Advanced Database Web Technologies

Practical Slip Solutions

Slip 1
To model a property system in MongoDB, we can create two collections: `Owners` and `Properties`. Below is a structured representation of the collections, sample documents, and the queries to retrieve the required information.

1. Collections

Owners Collection

Structure:
- `_id`: ObjectId (auto-generated)
- `name`: String (owner's name)
- `contact`: String (owner's contact number)
- `email`: String (owner's email)
- `properties`: Array of property IDs (references to properties owned)

Sample Documents:

```json
[
  {
    "name": "Mr. Patil",
    "contact": "1234567890",
    "email": "patil@example.com",
    "properties": []
  },
  {
    "name": "Mrs. Sharma",
    "contact": "0987654321",
    "email": "sharma@example.com",
    "properties": []
  },
  {
    "name": "Mr. Verma",
    "contact": "1122334455",
    "email": "verma@example.com",
    "properties": []
  },
  {
    "name": "Ms. Desai",
    "contact": "2233445566",
    "email": "desai@example.com",
    "properties": []
  },
  {
    "name": "Mr. Rao",
    "contact": "3344556677",
    "email": "rao@example.com",
    "properties": []
  }
]
```

Properties Collection

Structure:
- `_id`: ObjectId (auto-generated)
- `name`: String (property name)
- `location`: String (property location)
- `area`: String (area or neighborhood)
- `rate`: Number (property rate)
- `owner_id`: ObjectId (reference to the owner)

Sample Documents:

```json
[
  {
    "name": "Sunny Apartments",
    "location": "Nashik",
    "area": "Central",
    "rate": 90000,
    "owner_id": ObjectId("60a7b7c1f9d1e7bff80e0001") // Mr. Patil
  },
  {
    "name": "Green Fields",
    "location": "Nashik",
    "area": "East",
    "rate": 75000,
    "owner_id": ObjectId("60a7b7c1f9d1e7bff80e0001") // Mr. Patil
  },
  {
    "name": "Royal Estates",
    "location": "Pune",
    "area": "Downtown",
    "rate": 120000,
    "owner_id": ObjectId("60a7b7c1f9d1e7bff80e0002") // Mrs. Sharma
  },
  {
    "name": "Ocean View",
    "location": "Mumbai",
    "area": "Coastal",
    "rate": 150000,
    "owner_id": ObjectId("60a7b7c1f9d1e7bff80e0003") // Mr. Verma
  },
  {
    "name": "Mountain Retreat",
    "location": "Nashik",
    "area": "Hills",
    "rate": 85000,
    "owner_id": ObjectId("60a7b7c1f9d1e7bff80e0004") // Ms. Desai
  }
]
```

2. Queries

a. Display area-wise property details.
db.Properties.aggregate([

```
  {
   $group: {
     _id: "$area",
     properties: { $push: { name: "$name", location: "$location", rate: "$rate" } }
   }
  } ]);
```

Example Output:

```
[
  {
    "_id": "Central",
    "properties": [{ "name": "Sunny Apartments", "location": "Nashik", "rate": 90000 }]
  },
  {
    "_id": "East",
    "properties": [{ "name": "Green Fields", "location": "Nashik", "rate": 75000 }]
  },
  {
    "_id": "Downtown",
    "properties": [{ "name": "Royal Estates", "location": "Pune", "rate": 120000 }]
  },
  {
    "_id": "Coastal",
    "properties": [{ "name": "Ocean View", "location": "Mumbai", "rate": 150000 }]
  },
  {
    "_id": "Hills",
    "properties": [{ "name": "Mountain Retreat", "location": "Nashik", "rate": 85000 }]
  }
]
```

b. Display property owned by 'Mr. Patil' having minimum rate.

```
const owner = db.Owners.findOne({ name: "Mr. Patil" });
db.Properties.find({ owner_id: owner._id }).sort({ rate: 1 }).limit(1);
```

Example Output:

```
{
  "name": "Green Fields",
  "location": "Nashik",
  "area": "East",
  "rate": 75000,
  "owner_id": ObjectId("60a7b7c1f9d1e7bff80e0001")
}
```

c. Give the details of the owner whose property is at “Nashik”.

```
const propertiesInNashik = db.Properties.find({ location: "Nashik" }).toArray();
const ownerIds = propertiesInNashik.map(property => property.owner_id);
db.Owners.find({ _id: { $in: ownerIds } });
```

Example Output:

```
[
  {
    "name": "Mr. Patil",
    "contact": "1234567890",
    "email": "patil@example.com",
    "properties": []
  },
  {
    "name": "Ms. Desai",
    "contact": "2233445566",
    "email": "desai@example.com",
    "properties": []
  }
]
```


d. Display the area of property whose rate is less than 100000.

```
db.Properties.find({ rate: { $lt: 100000 } }, { area: 1 });
```

Example Output:


```
[
  { "area": "Central" },
  { "area": "East" },
  { "area": "Hills" }
]
```

Slip 2

Create a container add row inside it and add 3 columns inside row using BootStrap.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Bootstrap Container with Columns</title>
   <!-- Bootstrap CSS -->
   <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>

   <div class="container">
      <div class="row">
         <div class="col">
            <h2>Column 1</h2>
            <p>This is the first column.</p>
         </div>
         <div class="col">
            <h2>Column 2</h2>
            <p>This is the second column.</p>
         </div>
         <div class="col">
            <h2>Column 3</h2>
            <p>This is the third column.</p>
         </div>
      </div>
   </div>

   <!-- Bootstrap JS and dependencies (optional) -->
   <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
   <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
   <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

1. Data Model Design

We'll define the following collections:

Collection: publishers

Each document represents a publisher.

- **Attributes:**
  - _id: Unique identifier for the publisher.
  - name: Name of the publisher.
  - state: State where the publisher is located.
  - city: City where the publisher is located.
  - newspapers: Array of newspaper IDs (referencing documents in the newspapers collection).

Collection: newspapers

Each document represents a newspaper.

- **Attributes:**
  - _id: Unique identifier for the newspaper.
  - title: Title of the newspaper.
  - language: Language of the newspaper.

    - publisher_id: ID of the publisher (referencing a document in the publishers collection).
    - city: City where the newspaper is published.
    - state: State where the newspaper is published.
    - daily_sales: Daily sales figures.

Collection: cities

Each document represents a city.

- **Attributes:**
    - _id: Unique identifier for the city.
    - name: Name of the city.
    - state: State where the city is located.
    - newspapers: Array of newspaper IDs (referencing documents in the newspapers collection).

2. Sample Documents

publishers Collection:

json

```
[
  {
    "_id": 1,
    "name": "Publisher One",
    "state": "Maharashtra",
    "city": "Mumbai",
    "newspapers": [1, 2]
  },
  {
    "_id": 2,
    "name": "Publisher Two",
    "state": "Gujarat",
    "city": "Ahmedabad",
    "newspapers": [3]
  },
  {
    "_id": 3,
    "name": "Publisher Three",
    "state": "Maharashtra",
    "city": "Nashik",
    "newspapers": [4]
  },
  {
    "_id": 4,
    "name": "Publisher Four",
    "state": "Gujarat",
    "city": "Surat",
    "newspapers": [5]
  },
  {
    "_id": 5,
    "name": "Publisher Five",
    "state": "Maharashtra",
    "city": "Pune",
    "newspapers": [6]
  }
]
```

newspapers Collection:

json

Copy code

```
[
  {
    "_id": 1,
    "title": "Mumbai Mirror",
    "language": "English",
    "publisher_id": 1,
    "city": "Mumbai",
    "state": "Maharashtra",
    "daily_sales": 50000
  },
  {
    "_id": 2,
    "title": "Maharashtra Times",
    "language": "Marathi",
    "publisher_id": 1,
    "city": "Mumbai",
    "state": "Maharashtra",
    "daily_sales": 75000
  },
  {
    "_id": 3,
    "title": "Gujarat Samachar",
    "language": "Gujarati",
    "publisher_id": 2,
    "city": "Ahmedabad",
    "state": "Gujarat",
    "daily_sales": 60000
  },
  {
    "_id": 4,
    "title": "Sakal Nashik",
    "language": "Marathi",
    "publisher_id": 3,
    "city": "Nashik",
    "state": "Maharashtra",
    "daily_sales": 40000
  },
  {
    "_id": 5,
    "title": "Sandesh Surat",
    "language": "Gujarati",
    "publisher_id": 4,
    "city": "Surat",
    "state": "Gujarat",
    "daily_sales": 35000
  },
  {
    "_id": 6,
    "title": "Sakal Pune",
    "language": "Marathi",
    "publisher_id": 5,
    "city": "Pune",
    "state": "Maharashtra",
    "daily_sales": 45000
```

```
  }
]
```
cities Collection:
json
Copy code
```
[
  {
    "_id": 1,
    "name": "Mumbai",
    "state": "Maharashtra",
    "newspapers": [1, 2]
  },
  {
    "_id": 2,
    "name": "Ahmedabad",
    "state": "Gujarat",
    "newspapers": [3]
  },
  {
    "_id": 3,
    "name": "Nashik",
    "state": "Maharashtra",
    "newspapers": [4]
  },
  {
    "_id": 4,
    "name": "Surat",
    "state": "Gujarat",
    "newspapers": [5]
  },
  {
    "_id": 5,
    "name": "Pune",
    "state": "Maharashtra",
    "newspapers": [6]
  }
]
```
3. MongoDB Queries

a. List all newspapers available in “Nashik” city.
```
db.newspapers.find({ city: "Nashik" }, { title: 1, _id: 0 })
```
b. List all the newspapers of “Marathi” language.
```
db.newspapers.find({ language: "Marathi" }, { title: 1, _id: 0 })
```
c. Count the number of publishers in the “Gujarat” state.
```
db.publishers.countDocuments({ state: "Gujarat" })
```
d. Write a cursor to show newspapers with the highest sales in Maharashtra State.
```
var cursor = db.newspapers.find({ state: "Maharashtra" }).sort({ daily_sales: -1 }).limit(1);
while (cursor.hasNext()) {
  printjson(cursor.next());
}
```

Slip 3

Write a bootstrap application to display thumbnails of the images.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Thumbnails</title>
    <!-- Bootstrap CSS -->
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <style>
        .thumbnail {
            margin-bottom: 20px;
        }
    </style>
</head>
<body>

    <div class="container">
        <h1 class="text-center my-4">Image Thumbnails</h1>
        <div class="row">
            <div class="col-md-4">
                <div class="card thumbnail">
                    <img src="https://via.placeholder.com/300" class="card-img-top" alt="Image 1">
                    <div class="card-body">
                        <h5 class="card-title">Image 1</h5>
                        <p class="card-text">Description for image 1.</p>
                    </div>
                </div>
            </div>
            <div class="col-md-4">
                <div class="card thumbnail">
                    <img src="https://via.placeholder.com/300" class="card-img-top" alt="Image 2">
                    <div class="card-body">
                        <h5 class="card-title">Image 2</h5>
                        <p class="card-text">Description for image 2.</p>
                    </div>
                </div>
            </div>
            <div class="col-md-4">
                <div class="card thumbnail">
                    <img src="https://via.placeholder.com/300" class="card-img-top" alt="Image 3">
                    <div class="card-body">
                        <h5 class="card-title">Image 3</h5>
                        <p class="card-text">Description for image 3.</p>
                    </div>
                </div>
            </div>
            <!-- Add more images as needed -->
        </div>
    </div>
```

<!—B

Model the following system as a document database. Consider employee and department's information. Assume appropriate attributes and collections as per the query requirements. [3]
Insert at least 5 documents in each collection. [3]
Answer the following Queries.

- **Display name of employee who has highest salary [3]**
- **Display biggest department with max. no. of employees [3]**
- **Write a cursor which shows department wise employee information [4]**
- **List all the employees who work in Sales dept and salary > 50000 [4]**

1. Data Model Design

We'll define the following collections:

Collection: employees

Each document represents an employee.

- **Attributes:**
  - _id: Unique identifier for the employee.
  - name: Name of the employee.
  - department_id: ID of the department (referencing a document in the departments collection).
  - position: Job title or position of the employee.
  - salary: Salary of the employee.

Collection: departments

Each document represents a department.

- **Attributes:**
  - _id: Unique identifier for the department.
  - name: Name of the department.
  - location: Location of the department.
  - employees: Array of employee IDs (referencing documents in the employees collection).

2. Sample Documents

employees Collection:

json
Copy code

```
[
  {
   "_id": 1,
   "name": "John Doe",
   "department_id": 1,
   "position": "Manager",
   "salary": 75000
  },
  {
   "_id": 2,
   "name": "Jane Smith",
   "department_id": 2,
   "position": "Sales Executive",
   "salary": 60000
  },
  {
   "_id": 3,
   "name": "Alice Johnson",
   "department_id": 1,
   "position": "Developer",
   "salary": 50000
```

```
  },
  {
    "_id": 4,
    "name": "Bob Brown",
    "department_id": 3,
    "position": "HR Specialist",
    "salary": 55000
  },
  {
    "_id": 5,
    "name": "Charlie Davis",
    "department_id": 2,
    "position": "Sales Executive",
    "salary": 45000
  }
]
```

departments Collection:

json

Copy code

```
[
  {
    "_id": 1,
    "name": "IT",
    "location": "New York",
    "employees": [1, 3]
  },
  {
    "_id": 2,
    "name": "Sales",
    "location": "San Francisco",
    "employees": [2, 5]
  },
  {
    "_id": 3,
    "name": "HR",
    "location": "Chicago",
    "employees": [4]
  },
  {
    "_id": 4,
    "name": "Marketing",
    "location": "Los Angeles",
    "employees": []
  },
  {
    "_id": 5,
    "name": "Finance",
    "location": "Boston",
    "employees": []
  }
]
```

3. MongoDB Queries

a. Display the name of the employee who has the highest salary.

