SLIP 11

Q1. Consider the Book table

Book (book_id,title,author,publication_year,language,available_copies,total_copies)

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

```
CREATE TABLE Book (
   book_id INT PRIMARY KEY,
   title VARCHAR(100),
   author VARCHAR(100),
   publication_year INT,
   language VARCHAR(50),
   available_copies INT,
   total_copies INT
);
```

Adding the values

```
INSERT INTO Book (book_id, title, author, publication_year, language, available_copies, total_copies)
VALUES
   (1, 'Book1', 'Author1', 2005, 'English', 10, 20),
   (2, 'Book2', 'Author2', 2010, 'English', 15, 25),
   (3, 'Book3', 'Author1', 2015, 'English', 20, 30),
   (4, 'Book4', 'Author3', 2015, 'French', 25, 35),
   (5, 'Book5', 'Author2', 2020, 'Spanish', 30, 40);
```

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

```
SELECT publication_year, COUNT(*) AS num_books_published
FROM Book
GROUP BY publication_year
ORDER BY publication_year ASC;
```

2) 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

```
SELECT publication_year, COUNT(*) AS num_books_published
FROM Book
GROUP BY publication_year
ORDER BY publication_year ASC;
```

Q2. **Model the following sales system as a document database. Consider a set of products, customers, orders and invoices. An invoice is generated when an order is processed.**

`Products` collection:

Attributes: `_id`, `name`, `price`, `quantity`

‘Customers` collection:

Attributes: `_id`, `name`, `email`, `address`

`Orders` collection:

Attributes: `_id`, `customer_id`, `product_id`, `quantity`, `order_date`, `total_value`, `processed`

`Invoices` collection:

Attributes: `_id`, `order_id`, `invoice_date`, `amount`

1. **Assume appropriate attributes and collections as per the query requirements.[3]**

Collections:

- Products: Contains information about the products available for sale.
- Customers: Stores details about the customers who purchase products.
- Orders: Records the details of orders placed by customers.
- Invoices: Stores information about invoices generated for processed orders.

Attributes:

- Products:
- _id: Unique identifier for the product.
- name: Name of the product.
- price: Price of the product.
- quantity: Quantity of the product available in inventory.

Customers:

- _id: Unique identifier for the customer.
- name: Name of the customer.
- email: Email address of the customer.
- address: Address of the customer.

Orders:

- _id: Unique identifier for the order.
- customer_id: References the _id of the customer who placed the order.
- product_id: References the _id of the product ordered.
- quantity: Quantity of the product ordered.
- order_date: Date when the order was placed.
- total_value: Total value of the order.
- processed: Indicates whether the order has been processed (true/false).

Invoices:

- _id: Unique identifier for the invoice.
- order_id: References the _id of the order for which the invoice is generated.
- invoice_date: Date when the invoice was generated.
- amount: Amount payable as per the invoice.

2. **Insert at least 5 documents in each collection.**

// Products collection

```
[
  { "_id": 1, "name": "Product1", "price": 100, "quantity": 50 },
```

{ "_id": 2, "name": "Product2", "price": 150, "quantity": 30 },

{ "_id": 3, "name": "Product3", "price": 200, "quantity": 20 },

{ "_id": 4, "name": "Product4", "price": 250, "quantity": 25 },

{ "_id": 5, "name": "Product5", "price": 300, "quantity": 40 }

]

// Customers collection

[

{ "_id": 1, "name": "Mr. Rajiv", "email": "rajiv@example.com", "address": "Address1" },

{ "_id": 2, "name": "Mrs. Smith", "email": "smith@example.com", "address": "Address2" },

{ "_id": 3, "name": "Ms. Patel", "email": "patel@example.com", "address": "Address3" },

{ "_id": 4, "name": "Mr. Kumar", "email": "kumar@example.com", "address": "Address4" },

{ "_id": 5, "name": "Mrs. Gonzalez", "email": "gonzalez@example.com", "address": "Address5" }

]

// Orders collection

[

{ "_id": 1, "customer_id": 1, "product_id": 1, "quantity": 2, "order_date": "2024-03-28", "total_value": 200, "processed": true },

{ "_id": 2, "customer_id": 2, "product_id": 2, "quantity": 3, "order_date": "2024-03-27", "total_value": 450, "processed": false },

{ "_id": 3, "customer_id": 3, "product_id": 3, "quantity": 1, "order_date": "2024-03-26", "total_value": 200, "processed": true },

{ "_id": 4, "customer_id": 4, "product_id": 4, "quantity": 2, "order_date": "2024-03-25", "total_value": 500, "processed": false },

{ "_id": 5, "customer_id": 5, "product_id": 5, "quantity": 4, "order_date": "2024-03-24", "total_value": 1200, "processed": true }

]

// Invoices collection (assuming invoices are generated after order processing)

[

{ "_id": 1, "order_id": 1, "invoice_date": "2024-03-29", "amount": 200 },

{ "_id": 3, "order_id": 3, "invoice_date": "2024-03-29", "amount": 200 },

{ "_id": 5, "order_id": 5, "invoice_date": "2024-03-29", "amount": 1200 }

]

3. **Answer the following Queries.**

a. List all products in the inventory.

db.Products.find({})

A. **List the details of orders with a value >20000.**

```
db.Orders.find({ total_value: { $gt: 20000 } })
```

B. List all the orders which has not been processed (invoice not generated)

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

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

```
db.Orders.aggregate([
 { $match: { customer_id: db.Customers.findOne({ name: "Mr. Rajiv" })._id } },
 {
   $lookup: {
    from: "Invoices",
    localField: "_id",
    foreignField: "order_id",
    as: "invoices"
   }
 }
])
```

SLIP 12

Consider the Worker table

Worker (WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT)

```
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)
);
```

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

```
-- Inserting sample data into the Worker table
INSERT INTO Worker (WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT)
VALUES
   (1, 'John', 'Doe', 50000, '2022-03-15', 'IT'),
   (2, 'Jane', 'Smith', 60000, '2021-05-20', 'HR'),
   (3, 'Michael', 'Johnson', 55000, '2021-07-10', 'Finance'),
```

(4, 'Emily', 'Brown', 65000, '2020-11-30', 'Finance'),

(5, 'David', 'Lee', 70000, '2021-02-28', 'IT');

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

```
SELECT DEPARTMENT

FROM Worker

GROUP BY DEPARTMENT

HAVING COUNT(*) = COUNT(DISTINCT FIRST_NAME)

ORDER BY DEPARTMENT;
```

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

```
SELECT DEPARTMENT,  COUNT(*) AS num_workers

FROM Worker

WHERE JOINING_DATE > '2021-01-01'

GROUP BY DEPARTMENT

ORDER BY DEPARTMENT;
```

Q2. Model the following University information system as a graph model, and answer the following queries using Cypher.

