ORDER BY AvgEnrollment DESC LIMIT 3;

**Q.2) Consider the doctors in and around Pune. Each Doctor is specialized in some stream like Pediatric, Gynaec, Heart Specialist, Cancer Specialist, ENT, etc. A doctor may be a visiting doctor across many hospitals or he may own a clinic.**

// Create Doctor nodes

CREATE (:Doctor {name: 'Dr. Smith'})

CREATE (:Doctor {name: 'Dr. Patel'})

CREATE (:Doctor {name: 'Dr. Gupta'})

// Create Specialization nodes

CREATE (:Specialization {name: 'Orthopedic'})

CREATE (:Specialization {name: 'Pediatrics'})

CREATE (:Specialization {name: 'Cardiologist'})

// Create Hospital nodes

CREATE (:Hospital {name: 'ABC Hospital'})

CREATE (:Hospital {name: 'XYZ Hospital'})

CREATE (:Hospital {name: 'DEF Hospital'})

// Create Clinic nodes

CREATE (:Clinic {name: 'Smith Clinic'})

CREATE (:Clinic {name: 'Patel Clinic'})

CREATE (:Clinic {name: 'Gupta Clinic'})

// Create Person nodes

CREATE (:Person {name: 'John Doe'})

CREATE (:Person {name: 'Alice Smith'})

CREATE (:Person {name: 'Bob Patel'})

// Create relationships

// Doctor specialization

MATCH (d:Doctor {name: 'Dr. Smith'}), (s:Specialization {name: 'Orthopedic'})

CREATE (d)-[:SPECIALIZES\_IN]->(s)

MATCH (d:Doctor {name: 'Dr. Patel'}), (s:Specialization {name: 'Pediatrics'})

CREATE (d)-[:SPECIALIZES\_IN]->(s)

MATCH (d:Doctor {name: 'Dr. Gupta'}), (s:Specialization {name: 'Cardiologist'})

CREATE (d)-[:SPECIALIZES\_IN]->(s)

// Doctor visits hospitals

MATCH (d:Doctor {name: 'Dr. Smith'}), (h:Hospital {name: 'ABC Hospital'})

CREATE (d)-[:VISITS]->(h)

MATCH (d:Doctor {name: 'Dr. Patel'}), (h:Hospital {name: 'XYZ Hospital'})

CREATE (d)-[:VISITS]->(h)

MATCH (d:Doctor {name: 'Dr. Gupta'}), (h:Hospital {name: 'DEF Hospital'})

CREATE (d)-[:VISITS]->(h)

// Doctor owns clinics

MATCH (d:Doctor {name: 'Dr. Smith'}), (c:Clinic {name: 'Smith Clinic'})

CREATE (d)-[:OWNS]->(c)

MATCH (d:Doctor {name: 'Dr. Patel'}), (c:Clinic {name: 'Patel Clinic'})

CREATE (d)-[:OWNS]->(c)

MATCH (d:Doctor {name: 'Dr. Gupta'}), (c:Clinic {name: 'Gupta Clinic'})

CREATE (d)-[:OWNS]->(c)

// Person reviews doctors

MATCH (p:Person {name: 'John Doe'}), (d:Doctor {name: 'Dr. Smith'})

CREATE (p)-[:REVIEWED\_BY]->(d)

MATCH (p:Person {name: 'Alice Smith'}), (d:Doctor {name: 'Dr. Patel'})

CREATE (p)-[:REVIEWED\_BY]->(d)

MATCH (p:Person {name: 'Bob Patel'}), (d:Doctor {name: 'Dr. Gupta'})

CREATE (p)-[:REVIEWED\_BY]->(d)

**List the Orthopedic doctors in a specific area.** cypher

MATCH (d:Doctor)-[:SPECIALIZES\_IN]->(:Specialization {name: 'Orthopedic'})-[:LOCATED\_IN]->(:Area {name: 'Pune'}) RETURN d.name

**List the doctors who have specialization in a specific field.** cypher

MATCH (d:Doctor)-[:SPECIALIZES\_IN]->(:Specialization {name: 'SpecializationName'}) RETURN d.name

**List the most recommended Pediatrics doctor in a specific hospital.** Cypher

MATCH (d:Doctor)-[:SPECIALIZES\_IN]->(:Specialization {name: 'Pediatrics'})-[:VISITS]->(:Hospital {name: 'Seren Medows'}) WITH d, COUNT(\*) AS recommendations ORDER BY recommendations DESC LIMIT 1 RETURN d.name

**List all the doctors who visit more than 2 hospitals.** cypher

MATCH (d:Doctor)-[:VISITS]->(h:Hospital) WITH d, COUNT(DISTINCT h) AS numHospitals WHERE numHospitals > 2 RETURN d.name

Slip -20

**Q.1) Consider the course table**

a) List the categories where the total enrollment count of courses is more than 3000, sorted by total enrollment count in descending order:

b) Find the categories where the total number of courses is equal to the number of distinct instructors, sorted by category name:

CREATE TABLE Courses ( CourselD INT PRIMARY KEY, Title VARCHAR(255),

Instructor VARCHAR(255), Category VARCHAR(50), Price DECIMAL(10, 2),

Duration INT, -- Assuming duration is in weeks EnrollmentCount INT

)

INSERT INTO Courses (CourselD, Title, Instructor, Category, Price, Duration, EnrollmentCount) VALUES

(1, 'Introduction to Programming', 'John Smith', 'Programming', 99.99, 6, 400),

(2, 'Data Structures and Algorithms', 'Jane Doe', 'Programming', 129.99, 10, 600),