```
db.employees.find().sort({ salary: -1 }).limit(1).project({ name: 1, _id: 0 })
```

b. Display the biggest department with the maximum number of employees.

```
db.departments.aggregate([
   { $project: { name: 1, numberOfEmployees: { $size: "$employees" } } },
   { $sort: { numberOfEmployees: -1 } },
   { $limit: 1 }
])
```

c. Write a cursor to show department-wise employee information.

```
var cursor = db.departments.aggregate([
   {
      $lookup: {
         from: "employees",
         localField: "employees",
         foreignField: "_id",
         as: "employee_info"
      }
   },
   { $unwind: "$employee_info" },
   { $group: { _id: "$name", employees: { $push: "$employee_info" } } }
]);

while (cursor.hasNext()) {
  printjson(cursor.next());
}
```

d. List all the employees who work in the Sales department and have a salary greater than 50000.

```
db.employees.find({ department_id: 2, salary: { $gt: 50000 } }, { name: 1, _id: 0 })
```

Slip 4

Write a bootstrap program for the following
“The .table class adds basic styling (light padding and only horizontal dividers) to a table” The table can have the first name, last name, and email id as columns

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Bootstrap Table Example</title>
    <!-- Bootstrap CSS -->
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>

    <div class="container my-4">
        <h1 class="text-center">User Information</h1>
        <table class="table">
            <thead class="thead-light">
                <tr>
                    <th>First Name</th>
                    <th>Last Name</th>
                    <th>Email ID</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>John</td>
                    <td>Doe</td>
                    <td>john.doe@example.com</td>
                </tr>
                <tr>
                    <td>Jane</td>
                    <td>Smith</td>
                    <td>jane.smith@example.com</td>
                </tr>
                <tr>
                    <td>Michael</td>
                    <td>Johnson</td>
                    <td>michael.johnson@example.com</td>
                </tr>
                <tr>
                    <td>Emily</td>
                    <td>Williams</td>
                    <td>emily.williams@example.com</td>
                </tr>
                <tr>
                    <td>David</td>
                    <td>Brown</td>
                    <td>david.brown@example.com</td>
                </tr>
            </tbody>
```

```
      </table>
    </div>

    <!-- Bootstrap JS and dependencies (optional) -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

1. Model the following information system as a document database.
Consider hospitals around Nashik. Each hospital may have one or more specializations like Pediatric, Gynaec, Orthopedic, etc. A person can recommend/provide review for a hospital. A doctor can give service to one or more hospitals.
2. Assume appropriate attributes and collections as per the query requirements. [3]
3. Insert at least 10 documents in each collection. [3]
4. Answer the following Queries
a. List the names of hospitals with…… specialization. [3]
b. List the Names of all hospital located in ……. city [3]
c. List the names of hospitals where Dr. Deshmukh visits [4]
d. List the names of hospitals whose rating >=4 [4]

Solution:

1. Schema Design

Hospital Collection

- **Attributes:**
  - hospital_id: Unique identifier for the hospital.
  - name: Name of the hospital.
  - city: City where the hospital is located.
  - specializations: Array of specializations offered by the hospital.
  - doctors: Array of doctor IDs who visit the hospital.
  - rating: Average rating of the hospital.

Doctor Collection

- **Attributes:**
  - doctor_id: Unique identifier for the doctor.
  - name: Name of the doctor.
  - specialization: Specialization of the doctor.
  - hospitals: Array of hospital IDs where the doctor visits.

Review Collection

- **Attributes:**
  - review_id: Unique identifier for the review.
  - hospital_id: ID of the hospital being reviewed.
  - reviewer_name: Name of the person who provided the review.
  - rating: Rating given by the reviewer (out of 5).
  - comment: Textual comment of the review.

2. Sample Documents

Hospital Collection:

```
{
 "hospital_id": 1,
 "name": "Nashik General Hospital",
 "city": "Nashik",
 "specializations": ["Pediatric", "Orthopedic"],
```

```
 "doctors": [101, 102],
 "rating": 4.5
},
{
 "hospital_id": 2,
 "name": "Sunrise Health Care",
 "city": "Nashik",
 "specializations": ["Gynaec", "Cardiology"],
 "doctors": [103, 104],
 "rating": 4.0
}
```

Doctor Collection:

```
{
 "doctor_id": 101,
 "name": "Dr. Deshmukh",
 "specialization": "Pediatric",
 "hospitals": [1]
},
{
 "doctor_id": 102,
 "name": "Dr. Sharma",
 "specialization": "Orthopedic",
 "hospitals": [1, 2]
}
```

Review Collection:

```
{
 "review_id": 1001,
 "hospital_id": 1,
 "reviewer_name": "John Doe",
 "rating": 5,
 "comment": "Excellent care!"
},
{
 "review_id": 1002,
 "hospital_id": 2,
 "reviewer_name": "Jane Smith",
 "rating": 4,
 "comment": "Very professional staff."
}
```

3. Queries

a. List the names of hospitals with a particular specialization:

```
db.hospitals.find({ "specializations": "Pediatric" }, { "name": 1, "_id": 0 })
```

b. List the names of all hospitals located in a specific city:

```
db.hospitals.find({ "city": "Nashik" }, { "name": 1, "_id": 0 })
```

c. List the names of hospitals where Dr. Deshmukh visits:

```
var doctor = db.doctors.findOne({ "name": "Dr. Deshmukh" });
db.hospitals.find({ "doctors": doctor.doctor_id }, { "name": 1, "_id": 0 })
```

d. List the names of hospitals whose rating is greater than or equal to 4:

```
db.hospitals.find({ "rating": { "$gte": 4 } }, { "name": 1, "_id": 0 })
```

This schema and the example queries should meet the requirements for modeling the hospital information system as a document database. You can modify the attributes as needed based on specific requirements.

Slip 5

Write a HTML code, which generate the following output

[Apply border, border radius tags]

List of Persons			
Srno	Person Name	Age	Country
1			
2			
3			

```
<!DOCTYPE  html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>List of Persons</title>
<style>
body {
        font-family: Arial, sans-serif;
        margin: 20px;
        }

table {
        width: 100%;
        border-collapse: collapse;
        border: 2px solid  000; /  Border for the table  /
        border-radius: 10px; /  Rounded corners  /
        overflow: hidden; /  Ensure corners are rounded  /
    }

th, td {
border: 1px solid  000; /  Border for cells  /
padding: 8px;
text-align: left;
}
th {
background-color:  f2f2f2; /  Light grey background for header  /
}
tr:hover {
background-color:  f9f9f9; /  Highlight row on hover  /
}
</style>
</head>
<body>

<h1>List of Persons</h1>
<table>
<thead>

<tr>
<th>Srno</th>
<th>Person Name</th>
<th>Age</th>
```

```
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Doe</td>
<td>30</td>
<td>USA</td>
</tr>
<tr>
<td>2</td>
<td>Jane Smith</td>
<td>25</td>
<td>UK</td>
</tr>
<tr>
<td>3</td>
<td>Emily Johnson</td>
<td>28</td>
<td>Canada</td>
</tr>
</tbody>
</table>
</body>
</html>
```

. Model the following database. Many employees working on one project.
A company has various ongoing projects.
2. Assume appropriate attributes and collections as per the query
requirements. [3]
3. Insert at least 5 documents in each collection. [3]
4. Answer the following Queries
a. List all names of projects where Project_type =….. [3]
b. List all the projects with duration greater than 3 months [3]
c. Count no. of employees working on ……..project [4]
d. List the names of projects on which Mr. Patil is working [4]

To model a database where many employees work on one project and a company has various ongoing projects, we'll use a document-oriented approach, like MongoDB. Below is the schema design, sample documents, and queries based on your requirements.

1. Schema Design

Employee Collection

- **Attributes:**
  - employee_id: Unique identifier for the employee.
  - name: Name of the employee.
  - projects: Array of project IDs the employee is working on.

Project Collection

- **Attributes:**
  - project_id: Unique identifier for the project.
  - name: Name of the project.
  - type: Type of the project (e.g., "Software", "Construction").
  - duration_months: Duration of the project in months.

- employees: Array of employee IDs working on the project.

2. Sample Documents

Employee Collection:


```
{
 "employee_id": 1,
 "name": "Mr. Patil",
 "projects": [101, 102]
},
{
 "employee_id": 2,
 "name": "Ms. Sharma",
 "projects": [101, 103]
},
{
 "employee_id": 3,
 "name": "Mr. Gupta",
 "projects": [102, 104]
},
{
 "employee_id": 4,
 "name": "Ms. Rao",
 "projects": [103, 105]
},
{
 "employee_id": 5,
 "name": "Mr. Mehta",
 "projects": [104, 105]
}
```


Project Collection:


```
{
 "project_id": 101,
 "name": "Project Alpha",
 "type": "Software",
 "duration_months": 6,
 "employees": [1, 2]
},
{
 "project_id": 102,
 "name": "Project Beta",
 "type": "Construction",
 "duration_months": 4,
 "employees": [1, 3]
},
{
 "project_id": 103,
 "name": "Project Gamma",
 "type": "Research",
 "duration_months": 2,
 "employees": [2, 4]
},
{
 "project_id": 104,
 "name": "Project Delta",
 "type": "Software",
```

```
 "duration_months": 5,
 "employees": [3, 5]
},
{
 "project_id": 105,
 "name": "Project Epsilon",
 "type": "Infrastructure",
 "duration_months": 8,
 "employees": [4, 5]
}
```

3. Queries

a. List all names of projects where Project_type = "Software":

```
db.projects.find({ "type": "Software" }, { "name": 1, "_id": 0 })
```

b. List all the projects with a duration greater than 3 months:

```
db.projects.find({ "duration_months": { "$gt": 3 } }, { "name": 1, "_id": 0 })
```

c. Count the number of employees working on "Project Alpha":

```
var project = db.projects.findOne({ "name": "Project Alpha" });
db.employees.count({ "projects": project.project_id })
```

d. List the names of projects on which Mr. Patil is working:

```
var employee = db.employees.findOne({ "name": "Mr. Patil" });
db.projects.find({ "project_id": { "$in": employee.projects } }, { "name": 1, "_id": 0 })
```

Slip 6

Create a web page being rendered in the browser consists of many things - logo, informative text, pictures, hyperlinks, navigational structure and table.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Sample Web Page</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         margin: 20px;
         background-color:  f4f4f4;
      }
      header {
         text-align: center;
         margin-bottom: 20px;
      }
      nav {
         background-color:  333;
         color: white;
         padding: 10px;
      }
      nav a {
         color: white;
         margin: 0 15px;
         text-decoration: none;
      }
      nav a:hover {
```

```
            text-decoration: underline;
        }
        .content {
            display: flex;
            justify-content: space-between;
            margin-bottom: 20px;
        }
        .sidebar {
            width: 30%;
        }
        .main {
            width: 65%;
        }
        table {
            width: 100%;
            border-collapse: collapse;
            margin-top: 20px;
        }
        table, th, td {
            border: 1px solid  333;
        }
        th, td {
            padding: 8px;
            text-align: left;
        }
        th {
            background-color:  4CAF50;
            color: white;
        }
    </style>
</head>
<body>

<header>
    <img src="logo.png" alt="Logo" width="100">
    <h1>Welcome to Our Website</h1>
</header>

<nav>
    <a href=" home">Home</a>
    <a href=" about">About Us</a>
    <a href=" services">Services</a>
    <a href=" contact">Contact</a>
</nav>

<div class="content">
    <div class="sidebar">
        <h2>Informative Text</h2>
        <p>This website is designed to provide you with the latest information and services we offer. Explore our pages
to find out more!</p>
        <p><a href="https://example.com" target="_blank">Learn more about us</a></p>
    </div>

    <div class="main">
```

```
        <h2>Featured Images</h2>
        <img src="image1.jpg" alt="Image 1" width="100%">
        <img src="image2.jpg" alt="Image 2" width="100%" style="margin-top: 10px;">

        <h2>Our Services</h2>
        <table>
            <tr>
                <th>Service</th>
                <th>Description</th>
                <th>Price</th>
            </tr>
            <tr>
                <td>Consulting</td>
                <td>Expert advice for your business.</td>
                <td>$100/hr</td>
            </tr>
            <tr>
                <td>Design</td>
                <td>Creative design solutions.</td>
                <td>$500/project</td>
            </tr>
            <tr>
                <td>Development</td>
                <td>Building web applications.</td>
                <td>$2000/project</td>
            </tr>
        </table>
    </div>
</div>

<footer>
    <p>&copy; 2024 Our Website. All rights reserved.</p>
</footer>

</body>
</html>
```

- **Header**: Contains the logo and main title.
- **Navigation**: Links to different sections of the website.
- **Content Area**: Divided into a sidebar with informative text and main section with images and a table.
- **Table**: Displays services with descriptions and prices.
- **Footer**: Basic copyright information.

Make sure to replace logo.png, image1.jpg, and image2.jpg with actual image file paths to display them correctly. Enjoy building your webpage!

1. Model the following information as a document database.

A customer can take different policies and get the benefit. There are different types of policies provided by various companies

2. Assume appropriate attributes and collections as per the query requirements. [3]
3. Insert at least 5 documents in each collection. [3]
4. Answer the following Queries.

a. List the details of customers who have taken “Komal Jeevan” Policy[3]

b. Display average premium amount [3]

c. Increase the premium amount by 5% for policy type=”Monthly”[4]

d. Count no. of customers who have taken policy type “half yearly”. [4]

To model the information where a customer can take different policies provided by various companies as a document database, we can design collections that represent customers, policies, and companies. Below is the schema design, sample documents, and queries based on the provided requirements.

1. Schema Design

Customer Collection

- **Attributes:**
  - customer_id: Unique identifier for the customer.
  - name: Name of the customer.
  - policies: Array of policy IDs taken by the customer.