University has various departments like Physics, Geography, and Computer etc. Each department conducts various courses and a course may be conducted by multiple departments. Every course may have recommendations provided by people.

1. **Identify the labels and relationships, along with their properties, And draw a high-level Graph model.**

- **Labels**:
  - `University`
  - `Department`
  - `Course`
  - `Person`
  - `Recommendation`
- **Relationships**:
  - `DEPARTMENT_OF`: Connects a department to the university.
  - `CONDUCTS`: Connects a department to a course.
  - `RECOMMENDS`: Connects a person to a course.
  - `COMMON_COURSE`: Connects courses across different departments.

2. Create nodes and relationships, along with their properties, and visualize your actual Graph model.

```
(// Create University node

CREATE (:University {name: 'University of Example'});


// Create Department nodes and relationships
```

```
CREATE (:Department {name: 'Physics'}),
    (:Department {name: 'Computer Science'}),
    (:Department {name: 'Mathematics'});

// Create Course nodes and relationships
CREATE (:Course {name: 'Physics Course1'}),
    (:Course {name: 'Computer Science Course1'}),
    (:Course {name: 'Mathematics Course1'});

// Create Person nodes
CREATE (:Person {name: 'John Doe'}),
    (:Person {name: 'Jane Smith'});

// Create relationships
MATCH (u:University), (d:Department {name: 'Physics'})
CREATE (u)-[:DEPARTMENT_OF]->(d);

MATCH (d:Department {name: 'Physics'}), (c:Course {name: 'Physics Course1'})
CREATE (d)-[:CONDUCTS]->(c);

MATCH (c1:Course {name: 'Physics Course1'}), (p:Person {name: 'John Doe'})
CREATE (c1)-[:RECOMMENDS]->(p);

MATCH (c1:Course {name: 'Physics Course1'}), (c2:Course {name: 'Computer Science Course1'})
CREATE (c1)<-[:COMMON_COURSE]->(c2);
```

3. Answer the following Queries:

A. List the details of all the departments in the university.

```
MATCH (d:Department)
RETURN d;
```

B. List the names of the courses provided by Physics department.

```
MATCH (d:Department {name: 'Physics'})-[:CONDUCTS]->(c:Course)
RETURN c.name;
```

C. List the most recommended course in Geography department.

```
MATCH (d:Department {name: 'Geography'})-[:CONDUCTS]->(c:Course)<-[:RECOMMENDS]-(p:Person)
WITH c, COUNT(p) AS num_recommendations
```

```
RETURN c.name
ORDER BY num_recommendations DESC
LIMIT 1;
```

D. List the names of common courses across Mathematics and computer department.

```
MATCH (math:Department {name: 'Mathematics'})-[:CONDUCTS]->(commonCourse:Course)<-[:CONDUCTS]-(comp:Department {name: 'Computer'})
RETURN commonCourse.name;
```

SLIP 13

Consider the Book table

Book (book_id, title, author, publication_year , language ,available_copies, total_copies)

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

```
-- Inserting sample data into the book table
INSERT INTO book (book_id, title, author, publication_year, language, available_copies, total_copies)
VALUES
  (1, 'Book1', 'Author1', 2005, 'English', 3, 10),
  (2, 'Book2', 'Author2', 2010, 'English', 5, 20),
  (3, 'Book3', 'Author1', 2015, 'French', 2, 15),
  (4, 'Book4', 'Author3', 2015, 'German', 4, 25),
  (5, 'Book5', 'Author2', 2020, 'Spanish', 1, 30);
```

1) **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:**

```
SELECT author, COUNT(DISTINCT language) AS num_languages
FROM book
GROUP BY author
HAVING COUNT(DISTINCT language) > 1
ORDER BY author;
```

2) **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**

```
SELECT title, AVG(available_copies) AS avg_available_copies
FROM book
GROUP BY title
HAVING AVG(available_copies) < 5
ORDER BY avg_available_copies DESC;
```

Q2. Model the following Library information system as a graph model, and answer the following queries using Cypher.

Consider a library information system having different types of books like text, reference, bibliography etc. A student can buy one or more types of book. A student can recommend or rate a book according to its type.

1. **Identify the labels and relationships, along with their properties, and draw a high-level Graph model.**

- **Labels**:
  - `Book`
  - `Student`
- **Relationships**:
  - `BOUGHT`: Connects a student to a book, indicating that the student bought the book.
  - `RECOMMENDED`: Connects a student to a book, indicating that the student recommended or rated the book.
  - `OF_TYPE`: Connects a book to its type (e.g., "text", "reference", "bibliography").

2. Create nodes and relationships, along with their properties, and visualize your actual Graph model.

```
// Create students
CREATE (Alice:Student {name: 'Alice'}),
    (Bob:Student {name: 'Bob'});


// Create books and their types
CREATE (Book1:Book {title: 'Book1', type: 'text'}),
    (Book2:Book {title: 'Book2', type: 'reference'}),
    (Book3:Book {title: 'Book3', type: 'bibliography'}),
    (Book4:Book {title: 'Book4', type: 'text'}),
    (Book5:Book {title: 'Book5', type: 'reference'});


// Create book types
CREATE (text:BookType {name: 'text'}),
    (reference:BookType {name: 'reference'}),
    (bibliography:BookType {name: 'bibliography'});


// Create relationships
// Alice bought Book1 (text) and Book2 (reference)
MATCH (alice:Student {name: 'Alice'}), (book1:Book {title: 'Book1'}),
   (book2:Book {title: 'Book2'})
CREATE (alice)-[:BOUGHT]->(book1),
```

```
    (alice)-[:BOUGHT]->(book2);

// Set relationships between books and their types
MATCH (book1:Book {title: 'Book1'}), (book2:Book {title: 'Book2'}),
    (book1)-[:OF_TYPE]->(text:BookType {name: 'text'}),
    (book2)-[:OF_TYPE]->(reference:BookType {name: 'reference'});
```