(3, 'Introduction to Statistics', 'Alice Johnson', 'Mathematics', 79.99, 8, 700),

(4, 'Calculus I', ’Michael Brown', 'Mathematics', 89.99, 12, 800),

(5, 'English Grammar Basics', 'Emily Wilson', 'Language', 49.99, 4, 500),

(6, 'Advanced French Conversation', 'Sophie Martin', 'Language', 79.99, 6, 400),

(7, 'Introduction to Physics', 'David Clark', 'Science', 69.99, 6, 300),

(8, 'Chemistry Fundamentals', 'Mark Thompson', 'Science', 79.99, 8, 600),

(9, 'Art History: Renaissance', 'Olivia Parker', 'Art', 59.99, 6, 800),

(10, 'Digital Photography Basics', 'Daniel White', 'Art', 69.99, 4, 300);

SELECT Category, SUM(EnroIImentCount) AS TotalEnrollment FROM Courses

GROUP BY Category

HAVING SUM(EnroIImentCount) > 3000 ORDER BY TotalEnrollment DESC;

SELECT Category

FROM (SELECT Category, COUNT(DISTINCT CourselD) AS NumCourses, COUNT(DISTINCT Instructor) AS Numlnstructors FROM Courses GROUP BY Category

) AS CategoryStats WHERE NumCourses = Numlnstructors ORDER BY Category;

**Q.2) Author wrote various types of books which is published by publishers. A reader reads a books according to his linking and can recommend/provide review for it.**

// Create Author nodes

CREATE (:Author {name: 'Author1'})

CREATE (:Author {name: 'Author2'})

CREATE (:Author {name: 'Author3'})

// Create Book nodes

CREATE (:Book {title: 'Book1', genre: 'Comics'})

CREATE (:Book {title: 'Book2', genre: 'Fantasy'})

CREATE (:Book {title: 'Book3', genre: 'Thriller'})

// Create Publisher nodes

CREATE (:Publisher {name: 'Sage'})

CREATE (:Publisher {name: 'NewAge'})

CREATE (:Publisher {name: 'Nova'})

// Create Reader nodes

CREATE (:Reader {name: 'Reader1'})

CREATE (:Reader {name: 'Reader2'})

CREATE (:Reader {name: 'Reader3'})

// Create relationships

// Authors wrote books

MATCH (a:Author {name: 'Author1'}), (b:Book {title: 'Book1'})

CREATE (a)-[:WROTE\_BY]->(b)

MATCH (a:Author {name: 'Author2'}), (b:Book {title: 'Book2'})

CREATE (a)-[:WROTE\_BY]->(b)

MATCH (a:Author {name: 'Author3'}), (b:Book {title: 'Book3'})

CREATE (a)-[:WROTE\_BY]->(b)

// Books published by publishers

MATCH (b:Book {title: 'Book1'}), (p:Publisher {name: 'Sage'})

CREATE (b)-[:PUBLISHED\_BY]->(p)

MATCH (b:Book {title: 'Book2'}), (p:Publisher {name: 'NewAge'})

CREATE (b)-[:PUBLISHED\_BY]->(p)

MATCH (b:Book {title: 'Book3'}), (p:Publisher {name: 'Nova'})

CREATE (b)-[:PUBLISHED\_BY]->(p)

// Readers read books

MATCH (r:Reader {name: 'Reader1'}), (b:Book {title: 'Book1'})

CREATE (r)-[:READ\_BY]->(b)

MATCH (r:Reader {name: 'Reader2'}), (b:Book {title: 'Book2'})

CREATE (r)-[:READ\_BY]->(b)

MATCH (r:Reader {name: 'Reader3'}), (b:Book {title: 'Book3'})

CREATE (r)-[:READ\_BY]->(b)

// Readers recommend books

MATCH (r:Reader {name: 'Reader1'}), (b:Book {title: 'Book1'})

CREATE (r)-[:RECOMMENDED\_BY]->(b)

MATCH (r:Reader {name: 'Reader2'}), (b:Book {title: 'Book2'})

CREATE (r)-[:RECOMMENDED\_BY]->(b)

MATCH (r:Reader {name: 'Reader3'}), (b:Book {title: 'Book3'})

CREATE (r)-[:RECOMMENDED\_BY]->(b)

// Readers rate books

MATCH (r:Reader {name: 'Reader1'}), (b:Book {title: 'Book1'})

CREATE (r)-[:RATED\_BY {rating: 4}]->(b)

MATCH (r:Reader {name: 'Reader2'}), (b:Book {title: 'Book2'})

CREATE (r)-[:RATED\_BY {rating: 5}]->(b)

MATCH (r:Reader {name: 'Reader3'}), (b:Book {title: 'Book3'})

CREATE (r)-[:RATED\_BY {rating: 3}]->(b)

**List the names of authors who wrote “Comics”.** cypher

MATCH (a:Author)-[:WROTE\_BY]->(b:Book) WHERE b.genre = 'Comics' RETURN DISTINCT a.name

**Count the number of readers of a specific book published by “Sage”.** cypher

MATCH (b:Book)-[:PUBLISHED\_BY]->(:Publisher {name: 'Sage'}) RETURN COUNT(DISTINCT (b)<-[:READ\_BY]-(:Reader)) AS readersCount

**List all the publishers whose name starts with “N”.** cypher

MATCH (p:Publisher) WHERE p.name STARTS WITH 'N' RETURN DISTINCT p.name

**List the names of people who have given a rating of (>=3) for a specific book.** cypher MATCH (:Reader)-[rating:RATED\_BY]->(b:Book {title: 'BookTitle'}) WHERE rating.rating >= 3 RETURN DISTINCT rating.name