Policy Collection

- **Attributes:**
  - policy_id: Unique identifier for the policy.
  - name: Name of the policy.
  - type: Type of the policy (e.g., "Monthly", "Quarterly", "Half Yearly").
  - premium_amount: Premium amount of the policy.
  - company_name: Name of the company providing the policy.
  - customers: Array of customer IDs who have taken the policy.

2. Sample Documents

Customer Collection:

json
Copy code

```
{
 "customer_id": 1,
 "name": "John Doe",
 "policies": [101, 102]
},
{
 "customer_id": 2,
 "name": "Jane Smith",
 "policies": [102, 103]
},
{
 "customer_id": 3,
 "name": "Alice Johnson",
 "policies": [101, 104]
},
{
 "customer_id": 4,
 "name": "Bob Brown",
 "policies": [103, 105]
},
{
 "customer_id": 5,
 "name": "Charlie Davis",
 "policies": [104, 105]
}
```

Policy Collection:

json
Copy code

```
{
 "policy_id": 101,
```

```
  "name": "Komal Jeevan",
  "type": "Monthly",
  "premium_amount": 1000,
  "company_name": "LIC",
  "customers": [1, 3]
},
{
  "policy_id": 102,
  "name": "Jeevan Anand",
  "type": "Quarterly",
  "premium_amount": 1500,
  "company_name": "HDFC Life",
  "customers": [1, 2]
},
{
  "policy_id": 103,
  "name": "Jeevan Saral",
  "type": "Half Yearly",
  "premium_amount": 2000,
  "company_name": "SBI Life",
  "customers": [2, 4]
},
{
  "policy_id": 104,
  "name": "Bhima Bima",
  "type": "Yearly",
  "premium_amount": 2500,
  "company_name": "Max Life",
  "customers": [3, 5]
},
{
  "policy_id": 105,
  "name": "Suraksha Kavach",
  "type": "Monthly",
  "premium_amount": 1200,
  "company_name": "ICICI Prudential",
  "customers": [4, 5]
}
```

3. Queries

a. List the details of customers who have taken the “Komal Jeevan” Policy:

```
var policy = db.policies.findOne({ "name": "Komal Jeevan" });
db.customers.find({ "policies": policy.policy_id }, { "name": 1, "policies": 1, "_id": 0 })
```

b. Display the average premium amount:

```
db.policies.aggregate([
  { "$group": { "_id": null, "averagePremium": { "$avg": "$premium_amount" } } },
  { "$project": { "_id": 0, "averagePremium": 1 } }
])
```

c. Increase the premium amount by 5% for policy type = “Monthly”:

```
db.policies.updateMany(
  { "type": "Monthly" },
  { "$mul": { "premium_amount": 1.05 } }
)
```

d. Count the number of customers who have taken policy type “Half Yearly”:

```
var policies = db.policies.find({ "type": "Half Yearly" }, { "policy_id": 1, "_id": 0 }).toArray();
```

```
var policyIds = policies.map(policy => policy.policy_id);
db.customers.count({ "policies": { "$in": policyIds } })
```

Slip 7

Create a 3D text, apply appropriate font, style, color. Use : Hover in the style selector so that the 3D effects appear only when you hover over the text

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>3D Text Hover Effect</title>
   <style>
      body {
         display: flex;
         justify-content: center;
         align-items: center;
         height: 100vh;
         background-color:  f4f4f4;
         font-family: 'Arial', sans-serif;
      }

      .text-container {
         perspective: 1000px;
      }

      .threeD-text {
         font-size: 50px;
         font-weight: bold;
         color:  007bff;
         text-align: center;
         transition: transform 0.3s, color 0.3s;
      }

      .threeD-text:hover {
         transform: rotateY(15deg) rotateX(15deg);
         color:  ff4500;
         text-shadow: 2px 2px 5px rgba(0, 0, 0, 0.3),
                      -2px -2px 5px rgba(0, 0, 0, 0.3);
      }
   </style>
</head>
<body>

<div class="text-container">
   <div class="threeD-text">WelCome to Mysite!</div>
</div>

</body>
</html>
```

Model the following information as a document database.
A customer operates his bank account, does various transactions and get the banking services
2. Assume appropriate attributes and collections as per the query requirements. [3]
3. Insert at least 5 documents in each collection. [3]
4. Answer the following Queries.
a. List names of all customers whose first name starts with a “S” [3]

b. List all customers who has open an account on 1/1/2020 in ___branch [3]
c. List the names customers where acctype=”Saving” [4]
d. Count total no. of loan account holder of …….branch [4]
solution:

To model a database for customers operating bank accounts, performing transactions, and utilizing banking services, we can create collections for customers, accounts, transactions, and branches. Below is the schema design, sample documents, and queries based on your requirements.

1. Schema Design

Customer Collection

- **Attributes:**
  - customer_id: Unique identifier for the customer.
  - first_name: First name of the customer.
  - last_name: Last name of the customer.
  - accounts: Array of account IDs held by the customer.
  - branch_id: ID of the branch where the customer opened the account.

Account Collection

- **Attributes:**
  - account_id: Unique identifier for the account.
  - acctype: Type of account (e.g., "Saving", "Current", "Loan").
  - balance: Current balance of the account.
  - customer_id: ID of the customer who owns the account.
  - open_date: Date when the account was opened.
  - branch_id: ID of the branch where the account was opened.

Branch Collection

- **Attributes:**
  - branch_id: Unique identifier for the branch.
  - branch_name: Name of the branch.
  - location: Location of the branch.

Transaction Collection

- **Attributes:**
  - transaction_id: Unique identifier for the transaction.
  - account_id: ID of the account related to the transaction.
  - transaction_date: Date of the transaction.
  - amount: Amount involved in the transaction.
  - transaction_type: Type of transaction (e.g., "Deposit", "Withdrawal").

2. Sample Documents

Customer Collection:

```
{
 "customer_id": 1,
 "first_name": "Sam",
 "last_name": "Wilson",
 "accounts": [101, 102],
 "branch_id": 201
},
{
 "customer_id": 2,
 "first_name": "Sophia",
 "last_name": "Brown",
 "accounts": [103],
 "branch_id": 202
},
{
 "customer_id": 3,
```

```
 "first_name": "John",
 "last_name": "Smith",
 "accounts": [104],
 "branch_id": 201
},
{
 "customer_id": 4,
 "first_name": "Sarah",
 "last_name": "Connor",
 "accounts": [105],
 "branch_id": 203
},
{
 "customer_id": 5,
 "first_name": "Alex",
 "last_name": "Johnson",
 "accounts": [106],
 "branch_id": 202
}
```


Account Collection:


```
{
 "account_id": 101,
 "acctype": "Saving",
 "balance": 5000,
 "customer_id": 1,
 "open_date": "2020-01-01",
 "branch_id": 201
},
{
 "account_id": 102,
 "acctype": "Loan",
 "balance": 20000,
 "customer_id": 1,
 "open_date": "2019-12-15",
 "branch_id": 201
},
{
 "account_id": 103,
 "acctype": "Saving",
 "balance": 7000,
 "customer_id": 2,
 "open_date": "2020-01-01",
 "branch_id": 202
},
{
 "account_id": 104,
 "acctype": "Current",
 "balance": 10000,
 "customer_id": 3,
 "open_date": "2018-03-20",
 "branch_id": 201
},
{
 "account_id": 105,
```

```
 "acctype": "Saving",
 "balance": 12000,
 "customer_id": 4,
 "open_date": "2020-01-01",
 "branch_id": 203
}
```


Branch Collection:


```
{
 "branch_id": 201,
 "branch_name": "Main Branch",
 "location": "City Center"
},
{
 "branch_id": 202,
 "branch_name": "East Branch",
 "location": "East Side"
},
{
 "branch_id": 203,
 "branch_name": "West Branch",
 "location": "West Side"
}
```


Transaction Collection:


```
{
 "transaction_id": 1001,
 "account_id": 101,
 "transaction_date": "2024-01-15",
 "amount": 1000,
 "transaction_type": "Deposit"
},
{
 "transaction_id": 1002,
 "account_id": 102,
 "transaction_date": "2024-02-05",
 "amount": 2000,
 "transaction_type": "Withdrawal"
},
{
 "transaction_id": 1003,
 "account_id": 103,
 "transaction_date": "2024-03-10",
 "amount": 1500,
 "transaction_type": "Deposit"
},
{
 "transaction_id": 1004,
 "account_id": 104,
 "transaction_date": "2024-01-25",
 "amount": 5000,
 "transaction_type": "Deposit"
},
{
 "transaction_id": 1005,
 "account_id": 105,
```

```
  "transaction_date": "2024-02-20",
  "amount": 2000,
  "transaction_type": "Withdrawal"
}
```

3. Queries

a. List names of all customers whose first name starts with "S":

```
db.customers.find({ "first_name": { "$regex": "^S" } }, { "first_name": 1, "last_name": 1, "_id": 0 })
```

b. List all customers who opened an account on 1/1/2020 in a specific branch (e.g., Main Branch):

```
var branch = db.branches.findOne({ "branch_name": "Main Branch" });
db.accounts.aggregate([
  { "$match": { "open_date": "2020-01-01", "branch_id": branch.branch_id } },
  { "$lookup": { "from": "customers", "localField": "customer_id", "foreignField": "customer_id", "as":
"customer_details" } },
  { "$unwind": "$customer_details" },
  { "$project": { "customer_details.first_name": 1, "customer_details.last_name": 1, "_id": 0 } }
])
```

c. List the names of customers where acctype = "Saving":

```
db.accounts.aggregate([
  { "$match": { "acctype": "Saving" } },
  { "$lookup": { "from": "customers", "localField": "customer_id", "foreignField": "customer_id", "as":
"customer_details" } },
  { "$unwind": "$customer_details" },
  { "$project": { "customer_details.first_name": 1, "customer_details.last_name": 1, "_id": 0 } }
])
```

d. Count the total number of loan account holders in a specific branch (e.g., East Branch):

```
var branch = db.branches.findOne({ "branch_name": "East Branch" });
db.accounts.count({ "acctype": "Loan", "branch_id": branch.branch_id })
```

Slip No 8

Create a button with different style (Secondary, Primary, Success, Error, Info, Warning, Danger) using BootStrap

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Bootstrap Button Styles</title>
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
   <style>
      body {
         display: flex;
         flex-direction: column;
         align-items: center;
         justify-content: center;
         height: 100vh;
         background-color:  f4f4f4;
      }
      .btn-group {
         margin: 10px;
      }
   </style>
</head>
<body>

<h1>Bootstrap Button Styles</h1>

<div class="btn-group">
   <button class="btn btn-primary">Primary</button>
   <button class="btn btn-secondary">Secondary</button>
   <button class="btn btn-success">Success</button>
   <button class="btn btn-danger">Danger</button>
   <button class="btn btn-warning">Warning</button>
   <button class="btn btn-info">Info</button>
   <button class="btn btn-light">Light</button>
   <button class="btn btn-dark">Dark</button>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>
</html>
```

Explanation:

- Bootstrap CSS : The code includes the Bootstrap CSS from a CDN, which provides the styles for the buttons.
- Button Classes : Each button uses Bootstrap classes:
  - `btn-primary`: Blue button
  - `btn-secondary`: Gray button
  - `btn-success`: Green button
  - `btn-danger`: Red button
  - `btn-warning`: Yellow button
  - `btn-info`: Light blue button

- `btn-light`: Light gray button
- `btn-dark`: Dark gray button
- Layout : The buttons are arranged in a flex container to center them vertically and horizontally.

You can modify the button text or add additional styles as needed. Just save the code and open it in a browser to see the buttons in action!

1. Model the following inventory information as a document database.

The inventory keeps track of various items. The items are tagged in various categories. Items may be kept in various warehouses and each warehouse keeps track of the quantity of the item.

2. Assume appropriate attributes and collections as per the query requirements [3]
3. Insert at least 5 documents in each collection. [3]
4. Answer the following Queries.
    a. List all the items qty is greater than 300 [3]
    b. List all items which have tags less than 5 [3]
    c. List all items having status equal to “B” or having quantity less than 50 and height of the product should be greater than 8 [4]
    d. Find all warehouse that keeps item “Planner” and having in stock quantity less than 20 [4]

To model the inventory information as a document database, we'll define two collections: Items and Warehouses . Each collection will have appropriate attributes based on the requirements.