3. Answer the following Queries:

A. List the books of type "text"

```
MATCH (b:Book)-[:OF_TYPE]->(t:Type {name: 'text'})
RETURN b;
```

B. List the name of student who bought a text and reference types books.

```
MATCH (s:Student)-[:BOUGHT]->(:Book)-[:OF_TYPE]->(t:Type)
WHERE t.name IN ['text', 'reference']
RETURN DISTINCT s.name;
```

C. List the most recommended book type.

```
MATCH (:Student)-[:RECOMMENDED]->(b:Book)-[:OF_TYPE]->(t:Type)
RETURN t.name, COUNT(*) AS recommendations
ORDER BY recommendations DESC
LIMIT 1;
```

D. List the student who buy the more than one type of book

```
MATCH (s:Student)-[:BOUGHT]->(:Book)-[:OF_TYPE]->(t:Type)
WITH s, COUNT(DISTINCT t) AS num_types
WHERE num_types > 1
RETURN s.name;
```

SLIP 14

Q1. Consider the Book table

Book (book_id, title, author, publication_year, language, available_copies, total_copies)

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

```
-- Inserting sample data into the book table
INSERT INTO book (book_id, title, author, publication_year, language, available_copies, total_copies)
VALUES
   (1, 'Book1', 'Author1', 2005, 'English', 20, 50),
   (2, 'Book2', 'Author2', 2010, 'English', 25, 60),
```

(3, 'Book3', 'Author1', 2015, 'French', 15, 40),

(4, 'Book4', 'Author3', 2015, 'German', 30, 70),

(5, 'Book5', 'Author2', 2020, 'Spanish', 10, 35);

1) **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**

```
SELECT language, COUNT(*) AS total_books

FROM book

GROUP BY language

ORDER BY total_books DESC;
```

2) **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**

```
SELECT language, COUNT(*) AS total_books

FROM book

GROUP BY language

ORDER BY total_books DESC;
```

Q2. Model the following Automobile information system as a graph model, and answer the following queries using Cypher.

Consider an Automobile industry manufacturing different types of vehicles like Two- Wheeler, Four-Wheeler, etc. A customer can buy one or more types of vehicle. A person can recommend or rate a vehicle type.

1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model.

1. **VehicleType:**
   - Properties:
     - type: Type of the vehicle (e.g., Two-Wheeler, Four-Wheeler)
2. **Customer:**
   - Properties:
     - name: Name of the customer
3. **Person:**
   - Properties:
     - name: Name of the person

Relationships:

1. **PURCHASED:**
   - Connects a Customer to a VehicleType, indicating that the customer bought the vehicle type.
2. **RECOMMENDED:**
   - Connects a Person to a VehicleType, indicating that the person recommended or rated the vehicle type.

2. Create nodes and relationships, along with their properties, and visualize your actual Graph model.

```
// Create Vehicle Type nodes
```

```
CREATE (:VehicleType {type: 'Two-Wheeler'}),
      (:VehicleType {type: 'Four-Wheeler'});

// Create Customer nodes
CREATE (:Customer {name: 'Customer1'}),
      (:Customer {name: 'Customer2'});

// Create Person node
CREATE (:Person {name: 'Person1'});

// Create relationships
MATCH (c1:Customer {name: 'Customer1'}), (p:Person {name: 'Person1'}),
     (c2:Customer {name: 'Customer2'})
CREATE (c1)-[:PURCHASED]->(:VehicleType {type: 'Two-Wheeler'}),
      (p)-[:RECOMMENDED]->(:VehicleType {type: 'Four-Wheeler'}),
      (c2)-[:PURCHASED]->(:VehicleType {type: 'Four-Wheeler'});
```

3. Answer the following Queries:

A. List the characteristics of four wheeler types.

```
// A. List the characteristics of four-wheeler types
MATCH (v:VehicleType {type: 'Four-Wheeler'})
RETURN v;
```

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

```
// B. List the name of customers who bought a two-wheeler vehicle
MATCH (c:Customer)-[:PURCHASED]->(:VehicleType {type: 'Two-Wheeler'})
RETURN DISTINCT c.name;
```

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

```
// C. List the customers who bought more than one type of vehicle
MATCH (c:Customer)-[:PURCHASED]->(:VehicleType)-[:PURCHASED]->(v:VehicleType)
WITH c, COUNT(DISTINCT v) AS num_types
WHERE num_types > 1
RETURN c.name;
```

D. List the most recommended vehicle type.

```
// D. List the most recommended vehicle type
MATCH (:Person)-[:RECOMMENDED]->(v:VehicleType)
```

```
RETURN v.type, COUNT(*) AS recommendations
ORDER BY recommendations DESC
LIMIT 1;
```

SLIP 15

Q1. Consider the Book table

Book (book_id, title, author, publication_year, language, available_copies, total_copies)

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

```
-- Inserting sample data into the Book table
INSERT INTO Book (book_id, title, author, publication_year, language, available_copies, total_copies)
VALUES
  (1, 'Book1', 'Author1', 2005, 'English', 10, 20),
  (2, 'Book2', 'Author2', 2010, 'English', 15, 25),
  (3, 'Book3', 'Author1', 2015, 'French', 5, 15),
  (4, 'Book4', 'Author3', 2015, 'German', 20, 30),
  (5, 'Book5', 'Author2', 2020, 'Spanish', 8, 12);
```

1) 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

```
SELECT title, available_copies
FROM Book
WHERE available_copies < total_copies
ORDER BY available_copies ASC;
```

2) 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

```
SELECT author, (total_copies - available_copies) AS difference
FROM Book
GROUP BY author
ORDER BY difference DESC
LIMIT 3;
```

Q2. Model the following Car Showroom information as a graph model, and answer the queries using Cypher. Consider a car showroom with different models of cars like sofas Honda city, Skoda, Creta, Swift, Ertiga etc.

Showroom is divided into different sections, one section for each car model; each section is handled by a sales staff. A sales staff can handle one or more sections. Customer may enquire about car. An enquiry may result in a purchase by the customer.