Collections and Documents

1. Collection: `Items`

Attributes:

- `item_id`: Unique identifier for the item
- `name`: Name of the item
- `tags`: Array of tags associated with the item
- `quantity`: Quantity of the item in stock
- `status`: Status of the item (e.g., "A", "B", "C")
- `height`: Height of the item

Example Documents:

```json
[
  {
    "item_id": "1",
    "name": "Notebook",
    "tags": ["stationery", "writing", "paper"],
    "quantity": 350,
    "status": "A",
    "height": 10
  },
  {
    "item_id": "2",
    "name": "Planner",
    "tags": ["stationery", "planning", "organizer"],
    "quantity": 15,
    "status": "B",
    "height": 9
```

```
    },
    {
        "item_id": "3",
        "name": "Marker",
        "tags": ["stationery", "writing", "color"],
        "quantity": 450,
        "status": "A",
        "height": 5
    },
    {
        "item_id": "4",
        "name": "Eraser",
        "tags": ["stationery", "writing"],
        "quantity": 20,
        "status": "C",
        "height": 3
    },
    {
        "item_id": "5",
        "name": "Folder",
        "tags": ["stationery", "organization", "storage"],
        "quantity": 500,
        "status": "A",
        "height": 12
    }
]
```

2. Collection: `Warehouses`

Attributes:

- `warehouse_id`: Unique identifier for the warehouse
- `location`: Location of the warehouse
- `stock`: Array of items stored in the warehouse, with quantity

Example Documents:

```json
[
    {
        "warehouse_id": "1",
        "location": "New York",
        "stock": [
            {"item_id": "1", "quantity": 200},
            {"item_id": "2", "quantity": 15}
        ]
    },
    {
        "warehouse_id": "2",
        "location": "Los Angeles",
        "stock": [
            {"item_id": "3", "quantity": 300},
            {"item_id": "4", "quantity": 20}
        ]
    },
    {
        "warehouse_id": "3",
        "location": "Chicago",
```

```
      "stock": [
        {"item_id": "5", "quantity": 150}
      ]
    },
    {
      "warehouse_id": "4",
      "location": "Houston",
      "stock": [
        {"item_id": "1", "quantity": 150},
        {"item_id": "3", "quantity": 150}
      ]
    },
    {
      "warehouse_id": "5",
      "location": "Miami",
      "stock": [
        {"item_id": "2", "quantity": 5},
        {"item_id": "5", "quantity": 200}
      ]
    }
]
```

Queries

a. List all the items where quantity is greater than 300

```json
db.Items.find({"quantity": {"$gt": 300}})
```

Result:

```json
[
    {
      "item_id": "1",
      "name": "Notebook",
      "tags": ["stationery", "writing", "paper"],
      "quantity": 350,
      "status": "A",
      "height": 10
    },
    {
      "item_id": "3",
      "name": "Marker",
      "tags": ["stationery", "writing", "color"],
      "quantity": 450,
      "status": "A",
      "height": 5
    },
    {
      "item_id": "5",
      "name": "Folder",
      "tags": ["stationery", "organization", "storage"],
      "quantity": 500,
```

```
    "status": "A",
    "height": 12
  }
]
```

b. List all items which have tags less than 5

```json
db.Items.find({"tags": {"$size": {"$lt": 5}}})
```

Result:

```json
[
  {
    "item_id": "2",
    "name": "Planner",
    "tags": ["stationery", "planning", "organizer"],
    "quantity": 15,
    "status": "B",
    "height": 9
  },
  {
    "item_id": "4",
    "name": "Eraser",
    "tags": ["stationery", "writing"],
    "quantity": 20,
    "status": "C",
    "height": 3
  }
]
```

c. List all items having status equal to “B” or having quantity less than 50 and height of the product greater than 8

```json
db.Items.find({
  "$or": [
    {"status": "B"},
    {"$and": [{"quantity": {"$lt": 50}}, {"height": {"$gt": 8}}]}
  ]
})
```

Result:

```json
[
  {
    "item_id": "2",
    "name": "Planner",
    "tags": ["stationery", "planning", "organizer"],
    "quantity": 15,
    "status": "B",
```

```
      "height": 9
   },
   {
      "item_id": "4",
      "name": "Eraser",
      "tags": ["stationery", "writing"],
      "quantity": 20,
      "status": "C",
      "height": 3
   }
]
```

d. Find all warehouses that keep item “Planner” and have in-stock quantity less than 20

```json
db.Warehouses.find({
   "stock": {
      "$elemMatch": {
         "item_id": "2",
         "quantity": {"$lt": 20}
      }
   }
})
```

Result:

```json
[
   {
      "warehouse_id": "1",
      "location": "New York",
      "stock": [
         {"item_id": "1", "quantity": 200},
         {"item_id": "2", "quantity": 15}
      ]
   },
   {
      "warehouse_id": "5",
      "location": "Miami",
      "stock": [
         {"item_id": "2", "quantity": 5},
         {"item_id": "5", "quantity": 200}
      ]
   }
]
```

Slip 9

Write an HTML 5 program for student registration form for college admission. Use input type like search, email, date etc

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Student Registration Form</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <style>
        body {
            background-color:  f4f4f4;
            padding: 20px;
        }
        .form-container {
            background-color: white;
            padding: 20px;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
    </style>
</head>
<body>

<div class="container form-container">
    <h2>Student Registration Form</h2>
    <form>
        <div class="form-group">
            <label for="studentName">Full Name:</label>
            <input type="text" class="form-control" id="studentName" placeholder="Enter your full name" required>
        </div>

        <div class="form-group">
            <label for="email">Email:</label>
            <input type="email" class="form-control" id="email" placeholder="Enter your email" required>
        </div>

        <div class="form-group">
            <label for="dateOfBirth">Date of Birth:</label>
            <input type="date" class="form-control" id="dateOfBirth" required>
        </div>

        <div class="form-group">
            <label for="phone">Phone Number:</label>
            <input type="tel" class="form-control" id="phone" placeholder="Enter your phone number" required>
        </div>

        <div class="form-group">
            <label for="address">Address:</label>
            <textarea class="form-control" id="address" rows="3" placeholder="Enter your address" required></textarea>
        </div>

        <div class="form-group">
            <label for="course">Course of Study:</label>
            <input type="search" class="form-control" id="course" placeholder="Search for your course" required>
        </div>

        <div class="form-group">
            <label for="gender">Gender:</label>
```

```
            <select class="form-control" id="gender" required>
                <option value="">Select your gender</option>
                <option value="male">Male</option>
                <option value="female">Female</option>
                <option value="other">Other</option>
            </select>
        </div>

        <div class="form-group">
            <label for="terms">
                <input type="checkbox" id="terms" required> I agree to the terms and conditions
            </label>
        </div>

        <button type="submit" class="btn btn-primary">Register</button>
    </form>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>
</html>
```

Explanation:

- Form Structure : The form includes various input fields:
  - Full Name : Text input for the student's name.
  - Email : Email input to validate email format.
  - Date of Birth : Date input for selecting a date.
  - Phone Number : Telephone input for phone numbers.
  - Address : Text area for the address.
  - Course of Study : Search input for finding courses.
  - Gender : Dropdown select for gender options.
  - Terms and Conditions : Checkbox for agreement.
- Bootstrap : The form uses Bootstrap classes for styling and layout, making it responsive and visually appealing.
- Validation : Required fields are indicated to ensure that all necessary information is provided before submission.

1. Model the following Customer Loan information as a document database.

Consider Customer Loan information system where the customer can take
many types of loans.
2. Assume appropriate attributes and collections as per the query

requirements [3]
3. Insert at least 10 documents in each collection. [3]
4. Answer the following Queries.
a. List all customers whose name starts with 'D' character [3]
b. List the names of customer in descending order who has taken a loan from Pimpri city. [3]
c. Display customer details having maximum loan amount. [4]
d. Update the address of customer whose name is “Mr. Patil” and loan_amt is greater than 100000. [4]

To model a Customer Loan information system as a document database, we can use a structure that consists of collections representing customers and loans. Here's a breakdown of the schema and sample documents.

Collections

1. Customers
   - Attributes:
     - `customer_id`: Unique identifier for the customer
     - `name`: Name of the customer
     - `address`: Address of the customer
     - `phone`: Phone number of the customer
     - `email`: Email of the customer

2. Loans
   - Attributes:
     - `loan_id`: Unique identifier for the loan
     - `customer_id`: Reference to the customer who took the loan
     - `loan_type`: Type of loan (e.g., personal, home, auto)
     - `loan_amt`: Amount of the loan
     - `city`: City where the loan was taken
     - `date_taken`: Date when the loan was taken

Sample Documents

Customers Collection

```json
[
  {"customer_id": 1, "name": "David Smith", "address": "123 Elm St", "phone": "123-456-7890", "email": "david@example.com"},
  {"customer_id": 2, "name": "Diana Prince", "address": "456 Maple St", "phone": "234-567-8901", "email": "diana@example.com"},
  {"customer_id": 3, "name": "John Doe", "address": "789 Oak St", "phone": "345-678-9012", "email": "john@example.com"},
  {"customer_id": 4, "name": "Mr. Patil", "address": "101 Pine St", "phone": "456-789-0123", "email": "patil@example.com"},
  {"customer_id": 5, "name": "Alice Brown", "address": "202 Birch St", "phone": "567-890-1234", "email": "alice@example.com"},
  {"customer_id": 6, "name": "Daniel Craig", "address": "303 Cedar St", "phone": "678-901-2345", "email": "daniel@example.com"},
  {"customer_id": 7, "name": "Dorothy Gale", "address": "404 Fir St", "phone": "789-012-3456", "email": "dorothy@example.com"},
  {"customer_id": 8, "name": "Steve Rogers", "address": "505 Spruce St", "phone": "890-123-4567", "email": "steve@example.com"},
  {"customer_id": 9, "name": "Bruce Wayne", "address": "606 Willow St", "phone": "901-234-5678", "email": "bruce@example.com"},
  {"customer_id": 10, "name": "Clark Kent", "address": "707 Cherry St", "phone": "012-345-6789", "email": "clark@example.com"}
]
```

Loans Collection

```json
```

```
[
  {"loan_id": 1, "customer_id": 1, "loan_type": "Personal", "loan_amt": 50000, "city": "Pimpri", "date_taken": "2024-01-15"},
  {"loan_id": 2, "customer_id": 2, "loan_type": "Home", "loan_amt": 200000, "city": "Pimpri", "date_taken": "2023-11-20"},
  {"loan_id": 3, "customer_id": 3, "loan_type": "Auto", "loan_amt": 15000, "city": "Mumbai", "date_taken": "2023-12-01"},
  {"loan_id": 4, "customer_id": 4, "loan_type": "Personal", "loan_amt": 120000, "city": "Pimpri", "date_taken": "2024-02-10"},
  {"loan_id": 5, "customer_id": 5, "loan_type": "Home", "loan_amt": 300000, "city": "Navi Mumbai", "date_taken": "2024-03-05"},
  {"loan_id": 6, "customer_id": 6, "loan_type": "Personal", "loan_amt": 75000, "city": "Pimpri", "date_taken": "2024-04-22"},
  {"loan_id": 7, "customer_id": 7, "loan_type": "Auto", "loan_amt": 22000, "city": "Thane", "date_taken": "2024-05-30"},
  {"loan_id": 8, "customer_id": 8, "loan_type": "Home", "loan_amt": 500000, "city": "Pimpri", "date_taken": "2024-06-15"},
  {"loan_id": 9, "customer_id": 9, "loan_type": "Personal", "loan_amt": 60000, "city": "Pimpri", "date_taken": "2024-07-19"},
  {"loan_id": 10, "customer_id": 10, "loan_type": "Home", "loan_amt": 250000, "city": "Mumbai", "date_taken": "2024-08-25"}
]
```

Queries

a. List all customers whose name starts with 'D'

```json
db.customers.find({ "name": /^D/ })
```

b. List the names of customers in descending order who have taken a loan from Pimpri city

```json
db.customers.aggregate([
  { $lookup: {
    from: "loans",
    localField: "customer_id",
    foreignField: "customer_id",
    as: "customer_loans"
  }},
  { $match: { "customer_loans.city": "Pimpri" }},
  { $project: { name: 1 }},
  { $sort: { name: -1 }}
])
```

c. Display customer details having the maximum loan amount

```json
const maxLoan = db.loans.find().sort({ loan_amt: -1 }).limit(1).toArray()[0];
db.customers.find({ customer_id: maxLoan.customer_id });
```

d. Update the address of customer whose name is “Mr. Patil” and loan_amt is greater than 100000

```json
db.customers.updateOne(
  { "name": "Mr. Patil" },
  { $set: { "address": "New Address St" }},
  { $merge: true }
);
```

Slip no 10

Create a web page that shows use of transition properties, transition delay and duration effect.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS Transition Effects</title>
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color:  f4f4f4;
            font-family: Arial, sans-serif;
        }

        .box {
            width: 100px;
            height: 100px;
            background-color:  007bff;
            margin: 20px;
            transition: all 0.5s ease;
            cursor: pointer;
        }

        .box:hover {
            background-color:  28a745;
            transform: scale(1.2);
        }

        .box.transition-delay {
            transition-delay: 0.3s; /  Delay effect  /
        }

        .box.transition-duration {
            transition-duration: 1s; /  Duration effect  /
        }

        .box.transition-both {
            transition: all 0.5s ease 0.3s; /  Both duration and delay  /
        }
    </style>
</head>
<body>

<div class="box transition-both">Hover Me!</div>
<div class="box transition-delay">Hover Me!</div>
<div class="box transition-duration">Hover Me!</div>

<script>
    // Optional: Add some interactivity with JavaScript if needed
```

```
</script>

</body>
</html>
```

Explanation:

- HTML Structure : The page contains three boxes, each demonstrating different transition effects.
- CSS Styles :
  - Box Style : Each box has a width, height, background color, and margin. The `transition` property applies to all properties and has a duration of 0.5 seconds with an easing function.
  - Hover Effect : On hover, the background color changes and the box scales up.
  - Transition Delay : The box with the class `transition-delay` has a delay before the transition starts (0.3 seconds).
  - Transition Duration : The box with the class `transition-duration` has an extended duration for transitions (1 second).
  - Both Effects : The box with the class `transition-both` combines both duration and delay in a single transition property.

1. Model the following Online shopping information as a document database. Consider online shopping where the customer can get different products from different brands. Customers can rate the brands and products
2. Assume appropriate attributes and collections as per the query requirements [3]
3. Insert at least 5 documents in each collection. [3]
4. Answer the following Queries.

a. List the names of product whose warranty period is one year [3]
b. List the customers has done purchase on "15/08/2023". [3]
c. Display the name s of products with brand which have highest rating.[4]
d. Display customers who stay in …… city and billamt >50000 . [4]

To model an online shopping information system as a document database, we can create collections for customers, products, and brands. This structure allows us to manage relationships between customers, products, and brands effectively.

Collections

1. Customers
   - Attributes:
     - `customer_id`: Unique identifier for the customer
     - `name`: Name of the customer
     - `address`: Address of the customer
     - `city`: City where the customer lives
     - `email`: Email of the customer
     - `purchases`: List of purchase records (each containing product_id, purchase_date, and bill_amt)
2. Products
   - Attributes:
     - `product_id`: Unique identifier for the product
     - `name`: Name of the product
     - `brand_id`: Reference to the brand of the product
     - `warranty_period`: Warranty period of the product (in years)
     - `rating`: Average rating of the product
3. Brands
   - Attributes:
     - `brand_id`: Unique identifier for the brand
     - `name`: Name of the brand
     - `rating`: Average rating of the brand

Sample Documents

Customers Collection

```json
[
  {"customer_id": 1, "name": "Alice Johnson", "address": "123 Main St", "city": "New York", "email": "alice@example.com", "purchases": [{"product_id": 1, "purchase_date": "2023-08-15", "bill_amt": 60000}]},
  {"customer_id": 2, "name": "Bob Smith", "address": "456 Park Ave", "city": "Los Angeles", "email": "bob@example.com", "purchases": [{"product_id": 2, "purchase_date": "2023-07-20", "bill_amt": 30000}]},
  {"customer_id": 3, "name": "Charlie Brown", "address": "789 Pine St", "city": "Chicago", "email": "charlie@example.com", "purchases": []},
  {"customer_id": 4, "name": "Diana Prince", "address": "101 Maple St", "city": "New York", "email": "diana@example.com", "purchases": [{"product_id": 3, "purchase_date": "2023-08-15", "bill_amt": 55000}]},
  {"customer_id": 5, "name": "Eve Adams", "address": "202 Oak St", "city": "Miami", "email": "eve@example.com", "purchases": [{"product_id": 4, "purchase_date": "2023-09-01", "bill_amt": 45000}]}
]
```

Products Collection

```json
[
  {"product_id": 1, "name": "Smartphone A", "brand_id": 1, "warranty_period": 1, "rating": 4.5},
  {"product_id": 2, "name": "Laptop B", "brand_id": 2, "warranty_period": 2, "rating": 4.7},
  {"product_id": 3, "name": "Tablet C", "brand_id": 1, "warranty_period": 1, "rating": 4.3},
  {"product_id": 4, "name": "Headphones D", "brand_id": 3, "warranty_period": 1, "rating": 4.8},
  {"product_id": 5, "name": "Smartwatch E", "brand_id": 2, "warranty_period": 1, "rating": 4.2}
]
```

Brands Collection

```json
[
  {"brand_id": 1, "name": "BrandX", "rating": 4.5},
  {"brand_id": 2, "name": "BrandY", "rating": 4.7},
  {"brand_id": 3, "name": "BrandZ", "rating": 4.8}
]
```

Queries

a. List the names of products whose warranty period is one year

```json
db.products.find({ "warranty_period": 1 }, { "name": 1 })
```

b. List the customers who made a purchase on “15/08/2023”

```json
db.customers.find({ "purchases.purchase_date": "2023-08-15" }, { "name": 1 })
```

c. Display the names of products with the brand that has the highest rating

```json
const highestRatedBrand = db.brands.find().sort({ rating: -1 }).limit(1).toArray()[0];
db.products.find({ brand_id: highestRatedBrand.brand_id }, { name: 1 })
```

d. Display customers who stay in a specific city (e.g., "New York") and have a bill amount greater than 50000

```json
db.customers.find({
  "city": "New York",
  "purchases.bill_amt": { $gt: 50000 }
}, { "name": 1 })
```

Slip no 11

Write a HTML code which will divide web page in three frames. First frame should consists of company name as heading. Second frame should consists of name of departments with hyperlink.
Once click on any department, it should display information of that department in third frame.

To create a web page with three frames (or sections) that displays a company name, a list of departments with hyperlinks, and information for each department, you can use a combination of HTML and CSS. Below is a simple example using `<iframe>` elements.

Note: Since the use of `<frameset>` is deprecated in HTML5, this example will use CSS Flexbox to create a layout that mimics the behavior of frames.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Company Departments</title>
    <style>
        body {
            display: flex;
            height: 100vh;
            margin: 0;
            font-family: Arial, sans-serif;
        }
        .header {
            flex: 1;
            background-color:  007bff;
            color: white;
            display: flex;
            align-items: center;
            justify-content: center;
            font-size: 24px;
        }
        .sidebar {
            flex: 1;
            background-color:  f4f4f4;
            padding: 20px;
        }
        .content {
            flex: 2;
            background-color:  ffffff;
            padding: 20px;
            border-left: 1px solid  ddd;
        }
        .department-link {
            display: block;
            margin: 10px 0;
            color:  007bff;
            text-decoration: none;
        }
        .department-link:hover {
```

```
            text-decoration: underline;
        }
    </style>
</head>
<body>

<div class="header">
    <h1>Company Name</h1>
</div>

<div class="sidebar">
    <h2>Departments</h2>
    <a href="  " class="department-link" onclick="showInfo('HR')">Human Resources</a>
    <a href="  " class="department-link" onclick="showInfo('IT')">Information Technology</a>
    <a href="  " class="department-link" onclick="showInfo('Sales')">Sales</a>
    <a href="  " class="department-link" onclick="showInfo('Marketing')">Marketing</a>
</div>

<div class="content" id="department-info">
    <h2>Select a department to see information</h2>
</div>

<script>
    function showInfo(department) {
        const info = {
            HR: "The Human Resources department manages employee relations, benefits, and recruitment.",
            IT: "The Information Technology department oversees the technology infrastructure and support.",
            Sales: "The Sales department is responsible for selling products and maintaining customer relationships.",
            Marketing: "The Marketing department focuses on promoting the company and its products."
        };

        document.getElementById('department-info').innerHTML = `<h2>${department}
Department</h2><p>${info[department]}</p>`;
    }
</script>

</body>
</html>
```

Explanation:

- Structure :
  - The page is divided into three main sections: a header, a sidebar for department links, and a content area for displaying department information.
- Header : Displays the company name.
- Sidebar : Contains department names as clickable links. Each link calls the `showInfo` function when clicked.
- Content Area : Displays information about the selected department.
- JavaScript : The `showInfo` function updates the content area with information about the selected department based on a predefined object containing descriptions.

1. 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.
2. Assume appropriate attributes and collections as per the query requirements.[3]
3. Insert at least 5 documents in each collection. [3]

4. Answer the following Queries.
a. List all products in the inventory. [3]
b. List the details of orders with a value >20000. [3]
c. List all the orders which has not been processed (invoice not generated)[4]
d. List all the orders along with their invoice for “Mr. Rajiv”.[4]

To model a sales system as a document database, we can create collections for products, customers, orders, and invoices. This structure captures the relationships among these entities and allows for efficient querying.

Collections

1. Products
- Attributes:
  - `product_id`: Unique identifier for the product
  - `name`: Name of the product
  - `price`: Price of the product
  - `stock`: Available stock quantity
  - `category`: Category of the product

2. Customers
- Attributes:
  - `customer_id`: Unique identifier for the customer
  - `name`: Name of the customer
  - `email`: Email of the customer
  - `address`: Address of the customer
  - `phone`: Phone number of the customer

3. Orders
- Attributes:
  - `order_id`: Unique identifier for the order
  - `customer_id`: Reference to the customer who placed the order
  - `order_date`: Date when the order was placed
  - `total_value`: Total value of the order
  - `status`: Status of the order (e.g., "Processed", "Pending")

4. Invoices
- Attributes:
  - `invoice_id`: Unique identifier for the invoice
  - `order_id`: Reference to the associated order
  - `invoice_date`: Date when the invoice was generated
  - `amount`: Amount to be paid (matches the order total)

Sample Documents

Products Collection

```json
[
  {"product_id": 1, "name": "Laptop A", "price": 80000, "stock": 15, "category": "Electronics"},
  {"product_id": 2, "name": "Smartphone B", "price": 50000, "stock": 30, "category": "Electronics"},
  {"product_id": 3, "name": "Washing Machine C", "price": 30000, "stock": 10, "category": "Home Appliances"},
  {"product_id": 4, "name": "Headphones D", "price": 2000, "stock": 50, "category": "Electronics"},
  {"product_id": 5, "name": "Refrigerator E", "price": 60000, "stock": 5, "category": "Home Appliances"}
]
```

```

Customers Collection

```json
[
  {"customer_id": 1, "name": "Mr. Rajiv", "email": "rajiv@example.com", "address": "123 Main St", "phone": "9876543210"},
  {"customer_id": 2, "name": "Alice Johnson", "email": "alice@example.com", "address": "456 Park Ave", "phone": "1234567890"},
  {"customer_id": 3, "name": "Bob Smith", "email": "bob@example.com", "address": "789 Pine St", "phone": "2345678901"},
  {"customer_id": 4, "name": "Diana Prince", "email": "diana@example.com", "address": "101 Maple St", "phone": "3456789012"},
  {"customer_id": 5, "name": "Charlie Brown", "email": "charlie@example.com", "address": "202 Oak St", "phone": "4567890123"}
]
```

Orders Collection

```json
[
  {"order_id": 1, "customer_id": 1, "order_date": "2023-08-15", "total_value": 82000, "status": "Processed"},
  {"order_id": 2, "customer_id": 2, "order_date": "2023-09-01", "total_value": 25000, "status": "Pending"},
  {"order_id": 3, "customer_id": 1, "order_date": "2023-09-05", "total_value": 12000, "status": "Pending"},
  {"order_id": 4, "customer_id": 3, "order_date": "2023-09-10", "total_value": 22000, "status": "Processed"},
  {"order_id": 5, "customer_id": 4, "order_date": "2023-09-15", "total_value": 45000, "status": "Pending"}
]
```

Invoices Collection

```json
[
  {"invoice_id": 1, "order_id": 1, "invoice_date": "2023-08-16", "amount": 82000},
  {"invoice_id": 2, "order_id": 4, "invoice_date": "2023-09-11", "amount": 22000}
]
```

Queries

a. List all products in the inventory

```json
db.products.find({}, { "name": 1, "price": 1, "stock": 1 })
```

b. List the details of orders with a value > 20000

```json
db.orders.find({ "total_value": { $gt: 20000 } })
```
```

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

```json
db.orders.find({ "status": "Pending" })
```

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

```json
db.orders.aggregate([
  { $match: { "customer_id": 1 } }, // Assuming Mr. Rajiv has customer_id 1
  { $lookup: {
    from: "invoices",
    localField: "order_id",
    foreignField: "order_id",
    as: "invoice_details"
  }}
])
```

Slip no 12

Design an appropriate HTML form for customer registration visiting a departmental store. Form should consist of fields such as name, contact no, gender, preferred days of purchasing, favorite item (to be selected from a list of items), suggestions etc. You should provide button to submit as well as reset the form contents.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Customer Registration Form</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <style>
        body {
            background-color:   f4f4f4;
            padding: 20px;
        }
        .form-container {
            background-color: white;
            padding: 20px;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
    </style>
</head>
<body>

<div class="container form-container">
    <h2>Customer Registration Form</h2>
    <form>
        <div class="form-group">
            <label for="name">Full Name:</label>
            <input type="text" class="form-control" id="name" placeholder="Enter your full name" required>
        </div>

        <div class="form-group">
            <label for="contact">Contact Number:</label>
            <input type="tel" class="form-control" id="contact" placeholder="Enter your contact number" required>
        </div>

        <div class="form-group">
            <label>Gender:</label><br>
            <div class="form-check">
                <input class="form-check-input" type="radio" name="gender" id="male" value="male" required>
                <label class="form-check-label" for="male">Male</label>
            </div>
            <div class="form-check">
                <input class="form-check-input" type="radio" name="gender" id="female" value="female" required>
                <label class="form-check-label" for="female">Female</label>
            </div>
            <div class="form-check">
                <input class="form-check-input" type="radio" name="gender" id="other" value="other" required>
                <label class="form-check-label" for="other">Other</label>
            </div>
```

```
    </div>

    <div class="form-group">
      <label for="preferredDays">Preferred Days of Purchasing:</label><br>
      <select class="form-control" id="preferredDays" multiple required>
        <option value="Monday">Monday</option>
        <option value="Tuesday">Tuesday</option>
        <option value="Wednesday">Wednesday</option>
        <option value="Thursday">Thursday</option>
        <option value="Friday">Friday</option>
        <option value="Saturday">Saturday</option>
        <option value="Sunday">Sunday</option>
      </select>
      <small class="form-text text-muted">Hold Ctrl (Windows) or Command (Mac) to select multiple
days.</small>
    </div>

    <div class="form-group">
      <label for="favoriteItem">Favorite Item:</label>
      <select class="form-control" id="favoriteItem" required>
        <option value="">Select your favorite item</option>
        <option value="Fruits">Fruits</option>
        <option value="Vegetables">Vegetables</option>
        <option value="Dairy Products">Dairy Products</option>
        <option value="Snacks">Snacks</option>
        <option value="Beverages">Beverages</option>
        <option value="Personal Care">Personal Care</option>
      </select>
    </div>

    <div class="form-group">
      <label for="suggestions">Suggestions:</label>
      <textarea class="form-control" id="suggestions" rows="3" placeholder="Your suggestions"></textarea>
    </div>

    <button type="submit" class="btn btn-primary">Submit</button>
    <button type="reset" class="btn btn-secondary">Reset</button>
  </form>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>
</html>
```

Explanation:
- Form Structure : The form includes the following fields:
  - Full Name : A text input for the customer's name.
  - Contact Number : A telephone input for the customer's contact number.
  - Gender : Radio buttons for selecting gender.

- Preferred Days of Purchasing : A multiple select dropdown allowing users to select their preferred shopping days.
- Favorite Item : A dropdown list to select a favorite item from predefined options.
- Suggestions : A textarea for additional comments or suggestions.
- Buttons : There are two buttons: one to submit the form and another to reset all fields.
- Bootstrap : The form uses Bootstrap for styling, ensuring a responsive and visually appealing design.