1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model.

Labels:

1. **CarModel:**
   - Properties:
     - name: Name of the car model (e.g., Honda City, Skoda, Creta, Swift, Ertiga)
2. **Section:**
   - Properties:
     - name: Name of the section (e.g., Honda City Section, Skoda Section, Creta Section, Swift Section, Ertiga Section)
3. **SalesStaff:**
   - Properties:
     - name: Name of the sales staff
4. **Customer:**
   - Properties:
     - name: Name of the customer

Relationships:

1. **HANDLES:**
   - Connects a SalesStaff to a Section, indicating that the sales staff handles that section.
2. **ENQUIRED:**
   - Connects a Customer to a CarModel, indicating that the customer enquired about that car model.
3. **PURCHASED:**
   - Connects a Customer to a CarModel, indicating that the customer purchased that car model.

2. Create nodes and relationships, along with their properties, and visualize your actual Graph model.

```
// Create CarModel nodes
CREATE (:CarModel {name: 'Honda City'}),
    (:CarModel {name: 'Skoda'}),
    (:CarModel {name: 'Creta'}),
    (:CarModel {name: 'Swift'}),
    (:CarModel {name: 'Ertiga'});

// Create Section nodes
CREATE (:Section {name: 'Honda City Section'}),
    (:Section {name: 'Skoda Section'}),
    (:Section {name: 'Creta Section'}),
    (:Section {name: 'Swift Section'}),
    (:Section {name: 'Ertiga Section'});

// Create SalesStaff nodes
```

```
CREATE (:SalesStaff {name: 'Mr. Narayan'}),
       (:SalesStaff {name: 'Mr. Kumar'}),
       (:SalesStaff {name: 'Ms. Singh'});

// Create Customer nodes
CREATE (:Customer {name: 'Customer1'}),
       (:Customer {name: 'Customer2'}),
       (:Customer {name: 'Customer3'});

// Establish relationships
MATCH (staff:SalesStaff {name: 'Mr. Narayan'}),
      (section:Section {name: 'Honda City Section'}),
      (model:CarModel {name: 'Honda City'})
CREATE (staff)-[:HANDLES]->(section),
       (:Customer {name: 'Customer1'})-[:ENQUIRED]->(model);

MATCH (customer:Customer {name: 'Customer2'}),
      (model:CarModel {name: 'Swift'})
CREATE (customer)-[:ENQUIRED]->(model);

MATCH (customer:Customer {name: 'Customer3'}),
      (model:CarModel {name: 'Creta'})
CREATE (customer)-[:ENQUIRED]->(model);

MATCH (customer:Customer {name: 'Customer3'}),
      (model:CarModel {name: 'Creta'})
CREATE (customer)-[:PURCHASED]->(model);
```

3. Answer the following queries:

A. List the types of cars available in the showroom.

```
MATCH (car:CarModel)
RETURN DISTINCT car.name AS CarModel;
```

B. List the sections handled by Mr. Narayan.

```
MATCH (staff:SalesStaff {name: 'Mr. Narayan'})-[:HANDLES]->(section:Section)
RETURN section.name AS SectionHandled;
```

C. List the names of customers who have done only enquiry but not made any purchase.

```
MATCH (customer:Customer)-[:ENQUIRED]->(car:CarModel)
WHERE NOT (customer)-[:PURCHASED]->(car)
RETURN DISTINCT customer.name AS CustomerName;
```

D. List the highly sale car model.

```
MATCH (car:CarModel)<-[p:PURCHASED]-()
RETURN car.name AS CarModel, COUNT(p) AS TotalSales
ORDER BY TotalSales DESC
LIMIT 1;
```

SLIP 16

Q1. Consider the Course table

Courses (CourseID, Title, Instructor, Category, Price, Duration, Enrollment_Count)

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

```
-- Inserting sample data into the Courses table
INSERT INTO Courses (CourseID, Title, Instructor, Category, Price, Duration, Enrollment_Count)
VALUES
  (1, 'Course 1', 'Instructor A', 'Programming', 50, '3 hours', 100),
  (2, 'Course 2', 'Instructor B', 'Programming', 60, '4 hours', 120),
  (3, 'Course 3', 'Instructor A', 'Mathematics', 40, '2 hours', 80),
  (4, 'Course 4', 'Instructor C', 'Mathematics', 55, '3.5 hours', 90),
  (5, 'Course 5', 'Instructor D', 'Design', 70, '5 hours', 150);
```

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

```
SELECT Category, COUNT(*) AS CourseCount
FROM Courses
GROUP BY Category
ORDER BY Category;
```

2) 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:

```
SELECT Instructor, COUNT(*) AS TotalCoursesTaught
FROM Courses
GROUP BY Instructor
HAVING COUNT(*) > 1
ORDER BY TotalCoursesTaught DESC;
```

Q2. Model the following Medical information as a graph model, and answer the following queries using Cypher.

There are various brands of medicine like Dr. Reddy, Cipla, SunPharma etc. Their uses vary across different states in India. The uses of medicine is measured as %, with a high use defined as >=90%, Medium Use between 50 to 90%, and Low Use <50%. Each medicine manufactures various types of medicine products like Tablet, Syrup, and Powder etc.

1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model.

Labels:

1. **MedicineBrand:**
   - Properties:
     - name: Name of the medicine brand (e.g., Dr. Reddy, Cipla, SunPharma)
2. **State:**
   - Properties:
     - name: Name of the state in India
3. **Medicine:**
   - Properties:
     - name: Name of the medicine
4. **MedicineType:**
   - Properties:
     - type: Type of medicine product (e.g., Tablet, Syrup, Powder)

Relationships:

1. **MANUFACTURES:**
   - Connects a MedicineBrand to a Medicine, indicating that the brand manufactures that medicine.
2. **USES:**
   - Connects a Medicine to a State, indicating the usage of the medicine in that state.