1. Model the following online movie information as a document database. Consider online movie information where the each actor has acted in one or more movie. Each producer has produced many movies but each movie can be produced by more than one producers. Each movie has one or more actors acting in it, in different roles.
2. Assume appropriate attributes and collections as per the query requirements[3]
3. Insert at least 05 documents in each collection. [3]
4. Answer the following Queries.

a. List the names of movies with the highest budget. [3] b. Display the details of producer who have produced more than
one movie in a year. [3]
c. List the names of actors who have acted in at least one movie, in which 'Akshay' has acted. [4]
d. List the names of movies, produced by more than one produce. [4]

To model an online movie information system as a document database, we can create collections for movies, actors, and producers. This structure captures the relationships among these entities and allows for efficient querying.

Collections

1. Movies
   - Attributes:
     - `movie_id`: Unique identifier for the movie
     - `title`: Title of the movie
     - `budget`: Budget of the movie
     - `release_year`: Year the movie was released
     - `actors`: List of actor IDs and their roles in the movie
     - `producers`: List of producer IDs

2. Actors
   - Attributes:
     - `actor_id`: Unique identifier for the actor
     - `name`: Name of the actor
     - `dob`: Date of birth of the actor
     - `movies`: List of movie IDs in which the actor has acted

3. Producers
   - Attributes:
     - `producer_id`: Unique identifier for the producer
     - `name`: Name of the producer
     - `movies`: List of movie IDs produced by the producer

Sample Documents

Movies Collection

```json
[
  {"movie_id": 1, "title": "Action Heroes", "budget": 150000000, "release_year": 2022, "actors": [{"actor_id": 1, "role": "Lead"}, {"actor_id": 2, "role": "Supporting"}], "producers": [1, 2]},
  {"movie_id": 2, "title": "Mystery Nights", "budget": 80000000, "release_year": 2021, "actors": [{"actor_id": 1, "role": "Lead"}, {"actor_id": 3, "role": "Supporting"}], "producers": [2]},
```

```
  {"movie_id": 3, "title": "Comedy Central", "budget": 60000000, "release_year": 2022, "actors": [{"actor_id": 4, "role": "Lead"}, {"actor_id": 5, "role": "Supporting"}], "producers": [1, 3]},
  {"movie_id": 4, "title": "Romantic Days", "budget": 50000000, "release_year": 2023, "actors": [{"actor_id": 2, "role": "Lead"}, {"actor_id": 5, "role": "Supporting"}], "producers": [3]},
  {"movie_id": 5, "title": "Thriller Nights", "budget": 120000000, "release_year": 2021, "actors": [{"actor_id": 1, "role": "Lead"}, {"actor_id": 3, "role": "Supporting"}], "producers": [2, 3]}
]
```

Actors Collection

```json
[
  {"actor_id": 1, "name": "Akshay Kumar", "dob": "1967-09-09", "movies": [1, 2, 5]},
  {"actor_id": 2, "name": "Kareena Kapoor", "dob": "1980-09-21", "movies": [1, 4]},
  {"actor_id": 3, "name": "Ranveer Singh", "dob": "1985-07-06", "movies": [2, 5]},
  {"actor_id": 4, "name": "Alia Bhatt", "dob": "1993-03-15", "movies": [3]},
  {"actor_id": 5, "name": "Varun Dhawan", "dob": "1987-04-24", "movies": [3, 4]}
]
```

Producers Collection

```json
[
  {"producer_id": 1, "name": "Rajesh Sharma", "movies": [1, 3]},
  {"producer_id": 2, "name": "Anil Kapoor", "movies": [1, 2, 5]},
  {"producer_id": 3, "name": "Suresh Gupta", "movies": [3, 4, 5]},
  {"producer_id": 4, "name": "Neha Singh", "movies": [2]},
  {"producer_id": 5, "name": "Vikram Mehta", "movies": []}
]
```

Queries

a. List the names of movies with the highest budget

```json
const highestBudget = db.movies.find().sort({ budget: -1 }).limit(1).toArray()[0];
db.movies.find({ budget: highestBudget.budget }, { title: 1 })
```

b. Display the details of producers who have produced more than one movie in a year

```json
db.producers.aggregate([
  { $unwind: "$movies" },
  { $lookup: {
    from: "movies",
    localField: "movies",
    foreignField: "movie_id",
    as: "movie_details"
  }},
  { $unwind: "$movie_details" },
```

```
  { $group: {
    _id: { producer_id: "$producer_id", year: "$movie_details.release_year" },
    count: { $sum: 1 },
    producer_name: { $first: "$name" }
  }},
  { $match: { count: { $gt: 1 } } },
  { $project: { producer_name: 1, year: "$_id.year", count: 1 } }
])
```

c. List the names of actors who have acted in at least one movie in which ‘Akshay’ has acted

```json
const akshayMovies = db.actors.findOne({ name: "Akshay Kumar" }).movies;
db.actors.find({ movies: { $in: akshayMovies } }, { name: 1 })
```

d. List the names of movies produced by more than one producer

```json
db.movies.find({ "producers": { $size: { $gt: 1 } } }, { title: 1 })
```

Slip no 13

Create a useful web with the following information and structure using
HTML5 tags like:<header> , <footer>, <nav>, <aside>, <section>

Certainly! Below is an example of a simple web page structured with HTML5 semantic elements such as `<header>`, `<footer>`, `<nav>`, `<aside>`, and `<section>`. This web page provides general information about a fictional company, including navigation links, a main content area, and an aside for additional information.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sample Company Web Page</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <style>
        body {
            background-color:   f4f4f4;
        }
        .content {
            padding: 20px;
            background-color: white;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
    </style>
</head>
<body>

<header class="bg-primary text-white text-center p-4">
    <h1>Welcome to Our Company</h1>
    <p>Your trusted partner in excellence</p>
</header>

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="  ">Company Logo</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="  navbarNav" aria-
controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav">
            <li class="nav-item active">
                <a class="nav-link" href="  about">About Us</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="  services">Services</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="  contact">Contact</a>
            </li>
        </ul>
```

```
    </div>
</nav>

<div class="container mt-4">
    <section id="about" class="content">
        <h2>About Us</h2>
        <p>We are a leading company in our industry, dedicated to providing top-notch services to our clients. Our team is comprised of experienced professionals who are passionate about delivering results.</p>
    </section>

    <section id="services" class="content mt-4">
        <h2>Our Services</h2>
        <ul>
            <li>Consulting</li>
            <li>Project Management</li>
            <li>Technical Support</li>
            <li>Product Development</li>
        </ul>
    </section>

    <aside class="content mt-4">
        <h2>Additional Information</h2>
        <p>Follow us on social media for updates:</p>
        <ul>
            <li><a href="  ">Facebook</a></li>
            <li><a href="  ">Twitter</a></li>
            <li><a href="  ">LinkedIn</a></li>
        </ul>
    </aside>
</div>

<footer class="bg-dark text-white text-center p-4 mt-4">
    <p>&amp;copy; 2024 Our Company. All rights reserved.</p>
    <p><a href="  " class="text-white">Privacy Policy</a> | <a href="  " class="text-white">Terms of Service</a></p>
</footer>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>
</html>
```

Explanation of Structure:

- Header : Contains the company name and a brief tagline, styled with a background color and centered text.
- Navigation : A responsive navigation bar with links to different sections of the page (About Us, Services, Contact).
- Main Content Area :
  - Sections :
    - The first section (`About Us`) describes the company.
    - The second section (`Our Services`) lists the services offered.
  - Aside : Provides additional information, such as social media links.
- Footer : Contains copyright information and links to the privacy policy and terms of service.

Styles:
- The page uses Bootstrap for responsive design and basic styling, ensuring it looks good on various devices.
- Custom styles are added to enhance the appearance of the content sections.

1. Model the following Student Competition information as a document database. Consider Student competition information where the student can participate in many competitions.
2. Assume appropriate attributes and collections as per the query requirements [3]
3. Insert at least 10 documents in each collection. [3]
4. Answer the following Queries.
a. Display average no. of students participating in each competition.[3]
b. Find the number of student for programming competition. [3]
c. Display the names of first three winners of each competition. [4]
d. Display students from class 'FY' and participated in 'E-Rangoli ' Competition. [4]

To model a Student Competition information system as a document database, we can create collections for students, competitions, and participations. This structure captures the relationships among these entities and allows for efficient querying.

Collections

1. Students
   - Attributes:
     - `student_id`: Unique identifier for the student
     - `name`: Name of the student
     - `class`: Class of the student (e.g., FY, SY)
     - `email`: Email of the student

2. Competitions
   - Attributes:
     - `competition_id`: Unique identifier for the competition
     - `name`: Name of the competition
     - `date`: Date of the competition
     - `category`: Category of the competition (e.g., Programming, Art)
     - `participants`: List of student IDs who participated

3. Participations (optional but useful for tracking individual results)
   - Attributes:
     - `participation_id`: Unique identifier for the participation record
     - `competition_id`: Reference to the competition
     - `student_id`: Reference to the student
     - `position`: Position in the competition (e.g., 1st, 2nd, 3rd)

Sample Documents

Students Collection

```json
[
  {"student_id": 1, "name": "Alice Johnson", "class": "FY", "email": "alice@example.com"},
  {"student_id": 2, "name": "Bob Smith", "class": "SY", "email": "bob@example.com"},
  {"student_id": 3, "name": "Charlie Brown", "class": "FY", "email": "charlie@example.com"},
  {"student_id": 4, "name": "Diana Prince", "class": "TY", "email": "diana@example.com"},
```

```
  {"student_id": 5, "name": "Eve Adams", "class": "FY", "email": "eve@example.com"},
  {"student_id": 6, "name": "Frank Castle", "class": "SY", "email": "frank@example.com"},
  {"student_id": 7, "name": "Grace Hopper", "class": "FY", "email": "grace@example.com"},
  {"student_id": 8, "name": "Harry Potter", "class": "TY", "email": "harry@example.com"},
  {"student_id": 9, "name": "Irene Adler", "class": "FY", "email": "irene@example.com"},
  {"student_id": 10, "name": "Jack Sparrow", "class": "SY", "email": "jack@example.com"}
]
```

Competitions Collection

```json
[
  {"competition_id": 1, "name": "E-Rangoli", "date": "2023-08-15", "category": "Art", "participants": [1, 3, 5, 7, 9]},
  {"competition_id": 2, "name": "Programming Contest", "date": "2023-09-01", "category": "Programming",
"participants": [2, 4, 6, 8]},
  {"competition_id": 3, "name": "Science Quiz", "date": "2023-09-15", "category": "Quiz", "participants": [1, 2, 5]},
  {"competition_id": 4, "name": "Math Olympiad", "date": "2023-10-01", "category": "Math", "participants": [3, 4, 7,
9]},
  {"competition_id": 5, "name": "Art Competition", "date": "2023-10-10", "category": "Art", "participants": [1, 5, 6]},
  {"competition_id": 6, "name": "Debate Competition", "date": "2023-10-20", "category": "Debate", "participants": [2,
3, 8]},
  {"competition_id": 7, "name": "Spelling Bee", "date": "2023-11-01", "category": "Quiz", "participants": [1, 2, 4]},
  {"competition_id": 8, "name": "Robotics Challenge", "date": "2023-11-10", "category": "Tech", "participants": [5, 6,
9]},
  {"competition_id": 9, "name": "Coding Challenge", "date": "2023-11-15", "category": "Programming", "participants":
[1, 2, 8]},
  {"competition_id": 10, "name": "Cultural Fest", "date": "2023-12-01", "category": "Cultural", "participants": [3, 7,
10]}
]
```

Participations Collection

```json
[
  {"participation_id": 1, "competition_id": 1, "student_id": 1, "position": "1st"},
  {"participation_id": 2, "competition_id": 1, "student_id": 3, "position": "2nd"},
  {"participation_id": 3, "competition_id": 1, "student_id": 5, "position": "3rd"},
  {"participation_id": 4, "competition_id": 2, "student_id": 2, "position": "1st"},
  {"participation_id": 5, "competition_id": 2, "student_id": 6, "position": "2nd"},
  {"participation_id": 6, "competition_id": 2, "student_id": 8, "position": "3rd"},
  {"participation_id": 7, "competition_id": 3, "student_id": 1, "position": "1st"},
  {"participation_id": 8, "competition_id": 3, "student_id": 5, "position": "2nd"},
  {"participation_id": 9, "competition_id": 4, "student_id": 3, "position": "1st"},
  {"participation_id": 10, "competition_id": 4, "student_id": 9, "position": "2nd"}
]
```

Queries

a. Display average number of students participating in each competition

```json
db.competitions.aggregate([
  { $project: { name: "$name", average_participants: { $avg: { $size: "$participants" } } } }
```

])
```

**b. Find the number of students for the Programming Contest**

```json
db.competitions.find({ name: "Programming Contest" }, { "participants": { $size: 1 } })
```

c. Display the names of the first three winners of each competition

```json
db.participations.aggregate([
  { $group: {
      _id: "$competition_id",
      winners: { $push: { student_id: "$student_id", position: "$position" } }
  }},
  { $project: {
      _id: 1,
      winners: { $slice: ["$winners", 3] }
  }},
  { $lookup: {
      from: "students",
      localField: "winners.student_id",
      foreignField: "student_id",
      as: "winner_details"
  }},
  { $unwind: "$winner_details" },
  { $project: {
      competition_id: "$_id",
      winner_name: "$winner_details.name",
      position: "$winners.position"
  }}
])
```

d. Display students from class 'FY' who participated in 'E-Rangoli' Competition

```json
const eRangoliId = db.competitions.findOne({ name: "E-Rangoli" }).competition_id;
db.students.find({ class: "FY", student_id: { $in: db.competitions.findOne({ competition_id: eRangoliId }).participants
} })
```
```

Slip: 14

Design an HTML form to take the information of a customer for booking a travel plan consisting of fields such as name, address, contact no., gender, preferred season(Checkboxes), location type(to be selected from a list) etc. You should provide button to submit as well as reset the form contents

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Travel Booking Form</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <style>
        body {
            background-color:   f4f4f4;
            padding: 20px;
        }
        .form-container {
            background-color: white;
            padding: 20px;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
    </style>
</head>
<body>

<div class="container form-container">
    <h2>Travel Booking Form</h2>
    <form>
        <div class="form-group">
            <label for="name">Full Name:</label>
            <input type="text" class="form-control" id="name" placeholder="Enter your full name" required>
        </div>

        <div class="form-group">
            <label for="address">Address:</label>
            <textarea class="form-control" id="address" rows="3" placeholder="Enter your address" required></textarea>
        </div>

        <div class="form-group">
            <label for="contact">Contact Number:</label>
            <input type="tel" class="form-control" id="contact" placeholder="Enter your contact number" required>
        </div>

        <div class="form-group">
            <label>Gender:</label><br>
            <div class="form-check">
                <input class="form-check-input" type="radio" name="gender" id="male" value="male" required>
                <label class="form-check-label" for="male">Male</label>
            </div>
            <div class="form-check">
                <input class="form-check-input" type="radio" name="gender" id="female" value="female" required>
                <label class="form-check-label" for="female">Female</label>
```

```
          </div>
          <div class="form-check">
            <input class="form-check-input" type="radio" name="gender" id="other" value="other" required>
            <label class="form-check-label" for="other">Other</label>
          </div>
        </div>

        <div class="form-group">
          <label>Preferred Season:</label><br>
          <div class="form-check">
            <input class="form-check-input" type="checkbox" id="spring" value="Spring">
            <label class="form-check-label" for="spring">Spring</label>
          </div>
          <div class="form-check">
            <input class="form-check-input" type="checkbox" id="summer" value="Summer">
            <label class="form-check-label" for="summer">Summer</label>
          </div>
          <div class="form-check">
            <input class="form-check-input" type="checkbox" id="autumn" value="Autumn">
            <label class="form-check-label" for="autumn">Autumn</label>
          </div>
          <div class="form-check">
            <input class="form-check-input" type="checkbox" id="winter" value="Winter">
            <label class="form-check-label" for="winter">Winter</label>
          </div>
        </div>

        <div class="form-group">
          <label for="locationType">Location Type:</label>
          <select class="form-control" id="locationType" required>
            <option value="">Select a location type</option>
            <option value="beach">Beach</option>
            <option value="mountain">Mountain</option>
            <option value="city">City</option>
            <option value="countryside">Countryside</option>
            <option value="desert">Desert</option>
          </select>
        </div>

        <button type="submit" class="btn btn-primary">Submit</button>
        <button type="reset" class="btn btn-secondary">Reset</button>
      </form>
    </div>

    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

    </body>
    </html>
    ```

Explanation:

- Form Structure :

- Full Name : A text input field for the customer's name.
- Address : A textarea for the customer to enter their address.
- Contact Number : A telephone input for the customer's contact number.
- Gender : Radio buttons for selecting the customer's gender.
- Preferred Season : Checkboxes for selecting preferred travel seasons.
- Location Type : A dropdown menu to select the type of location.
- Buttons : There are two buttons:
- Submit : To submit the form data.
- Reset : To clear all fields in the form.
- Bootstrap : The form uses Bootstrap for responsive design and styling, ensuring a clean and modern appearance.

Model the following system as a graph model, and answer the queries using Cypher. Government provides various scholarships for students. A students apply for scholarship. Student can get benefits of more than one scholarship if satisfies the criteria. Student can recommend it for his friends or other family members

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

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

3. Answer the following Queries in Cypher:

a. List the names of scholarship for OBC category. [3]

b. Count no. of students who are benefitted by ____ scholarship in year 2020-2021 [3]

c. Update the income limit for _____scholarship. [4]

d. List the most popular scholarship. [4]

To model the scholarship system as a graph, let's define the structure:

1. Graph Model

Labels:

- Student
  - Properties: `name`, `category`, `income`

- Scholarship
  - Properties: `name`, `category`, `income_limit`, `year`

- Recommendation
  - Properties: `date`, `status`

Relationships:
- `APPLIED_FOR` (between `Student` and `Scholarship`)
- `BENEFITED_FROM` (between `Student` and `Scholarship`)
- `RECOMMENDED` (between `Student` and `Student`)

High-Level Graph Model

(Student)-[:APPLIED_FOR]->(Scholarship)
(Student)-[:BENEFITED_FROM]->(Scholarship)
(Student)-[:RECOMMENDED]->(Student)

2. Create Nodes and Relationships

Students

```
CREATE (s1:Student {name: 'Alice', category: 'OBC', income: 30000}),
    (s2:Student {name: 'Bob', category: 'General', income: 40000}),
    (s3:Student {name: 'Charlie', category: 'OBC', income: 25000}),
    (s4:Student {name: 'David', category: 'SC', income: 20000});
```

Scholarships

```
CREATE (sch1:Scholarship {name: 'OBC Scholarship', category: 'OBC', income_limit: 35000, year: '2020-2021'}),
    (sch2:Scholarship {name: 'General Scholarship', category: 'General', income_limit: 50000, year: '2020-2021'}),
    (sch3:Scholarship {name: 'SC Scholarship', category: 'SC', income_limit: 25000, year: '2020-2021'});
```

// **Create Relationships**

```
CREATE (s1)-[:APPLIED_FOR]->(sch1),
    (s1)-[:BENEFITED_FROM]->(sch1),
    (s2)-[:APPLIED_FOR]->(sch2),
    (s3)-[:APPLIED_FOR]->(sch1),
    (s3)-[:BENEFITED_FROM]->(sch1),
    (s4)-[:APPLIED_FOR]->(sch3),
    (s4)-[:BENEFITED_FROM]->(sch3),
    (s1)-[:RECOMMENDED]->(s2),
    (s1)-[:RECOMMENDED]->(s3);
```

3. Cypher Queries

a. List the names of scholarships for the OBC category.

```
MATCH (s:Scholarship {category: 'OBC'})
RETURN s.name;
```

b. Count the number of students who benefited from a specific scholarship in the year 2020-2021. (Assuming we check for 'OBC Scholarship')

```
MATCH (s:Scholarship {name: 'OBC Scholarship', year: '2020-2021'})<-[:BENEFITED_FROM]-(student:Student)
RETURN COUNT(student) AS number_of_students;
```

c. Update the income limit for a specific scholarship. (Assuming we update 'OBC Scholarship' to a new limit)

```
MATCH (s:Scholarship {name: 'OBC Scholarship'})
SET s.income_limit = 40000;
```

d. List the most popular scholarship. (Based on the count of students benefiting)

```
MATCH (s:Scholarship)<-[:BENEFITED_FROM]-(student:Student)
RETURN s.name, COUNT(student) AS number_of_students
ORDER BY number_of_students DESC
LIMIT 1;
```

Slip-15:

Model the following movie system as a Graph database.
Consider a information about of movie and actors. One movie can have more than one actor
1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model. [3]
2. Create nodes and relationships, along with their properties, and visualize your actual Graph model. [3]
3. Answer the Queries:
a. Find movie which made highest business. [3]
b. Display details of movie along with actors. [3]
c. List all the movie of "Shahrukh Khan". [4]
d. Display all movie having more than 2 awards received [4]

To model a movie system as a graph database, let's break down the requirements step by step.
NODE:

Movie
Properties: `title`, `releaseYear`, `boxOffice`, `awards`
Actor
Properties: `name`, `dob`, `gender`, `nationality`
Relationships:

ACTED_IN: Connects `Actor` to `Movie`
Properties: `role`
(Actor) -[:ACTED_IN {role: "lead"}]-> (Movie)

Creating Nodes and Relationships:

Create Movies

```
CREATE
(m1:Movie {title: "Dilwale Dulhania Le Jayenge", releaseYear: 1995, boxOffice: 1000, awards: 10}),
(m2:Movie {title: "Chennai Express", releaseYear: 2013, boxOffice: 3000, awards: 5}),
(m3:Movie {title: "My Name is Khan", releaseYear: 2010, boxOffice: 2000, awards: 15}),
(m4:Movie {title: "Raees", releaseYear: 2017, boxOffice: 1500, awards: 2});
```

Create Actors

```
CREATE
(a1:Actor {name: "Shahrukh Khan", dob: "1965-11-02", gender: "Male", nationality: "Indian"}),
(a2:Actor {name: "Kajol", dob: "1974-08-05", gender: "Female", nationality: "Indian"}),
(a3:Actor {name: "Deepika Padukone", dob: "1986-01-05", gender: "Female", nationality: "Indian"}),
(a4:Actor {name: "Mahira Khan", dob: "1984-12-21", gender: "Female", nationality: "Pakistani"});
```

Create Relationships

```
CREATE
(a1)-[:ACTED_IN {role: "lead"}]->(m1),
(a1)-[:ACTED_IN {role: "lead"}]->(m2),
(a1)-[:ACTED_IN {role: "lead"}]->(m3),
(a1)-[:ACTED_IN {role: "lead"}]->(m4),
(a2)-[:ACTED_IN {role: "lead"}]->(m1),
(a3)-[:ACTED_IN {role: "lead"}]->(m2),
(a4)-[:ACTED_IN {role: "lead"}]->(m3);
```

Queries

a. Find movie which made highest business.

```
MATCH (m:Movie)
```

```
RETURN m.title AS MovieTitle, m.boxOffice AS BoxOffice
ORDER BY m.boxOffice DESC
LIMIT 1;
```

b. Display details of movie along with actors.

```
MATCH (m:Movie)<-[:ACTED_IN]-(a:Actor)
RETURN m.title AS MovieTitle, m.releaseYear AS ReleaseYear, m.boxOffice AS BoxOffice,
    m.awards AS Awards, collect(a.name) AS Actors;
```

c. List all the movies of “Shahrukh Khan”.

```
MATCH (a:Actor {name: "Shahrukh Khan"})-[:ACTED_IN]->(m:Movie)
RETURN m.title AS MovieTitle, m.releaseYear AS ReleaseYear, m.boxOffice AS BoxOffice;
```

d. Display all movies having more than 2 awards received.

```
MATCH (m:Movie)
WHERE m.awards > 2
RETURN m.title AS MovieTitle, m.awards AS Awards;
```

Slip-16 :

Model the following food service industry information as a graph model, and answer the following queries using Cypher.
Consider food service industries like ZOMATO, Swiggy around us. Popular restaurants are connected to these industries to increase sell. A person order food through this industry and get offers. A person give rate(1-5 stars) to company its facility/facilities. and can recommend this to his/her friends.

1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model. [3]
2. Create nodes and relationships, along with their properties, and visualize your actual Graph model. [3]
3. Answer the Queries.

a. Count no. of customers who place order on "1/1/2023" [3]
b. List the names of customers whose name starts with S and place order using Swiggy [3]
c. List the names of hotels with high rating (>=4). [4]
d. List the most recommended hotels in……. area. [4]

To model the food service industry in Neo4j and answer your queries, let's outline the labels, relationships, properties, and then provide the Cypher queries.

1. Graph Model

Labels and Relationships

Nodes (Labels):

Person: Represents customers.
- Properties: `name`, `customer_id`
- **Restaurant**: Represents popular restaurants.
- Properties: `name`, `restaurant_id`, `location`
- **FoodService**: Represents services like Zomato or Swiggy.
- Properties: `name`, `service_id`
- **Order**: Represents orders placed by customers.
- Properties: `order_id`, `date`, `offer`
- **Rating**: Represents ratings given by customers to food services.
- Properties: `stars`, `comment`

- Relationships:
- **PLACED**: (Person) -[PLACED]-> (Order)
- **ORDERED_FROM**: (Order) -[ORDERED_FROM]-> (FoodService)
- **RECOMMENDED**: (Person) -[RECOMMENDED]-> (Restaurant)
- **RATED**: (Person) -[RATED]-> (FoodService)

2. Nodes and Relationships
Sample Nodes Creation

Person

```
CREATE
(p1:Person {name: 'Sam', customer_id: 'C001'}),
(p2:Person {name: 'Sonia', customer_id: 'C002'}),
(p3:Person {name: 'Alex', customer_id: 'C003'}),
(p4:Person {name: 'Martin', customer_id: 'C004'}),
(p5:Person {name: 'Jimmy', customer_id: 'C005'}),
(p6:Person {name: 'Dinel', customer_id: 'C006'})
```

Restaurant

```
Create
(r1:Restaurant {name: 'Pizza Place', restaurant_id: 'R001', location: 'Chinchwad'}),
(r2:Restaurant {name: 'Burger House', restaurant_id: 'R002', location: 'Nigdi'}),
(r3:Restaurant {name: 'My Cafe', restaurant_id: 'R003', location: 'Nigdi'}),
(r4:Restaurant {name: 'Delta', restaurant_id: 'R004', location: 'Pimpri'})
```

FOOD Services

```
Create (fs1:FoodService {name: 'Swiggy', service_id: 'FS001'}), (fs2:FoodService {name: 'Zomato', service_id:
'FS002'}), (fs3:FoodService {name: 'Foodpanda', service_id: 'FS003'}), (fs4:FoodService {name: 'Doordash',
service_id: 'FS004'})
```

Order

```
 Create
(o1:Order {order_id: 'O001', date: '2023-01-01', offer: '10% OFF'}), (o2:Order {order_id: 'O002', date: '2023-01-01',
offer: '20% OFF'}), (o3:Order {order_id: 'O003', date: '2023-01-02', offer: '10% OFF'}), (o4:Order {order_id: 'O004',
date: '2023-01-02', offer: '20 OFF'})
```

Relations