2. Create nodes and relationships, along with their properties, and visualize your actual Graph model.

```
// Create MedicineBrand nodes
CREATE (:MedicineBrand {name: 'Dr. Reddy'}),
    (:MedicineBrand {name: 'Cipla'}),
    (:MedicineBrand {name: 'SunPharma'});

// Create State nodes
CREATE (:State {name: 'Rajasthan'}),
    (:State {name: 'Gujarat'}),
    (:State {name: 'Maharashtra'});

// Create Medicine nodes
CREATE (:Medicine {name: 'Medicine1'}),
    (:Medicine {name: 'Medicine2'}),
```

```
    (:Medicine {name: 'Medicine3'});

// Create MedicineType nodes
CREATE (:MedicineType {type: 'Tablet'}),
    (:MedicineType {type: 'Syrup'}),
    (:MedicineType {type: 'Powder'});

// Establish relationships
MATCH (brand:MedicineBrand {name: 'Dr. Reddy'}),
    (medicine:Medicine {name: 'Medicine1'})
CREATE (brand)-[:MANUFACTURES]->(medicine);

MATCH (brand:MedicineBrand {name: 'Cipla'}),
    (medicine:Medicine {name: 'Medicine2'})
CREATE (brand)-[:MANUFACTURES]->(medicine);

MATCH (brand:MedicineBrand {name: 'SunPharma'}),
    (medicine:Medicine {name: 'Medicine3'})
CREATE (brand)-[:MANUFACTURES]->(medicine);

MATCH (state:State {name: 'Rajasthan'}),
    (medicine:Medicine {name: 'Medicine1'})
CREATE (medicine)-[:USES {percentage: 'High'}]->(state);

MATCH (state:State {name: 'Gujarat'}),
    (medicine:Medicine {name: 'Medicine2'})
CREATE (medicine)-[:USES {percentage: 'Medium'}]->(state);

MATCH (state:State {name: 'Gujarat'}),
    (medicine:Medicine {name: 'Medicine1'})
CREATE (medicine)-[:USES {percentage: 'Low'}]->(state);

MATCH (medicine:Medicine {name: 'Medicine3'}),
    (type:MedicineType {type: 'Powder'})
```

```
CREATE (medicine)-[:BELONGS_TO]->(type);
```

3. Answer the following queries using Cypher:

A. List the names of different medicines considered in your graph

```
MATCH (m:Medicine)
RETURN DISTINCT m.name AS MedicineName;
```

B. List the medicine that are highly Used in Rajasthan.

```
MATCH (m:Medicine)-[u:USES]->(s:State {name: 'Rajasthan'})
WHERE u.percentage = 'High'
RETURN DISTINCT m.name AS MedicineName;
```

C. List the highly used tablet in Gujarat.

```
MATCH (m:Medicine)-[u:USES]->(s:State {name: 'Gujarat'})
WHERE u.percentage = 'High'
AND (m)-[:BELONGS_TO]->(:MedicineType {type: 'Tablet'})
RETURN DISTINCT m.name AS MedicineName;
```

D. List the medicine names manufacturing “Powder”

```
MATCH (m:Medicine)-[:BELONGS_TO]->(t:MedicineType {type: 'Powder'})
RETURN DISTINCT m.name AS MedicineName;
```

SLIP 17

Q1. Consider the Course table

Courses (CourseID,Title,Instructor,Category,Price,Duration,EnrollmentCount)

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

```
-- Inserting sample data into the Courses table
INSERT INTO Courses (CourseID, Title, Instructor, Category, Price, Duration, EnrollmentCount)
VALUES
  (1, 'Course 1', 'Instructor A', 'Programming', 50, '3 hours', 100),
  (2, 'Course 2', 'Instructor B', 'Programming', 60, '4 hours', 120),
  (3, 'Course 3', 'Instructor A', 'Mathematics', 40, '2 hours', 80),
  (4, 'Course 4', 'Instructor C', 'Mathematics', 55, '3.5 hours', 90),
  (5, 'Course 5', 'Instructor D', 'Design', 70, '5 hours', 150);
```

1) 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:

```
SELECT Category, COUNT(*) AS CourseCount
FROM Courses
GROUP BY Category
```

HAVING COUNT(*) > 5

ORDER BY CourseCount DESC;

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

SELECT Category, AVG(Duration) AS AvgDuration

FROM Courses

GROUP BY Category

HAVING AVG(Duration) < '8 hours'

ORDER BY Category;

Q2. Model the following nursery management information as a graph model, and answer the following queries using Cypher.

Nursery content various types of plants, fertilizers and required products. Customer visit the nursery or use an app , purchase the plants and necessary products also rate and recommend the app

1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model.

Labels:

1. **Plant:**
   - Properties:
     - name: Name of the plant
     - type: Type of plant (e.g., flowering, succulent, tree)
     - price: Price of the plant
2. **Fertilizer:**
   - Properties:
     - name: Name of the fertilizer
     - type: Type of fertilizer (e.g., organic, chemical)
     - price: Price of the fertilizer
3. **Product:**
   - Properties:
     - name: Name of the product (e.g., pot, gardening tools)
     - type: Type of product (e.g., accessory, tool)
     - price: Price of the product
4. **Customer:**
   - Properties:
     - name: Name of the customer
     - email: Email of the customer
5. **App:**
   - Properties:
     - name: Name of the app

Relationships:

1. **PURCHASES:**
   - Connects a Customer to a Plant, Fertilizer, or Product, indicating a purchase made by the customer.
   - Properties:
     - date: Date of the purchase

- quantity: Quantity purchased

2. **RECOMMENDS:**
   - Connects a Customer to an App, indicating a recommendation made by the customer.
   - Properties:
     - rating: Rating given by the customer

2. Create nodes and relationships, along with their properties, and visualize your actual Graph model.

```
// Create Plant nodes
CREATE (:Plant {name: 'Rose', type: 'Flowering', price: 10}),
    (:Plant {name: 'Succulent', type: 'Succulent', price: 15}),
    (:Plant {name: 'Maple', type: 'Tree', price: 20});

// Create Fertilizer nodes
CREATE (:Fertilizer {name: 'Organic Fertilizer', type: 'Organic', price: 5}),
    (:Fertilizer {name: 'Chemical Fertilizer', type: 'Chemical', price: 8});

// Create Product nodes
CREATE (:Product {name: 'Pot', type: 'Accessory', price: 10}),
    (:Product {name: 'Gardening Tools', type: 'Tool', price: 15});

// Create Customer nodes
CREATE (:Customer {name: 'John', email: 'john@example.com'}),
    (:Customer {name: 'Alice', email: 'alice@example.com'});

// Create App node
CREATE (:App {name: 'Nursery Management App'});

// Establish relationships
MATCH (customer:Customer {name: 'John'}),
   (plant:Plant {name: 'Rose'})
CREATE (customer)-[:PURCHASES {date: date('2024-04-01'), quantity: 2}]->(plant);

MATCH (customer:Customer {name: 'Alice'}),
   (fertilizer:Fertilizer {name: 'Organic Fertilizer'})
CREATE (customer)-[:PURCHASES {date: date('2024-04-02'), quantity: 1}]->(fertilizer);
```

```
MATCH (customer:Customer {name: 'John'}),
    (product:Product {name: 'Pot'})
CREATE (customer)-[:PURCHASES {date: date('2024-04-03'), quantity: 3}]->(product);

MATCH (customer:Customer {name: 'Alice'}),
    (app:App {name: 'Nursery Management App'})
CREATE (customer)-[:RECOMMENDS {rating: 5}]->(app);
```

3. Answer the following queries using Cypher:

A. List the types of plants from your graph model

```
MATCH (p:Plant)
RETURN DISTINCT p.type AS PlantType;
```

B. List the popular flowering plants.

```
MATCH (p:Plant {type: 'Flowering'})<-[:PURCHASES]-()
RETURN p.name AS FloweringPlant, COUNT(*) AS SalesCount
ORDER BY SalesCount DESC;
```

C. List the names of plants sold plant where qty >500 in last 2 days

```
MATCH (p:Plant)<-[purchase:PURCHASES]-()
WHERE purchase.quantity > 500 AND purchase.date >= date()-2
RETURN DISTINCT p.name AS PlantName;
```

D. List the names of suppliers in decreasing order who supplies “Creepers”.

```
MATCH (p:Plant {name: 'Creepers'})<-[purchase:PURCHASES]-(:Customer)-[:PURCHASES]->(supplier)
RETURN DISTINCT supplier.name AS SupplierName, COUNT(*) AS SalesCount
ORDER BY SalesCount DESC;
```

SLIP 18

Q1. Consider the Course table

Courses (CourseID,Title,Instructor,Category,Price,Duration,EnrollmentCount)

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

```
-- Inserting sample data into the Courses table
INSERT INTO Courses (CourseID, Title, Instructor, Category, Price, Duration, EnrollmentCount)
VALUES
   (1, 'Course 1', 'Instructor A', 'Programming', 50, '3 hours', 1000),
   (2, 'Course 2', 'Instructor B', 'Programming', 60, '4 hours', 800),
   (3, 'Course 3', 'Instructor A', 'Mathematics', 40, '2 hours', 600),
```

(4, 'Course 4', 'Instructor C', 'Mathematics', 55, '3.5 hours', 1200),

(5, 'Course 5', 'Instructor D', 'Design', 70, '5 hours', 400);

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

```
SELECT Instructor, COUNT(*) AS TotalCourses

FROM Courses

GROUP BY Instructor

HAVING SUM(EnrollmentCount) > 500

ORDER BY TotalCourses DESC;
```

2)Find the categories where the average enrollment count of courses is greaterthan 700, sorted by average enrollment count in descending order:

```
SELECT Category, AVG(EnrollmentCount) AS AvgEnrollmentCount

FROM Courses

GROUP BY Category

HAVING AVG(EnrollmentCount) > 700

ORDER BY AvgEnrollmentCount DESC;
```

Q2. Model the following Laptop manufacturing information system as a graph model, and answer the following queries using Cypher. Consider an Laptop manufacturing industries which produces different types of laptops.

A customer can buy a laptop, recommend or rate the product.

1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model.

Labels:

1. **Laptop:**
   - Properties:
     - brand: Brand of the laptop (e.g., DELL, HP, Lenovo)
     - model: Model of the laptop
     - price: Price of the laptop
2. **Customer:**
   - Properties:
     - name: Name of the customer

Relationships:

1. **PURCHASES:**
   - Connects a Customer to a Laptop, indicating a purchase made by the customer.
   - Properties:
     - date: Date of the purchase
2. **RECOMMENDS:**
   - Connects a Customer to a Laptop, indicating a recommendation made by the customer.
   - Properties:
     - rating: Rating given by the customer

2. Create nodes and relationships, along with their properties, and visualize your actual Graph model.

```
// Create Customer nodes
CREATE (:Customer {name: 'John'}),
      (:Customer {name: 'Alice'}),
      (:Customer {name: 'Bob'});

// Create Laptop nodes
CREATE (:Laptop {brand: 'DELL', model: 'Inspiron', price: 800}),
      (:Laptop {brand: 'HP', model: 'Pavilion', price: 900}),
      (:Laptop {brand: 'Lenovo', model: 'IdeaPad', price: 700});

// Establish relationships
MATCH (customer:Customer {name: 'John'}),
     (laptop:Laptop {brand: 'DELL'})
CREATE (customer)-[:PURCHASES {date: date('2023-01-26')}]->(laptop);

MATCH (customer:Customer {name: 'Alice'}),
     (laptop:Laptop {brand: 'HP'})
CREATE (customer)-[:PURCHASES {date: date('2023-02-15')}]->(laptop);

MATCH (customer:Customer {name: 'Bob'}),
     (laptop:Laptop {brand: 'Lenovo'})
CREATE (customer)-[:PURCHASES {date: date('2023-03-10')}]->(laptop);

MATCH (customer:Customer {name: 'Alice'}),
     (laptop:Laptop {brand: 'DELL'})
CREATE (customer)-[:RECOMMENDS {rating: 5}]->(laptop);

MATCH (customer:Customer {name: 'Bob'}),
     (laptop:Laptop {brand: 'DELL'})
CREATE (customer)-[:RECOMMENDS {rating: 4}]->(laptop);
```

3. Answer the Queries

A. List the characteristics of…….. laptop

```
MATCH (l:Laptop)
RETURN l.brand AS Brand, l.model AS Model, l.price AS Price;
```

B. List the name of customers who bought a “DELL” company laptop

```
MATCH (c:Customer)-[:PURCHASES]->(l:Laptop {brand: 'DELL'})
RETURN DISTINCT c.name AS CustomerName;
```

C. List the customers who purchase a device on “26/01/2023”

```
MATCH (c:Customer)-[purchase:PURCHASES]->(l:Laptop)
WHERE purchase.date = date('2023-01-26')
RETURN DISTINCT c.name AS CustomerName;
```

D. List the most recommended device.

```
MATCH (l:Laptop)<-[recommends:RECOMMENDS]-()
WITH l, COUNT(recommends) AS RecommendationCount
ORDER BY RecommendationCount DESC
LIMIT 1
RETURN l.brand AS Brand, l.model AS Model;
```

SLIP 19

Q1. Consider the Course table

Courses(CourseID,Title,Instructor,Category,Price,Duration,EnrollmentCount)

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

```
-- Inserting sample data into the Courses table
INSERT INTO Courses (CourseID, Title, Instructor, Category, Price, Duration, EnrollmentCount)
VALUES
  (1, 'Course 1', 'Instructor A', 'Programming', 40, '3 hours', 100),
  (2, 'Course 2', 'Instructor B', 'Programming', 45, '4 hours', 120),
  (3, 'Course 3', 'Instructor A', 'Mathematics', 30, '2 hours', 80),
  (4, 'Course 4', 'Instructor C', 'Mathematics', 35, '3.5 hours', 90),
  (5, 'Course 5', 'Instructor D', 'Design', 50, '5 hours', 150);
```

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

```
SELECT Category
FROM Courses
GROUP BY Category
HAVING MAX(Price) < 50
ORDER BY Category;
```

2)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:

```
SELECT Instructor, AVG(EnrollmentCount) AS AvgEnrollment
FROM Courses
GROUP BY Instructor
ORDER BY AvgEnrollment DESC
LIMIT 3;
```

Q2. Model the following Doctor's information system as a graph model, and answer the following queries using Cypher. 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. A person can provide a review/can recommend a doctor.

1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model. [3]

Labels:

1. **Doctor:**
   - Properties:
     - name: Name of the doctor
     - specialization: Specialization of the doctor
     - area: Area where the doctor practices
2. **Hospital:**
   - Properties:
     - name: Name of the hospital
     - location: Location of the hospital
3. **Clinic:**
   - Properties:
     - name: Name of the clinic
     - location: Location of the clinic
4. **Person:**
   - Properties:
     - name: Name of the person

Relationships:

1. **PRACTICES_AT:**
   - Connects a Doctor to a Hospital or Clinic where the doctor practices.
   - Properties:
     - visiting: Indicates whether the doctor is a visiting doctor or owns the clinic
2. **RECOMMENDS:**
   - Connects a Person to a Doctor, indicating a recommendation or review provided by the person.
   - Properties:
     - rating: Rating provided by the person

2. Create nodes and relationships, along with their properties, and visualize your actual Graph model. [3]

```
// Create Doctor nodes
CREATE (:Doctor {name: 'Dr. John', specialization: 'Orthopedic', area: 'Pune'}),
    (:Doctor {name: 'Dr. Alice', specialization: 'Pediatrics', area: 'Seren Medows'}),
    (:Doctor {name: 'Dr. Bob', specialization: 'ENT', area: 'Pune'});

// Create Hospital nodes
CREATE (:Hospital {name: 'Hospital A', location: 'Pune'}),
    (:Hospital {name: 'Hospital B', location: 'Seren Medows'}),
    (:Hospital {name: 'Hospital C', location: 'Pune'});

// Create Clinic nodes
CREATE (:Clinic {name: 'Clinic X', location: 'Pune'}),
    (:Clinic {name: 'Clinic Y', location: 'Seren Medows'}),
    (:Clinic {name: 'Clinic Z', location: 'Pune'});

// Create Person nodes
CREATE (:Person {name: 'Person A'}),
    (:Person {name: 'Person B'}),
    (:Person {name: 'Person C'});

// Establish relationships
MATCH (doctor:Doctor {name: 'Dr. John'}), (hospital:Hospital {name: 'Hospital A'})
CREATE (doctor)-[:PRACTICES_AT {visiting: true}]->(hospital);

MATCH (doctor:Doctor {name: 'Dr. Alice'}), (hospital:Hospital {name: 'Hospital B'})
CREATE (doctor)-[:PRACTICES_AT {visiting: true}]->(hospital);

MATCH (doctor:Doctor {name: 'Dr. Bob'}), (clinic:Clinic {name: 'Clinic Z'})
CREATE (doctor)-[:PRACTICES_AT {visiting: false}]->(clinic);

MATCH (person:Person {name: 'Person A'}), (doctor:Doctor {name: 'Dr. John'})
CREATE (person)-[:RECOMMENDS {rating: 5}]->(doctor);

MATCH (person:Person {name: 'Person B'}), (doctor:Doctor {name: 'Dr. Alice'})
```

```
CREATE (person)-[:RECOMMENDS {rating: 4}]->(doctor);


MATCH (person:Person {name: 'Person C'}), (doctor:Doctor {name: 'Dr. Bob'})
CREATE (person)-[:RECOMMENDS {rating: 3}]->(doctor);
```

3. Answer the Queries

A. a. List the Orthopedic doctors in ……. Area. [3]

```
MATCH (d:Doctor {specialization: 'Orthopedic', area: 'Pune'})
RETURN d.name AS DoctorName;
```

B. b. List the doctors who has specialization in ____ [3]

```
MATCH (d:Doctor {specialization: 'Specialization_Name'})
RETURN d.name AS DoctorName;
```

C. c. List the most recommended Pediatrics in Seren Medows. [4]

```
MATCH (:Doctor {specialization: 'Pediatrics', area: 'Seren Medows'})<-[:RECOMMENDS]-(p:Person)
WITH p, COUNT(*) AS recommendations
ORDER BY recommendations DESC
LIMIT 1
RETURN p.name AS DoctorName;
```

D. d. List all the who visits more than 2 hospitals [4]

```
MATCH (d:Doctor)-[r:PRACTICES_AT]->(h:Hospital)
WITH d, COUNT(DISTINCT h) AS hospitalCount
WHERE hospitalCount > 2
RETURN d.name AS DoctorName, hospitalCount;
```

SLIP 20

Q1. Consider the Course table

Courses(CourseID,Title,Instructor,Category,Price,Duration,EnrollmentCount)

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

```
-- Inserting sample data into the Courses table
INSERT INTO Courses (CourseID, Title, Instructor, Category, Price, Duration, EnrollmentCount)
VALUES
  (1, 'Course 1', 'Instructor A', 'Programming', 50, '4 hours', 800),
  (2, 'Course 2', 'Instructor B', 'Programming', 60, '5 hours', 1000),
  (3, 'Course 3', 'Instructor C', 'Mathematics', 40, '3 hours', 1200),
  (4, 'Course 4', 'Instructor D', 'Mathematics', 45, '4.5 hours', 1500),
```

(5, 'Course 5', 'Instructor E', 'Design', 55, '5 hours', 700);

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

```
SELECT Category, SUM(EnrollmentCount) AS TotalEnrollmentCount
FROM Courses
GROUP BY Category
HAVING SUM(EnrollmentCount) > 3000
ORDER BY TotalEnrollmentCount DESC;
```

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

```
SELECT Category
FROM Courses
GROUP BY Category
HAVING COUNT(DISTINCT Instructor) = COUNT(*)
ORDER BY Category;
```

Q2. Model the following Books and Publisher information as a graph model, and answer the following queries using Cypher.

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.

1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model. [3]

Labels:

1. **Author:**
   - Properties:
     - name: Name of the author
     - nationality: Nationality of the author (optional)
     - gender: Gender of the author (optional)
2. **Book:**
   - Properties:
     - title: Title of the book
     - genre: Genre of the book
     - publication_year: Year when the book was published
     - rating: Rating of the book (optional)
3. **Publisher:**
   - Properties:
     - name: Name of the publisher
     - location: Location of the publisher (optional)
     - founded_year: Year when the publisher was founded (optional)
4. **Reader:**
   - Properties:
     - name: Name of the reader

Relationships:

1. **WRITTEN_BY:**
   - Connects an Author to a Book that the author has written.
   - Properties:
     - role: Role of the author (e.g., author, co-author)
2. **PUBLISHED_BY:**
   - Connects a Book to a Publisher that published the book.
   - Properties:
     - imprint: Imprint of the book published by the publisher (optional)
3. **READ_BY:**
   - Connects a Reader to a Book that the reader has read.
   - Properties:
     - rating: Rating provided by the reader for the book
     - review: Review provided by the reader for the book

2. Create nodes and relationships, along with their properties, and visualize your actual Graph model.[3]

```
// Create Author nodes
CREATE (:Author {name: 'Author1', nationality: 'Nationality1', gender: 'Male'}),
    (:Author {name: 'Author2', nationality: 'Nationality2', gender: 'Female'}),
    (:Author {name: 'Author3', nationality: 'Nationality3', gender: 'Male'});

// Create Book nodes
CREATE (:Book {title: 'Book1', genre: 'Comics', publication_year: 2021, rating: 4}),
    (:Book {title: 'Book2', genre: 'Fiction', publication_year: 2020, rating: 3}),
    (:Book {title: 'Book3', genre: 'Comics', publication_year: 2019, rating: 5});

// Create Publisher nodes
CREATE (:Publisher {name: 'Publisher1', location: 'Location1', founded_year: 2000}),
    (:Publisher {name: 'Publisher2', location: 'Location2', founded_year: 1995}),
    (:Publisher {name: 'Publisher3', location: 'Location3', founded_year: 2010});

// Create Reader nodes
CREATE (:Reader {name: 'Reader1'}),
    (:Reader {name: 'Reader2'}),
    (:Reader {name: 'Reader3'});

// Establish relationships
MATCH (a:Author {name: 'Author1'}), (b1:Book {title: 'Book1'}), (p1:Publisher {name: 'Publisher1'})
CREATE (a)-[:WRITTEN_BY {role: 'Author'}]->(b1),
    (b1)-[:PUBLISHED_BY {imprint: 'Imprint1'}]->(p1);
```

```
MATCH (a:Author {name: 'Author2'}), (b2:Book {title: 'Book2'}), (p2:Publisher {name: 'Publisher2'})
CREATE (a)-[:WRITTEN_BY {role: 'Author'}]->(b2),
    (b2)-[:PUBLISHED_BY {imprint: 'Imprint2'}]->(p2);

MATCH (a:Author {name: 'Author3'}), (b3:Book {title: 'Book3'}), (p3:Publisher {name: 'Publisher3'})
CREATE (a)-[:WRITTEN_BY {role: 'Author'}]->(b3),
    (b3)-[:PUBLISHED_BY {imprint: 'Imprint3'}]->(p3);

MATCH (r:Reader {name: 'Reader1'}), (b1:Book {title: 'Book1'})
CREATE (r)-[:READ_BY {rating: 4, review: 'Great book!'}]->(b1);

MATCH (r:Reader {name: 'Reader2'}), (b2:Book {title: 'Book2'})
CREATE (r)-[:READ_BY {rating: 3, review: 'Good read.'}]->(b2);

MATCH (r:Reader {name: 'Reader3'}), (b3:Book {title: 'Book3'})
CREATE (r)-[:READ_BY {rating: 5, review: 'Amazing!'}]->(b3);
```

3. Answer the Queries

A. List the names of authors who wrote “Comics”. [3]

```
MATCH (a:Author)-[:WRITTEN_BY]->(b:Book {genre: 'Comics'})
RETURN DISTINCT a.name AS AuthorName;
```

B. Count no. of readers of _____book published by “Sage”. [3]

```
MATCH (:Publisher {name: 'Sage'})-[:PUBLISHED_BY]->(b:Book {title: 'Book_Title'})
MATCH (r:Reader)-[:READ_BY]->(b)
RETURN COUNT(DISTINCT r) AS ReaderCount;
```

C. List all the publisher whose name starts with “N” [4]

```
MATCH (p:Publisher)
WHERE p.name STARTS WITH 'N'
RETURN p.name AS PublisherName;
```

D. List the names of people who have given a rating of (>=3) for __ book

```
MATCH (p:Person)-[r:READ_BY {rating: >=3}]->(b:Book {title: 'Book_Title'})
RETURN DISTINCT p.name AS PersonName;
```