```
Create  (r1)<-[:RECOMMENDED]-(p2),
 (p1)-[:PLACED]->(o1),
 (o1)-[:ORDERED_FROM]->(fs1),
 (p1)-[:RATED {stars: 5, comment: 'Great service!'}]->(fs1)
Create  ('Pizza Place')<-[:RECOMMENDED]-('Sonia'),
 (p1)-[:PLACED]->(o1),
 (o1)-[:ORDERED_FROM]->(fs1),
 (p1)-[:RATED {stars: 5, comment: 'Great service!'}]->(fs1)
```

OR

```
CREATE
(p1:Person {name: 'Sam', customer_id: 'C001'}),
(p2:Person {name: 'Sonia', customer_id: 'C002'}),
(p3:Person {name: 'Alex', customer  id: 'C003'}),
(p4:Person {name: 'Martin', customer_id: 'C004'}),
(r1:Restaurant {name: 'Pizza Place', restaurant_id: 'R001', location: 'Chinchwad'}), (r2:Restaurant {name: 'Burger
House', restaurant_id: 'R002', location: 'Nigdi'}), (fs1:FoodService {name: 'Swiggy', service_id: 'FS001'}),
(fs2:FoodService {name: 'Zomato', service_id: 'FS002'}),
(o1:Order {order_id: 'O001', date: '2023-01-01', offer: '10% OFF'}),
 (o2:Order {order_id: 'O002', date: '2023-01-01', offer: '20% OFF'}),
(r1)<-[:RECOMMENDED]-(p2),
(p1)-[:PLACED]->(o1),
(o1)-[:ORDERED_FROM]->(fs1),
(p1)-[:RATED {stars: 5, comment: 'Great service!'}]->(fs1)
```

3. Cypher Queries

a. Count the number of customers who placed orders on "1/1/2023"

```
MATCH (o:Order {date: '2023-01-01'})<-[:PLACED]-(p:Person)
RETURN COUNT(DISTINCT p) AS customer_count;
```

b. List the names of customers whose name starts with S and placed orders using Swiggy

```
MATCH (p:Person)-[:PLACED]->(o:Order)-[:ORDERED_FROM]->(fs:FoodService {name: 'Swiggy'})
WHERE p.name STARTS WITH 'S'
RETURN p.name AS customer_names;
```

c. List the names of restaurants with high ratings (>= 4)

```
MATCH (p:Person)-[:RATED]->(fs:FoodService)
WHERE fs.stars >= 4 RETURN DISTINCT fs.name AS high_rated_restaurants;
```

d. List the most recommended restaurants in a specified area (replace 'AreaName' with the actual area)

```
MATCH (p:Person)-[:RECOMMENDED]->(r:Restaurant {location: 'Chinchwad'}) RETURN r.name AS
recommended_restaurants;
```

Slip no. 17
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]
2. Create nodes and relationships, along with their properties, and visualize your actual Graph model.[3]
3. Answer the Queries
a. List the names of authors who wrote "Comics". [3]
b. Count no. of readers of _____book published by "Sage". [3]
c. List all the publisher whose name starts with "N" [4]
d. List the names of people who have given a rating of (>=3) for _____ book [4]

1. Graph Model

Labels and Relationships

- Nodes (Labels):
Author: Represents authors of books.
- Properties: `name`, `author_id`
Book: Represents books written by authors.
- Properties: `title`, `book_id`, `genre`
Publisher: Represents publishers of books.
- Properties: `name`, `publisher_id`
Reader: Represents readers who read books.
- Properties: `name`, `reader_id`
Review: Represents reviews given by readers for books.
- Properties: `rating`, `comment`

- Relationships:
WROTE: (Author) -[WROTE]-> (Book)
PUBLISHED_BY: (Book) -[PUBLISHED_BY]-> (Publisher)
READS: (Reader) -[READS]-> (Book)
GAVE_REVIEW: (Reader) -[GAVE_REVIEW]-> (Review)
FOR: (Review) -[FOR]-> (Book)

2. Nodes and Relationships Creation

```
CREATE
(a1:Author {name: 'John Doe', author_id: 'A001'}),
(a2:Author {name: 'Jane Smith', author_id: 'A002'}),

(b1:Book {title: 'Amazing Comics', book_id: 'B001', genre: 'Comics'}),
(b2:Book {title: 'Mystery Novels', book_id: 'B002', genre: 'Mystery'}),

(p1:Publisher {name: 'Sage', publisher_id: 'P001'}),
(p2:Publisher {name: 'NextGen', publisher_id: 'P002'}),

(r1:Reader {name: 'Alice', reader_id: 'R001'}),
(r2:Reader {name: 'Bob', reader_id: 'R002'}),

(a1)-[:WROTE]->(b1),
(a2)-[:WROTE]->(b2),
```

```
(b1)-[:PUBLISHED_BY]->(p1),
(b2)-[:PUBLISHED_BY]->(p2),

(r1)-[:READS]->(b1),
(r2)-[:READS]->(b1),

(r1)-[:GAVE_REVIEW]->(review1:Review {rating: 4, comment: 'Great read!'}),
(r2)-[:GAVE_REVIEW]->(review2:Review {rating: 3, comment: 'Interesting.'}),

(review1)-[:FOR]->(b1),
(review2)-[:FOR]->(b1)
```

3. Cypher Queries

a. List the names of authors who wrote “Comics”

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

b. Count the number of readers of a specific book published by “Sage” (replace 'Amazing Comics' with your book title)

```
MATCH (b:Book {title: 'Amazing Comics'})<-[:READS]-(r:Reader)-[:PUBLISHED_BY]->(p:Publisher {name:
'Sage'}) RETURN COUNT(r) AS number_of_readers;
```

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

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

d. List the names of people who have given a rating of (>=3) for a specific book (replace 'Amazing Comics' with your book title)

```
MATCH (r:Reader)-[:GAVE_REVIEW]->(review:Review)-[:FOR]->(b:Book {title: 'Amazing Comics'})
WHERE review.rating >= 3
RETURN r.name AS readers_with_high_rating;
```

Slip no. 18
Create a web page, place an image in center and apply 2d transformation on it. (rotation, scaling, translation)

Here's an example of a simple web page that centers an image and applies 2D transformations such as rotation, scaling, and translation using CSS. You can copy this code into an HTML file and open it in a browser to see the effects.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>2D Transformations on Image</title>
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            background-color:  f4f4f4;
        }

        .image-container {
            width: 200px; /  Set container width  /
            height: 200px; /  Set container height  /
            overflow: hidden; /  Hide overflow  /
            position: relative; /  For absolute positioning  /
        }

        img {
            width: 100%; /  Responsive width  /
            height: auto; /  Maintain aspect ratio  /
            position: absolute; /  Position for transformations  /
            animation: transformAnimation 5s infinite; /  Animation  /
        }

        @keyframes transformAnimation {
            0% {
                transform: translate(0, 0) rotate(0deg) scale(1);
            }
            25% {
                transform: translate(20px, 20px) rotate(45deg) scale(1.2);
            }
            50% {
                transform: translate(40px, 0) rotate(90deg) scale(1.5);
            }
            75% {
                transform: translate(20px, -20px) rotate(135deg) scale(1.2);
            }
            100% {
                transform: translate(0, 0) rotate(180deg) scale(1);
            }
```

```
        }
    </style>
</head>
<body>

<div class="image-container">
    <img src="https://via.placeholder.com/200" alt="Placeholder Image">
</div>

</body>
</html>
```

Explanation:

- HTML Structure :
  - A `div` element (`image-container`) is used to contain the image, allowing it to be centered on the page.
  - An `img` tag is used to display a placeholder image.

- CSS Styles :
  - Flexbox : The `body` uses Flexbox to center the `image-container` both vertically and horizontally.
  - Image Container : A fixed width and height are set to ensure the image stays within the bounds.
  - Transformations : The `@keyframes` rule defines an animation that applies various transformations:
    - Translation : Moves the image along the x and y axes.
    - Rotation : Rotates the image in degrees.
    - Scaling : Changes the size of the image.
  - The animation lasts 5 seconds and repeats infinitely.

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 insome 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 adoctor.

1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model. [3]
2. Create nodes and relationships, along with their properties, and visualize your actual Graph model. [3]
3. Answer the Queries

a. List the Orthopedic doctors in ……. Area. [3]
b. List the doctors who has specialization in ____ [3]
c. List the most recommended Pediatrics in Seren Medows. [4]
d. List all the who visits more than 2 hospitals [4]

To model the Doctor's information system as a graph in Neo4j, we can define the necessary elements, create sample nodes and relationships, and then provide Cypher queries for specific queries.

1. Graph Model

Labels and Relationships

Nodes (Labels) :

Doctor : Represents doctors.
- Properties: `name`, `doctor_id`, `specialization`, `clinic_name`

Hospital : Represents hospitals.
- Properties: `name`, `hospital_id`, `location`

Person : Represents patients or people who can review doctors.
- Properties: `name`, `person_id`

Review : Represents reviews given by persons for doctors.
- Properties: `rating`, `comment`

Relationships :
SPECIALIZES_IN : (Doctor) -[SPECIALIZES_IN]-> (Specialization)
VISITS : (Doctor) -[VISITS]-> (Hospital)
RECOMMENDS : (Person) -[RECOMMENDS]-> (Doctor)
GAVE_REVIEW : (Person) -[GAVE_REVIEW]-> (Review)
FOR: (Review) -[FOR]-> (Doctor)

2. Nodes and Relationships Creation

Sample Nodes Creation

```
CREATE
(d1:Doctor {name: 'Dr. Smith', doctor_id: 'D001', specialization: 'Orthopedic'}), (d2:Doctor {name: 'Dr. John', doctor_id: 'D002', specialization: 'Pediatrics'}), (d3:Doctor {name: 'Dr. Emily', doctor_id: 'D003', specialization: 'Gynecologist'}), (d4:Doctor {name: 'Dr. Alice', doctor_id: 'D004', specialization: 'Heart Specialist'}),
(h1:Hospital {name: 'City Hospital', hospital_id: 'H001', location: 'Pune'}), (h2:Hospital {name: 'Seren Medows', hospital_id: 'H002', location: 'Pune'}), (h3:Hospital {name: 'Care Clinic', hospital_id: 'H003', location: 'Pune'}),
(p1:Person {name: 'Alice', person_id: 'P001'}),
(p2:Person {name: 'Bob', person_id: 'P002'}),
(d1)-[:VISITS]->(h1),
(d1)-[:VISITS]->(h2),
 (d2)-[:VISITS]->(h2),
 (d2)-[:VISITS]->(h3),
(d3)-[:VISITS]->(h1),
(p1)-[:RECOMMENDS]->(d2),
(p1)-[:GAVE_REVIEW]->(review1:Review {rating: 4, comment: 'Excellent doctor!'}), (p2)-[:GAVE_REVIEW]->(review2:Review {rating: 5, comment: 'Very caring.'}), (review1)-[:FOR]->(d2),
 (review2)-[:FOR]->(d1)
```

3. Cypher Queries

a. List the Orthopedic doctors in a specified area (replace 'AreaName' with the actual area)

```
MATCH (d:Doctor {specialization: 'Orthopedic'})-[:VISITS]->(h:Hospital {location: 'Pune'})
RETURN d.name AS orthopedic_doctors;
```

b. List the doctors who have specialization in a specified field (replace 'SpecializationType' with the actual type)

```
MATCH (d:Doctor {specialization: 'Heart Specialist'})
RETURN d.name AS doctors_with_specialization;
```

c. List the most recommended Pediatricians in Seren Medows

```
MATCH (d:Doctor {specialization: 'Pediatrics'})-[:VISITS]->(h:Hospital {name: 'Seren Medows'})
RETURN d.name AS recommended_pediatricians ORDER BY d.name
```

d. List all doctors who visit more than 2 hospitals

```
MATCH (d:Doctor)-[:VISITS]->(h:Hospital) WITH d, COUNT(h) AS hospital_count WHERE hospital_count >=2
RETURN d.name AS doctors_visiting_multiple_hospitals;
```

Slip 19

Create a web page in which show a button with a text “start download” , when click in start download a progress bar must be initialized with value 0 then increase by 10 in each second, change the color of progress bar after every three seconds..

Here’s an example of a web page that features a "Start Download" button. When clicked, it initializes a progress bar that increases its value by 10 every second and changes its color every three seconds. You can copy and paste this code into an HTML file and open it in a browser to see it in action.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Download Progress Bar</title>
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color:   f4f4f4;
            flex-direction: column;
        }
        button {
            padding: 10px 20px;
            font-size: 16px;
            cursor: pointer;
            margin-bottom: 20px;
        }
         progressContainer {
            width: 100%;
            background-color:   e0e0e0;
            border-radius: 5px;
            overflow: hidden;
        }
         progressBar {
            width: 0;
            height: 30px;
            background-color:   76c7c0;
            transition: width 0.5s;
        }
    </style>
</head>
<body>

<button id="startButton">Start Download</button>
<div id="progressContainer">
    <div id="progressBar"></div>
</div>

<script>
    const startButton = document.getElementById('startButton');
    const progressBar = document.getElementById('progressBar');
    let progressValue = 0;
```

```
    let interval;
    let colorIndex = 0;

    const colors = ['  76c7c0', '  ff9f40', '  ff5c5c', '  4caf50', '  2196f3'];

    startButton.addEventListener('click', () => {
        startButton.disabled = true; // Disable button
        progressValue = 0;
        progressBar.style.width = '0%';

        interval = setInterval(() => {
            if (progressValue < 100) {
                progressValue += 10;
                progressBar.style.width = progressValue + '%';
            } else {
                clearInterval(interval); // Clear interval when done
            }
        }, 1000);

        // Change color every 3 seconds
        setInterval(() => {
            if (progressValue < 100) {
                colorIndex = (colorIndex + 1) % colors.length;
                progressBar.style.backgroundColor = colors[colorIndex];
            }
        }, 3000);
    });
</script>

</body>
</html>
```

Explanation:

- HTML Structure :
  - A button labeled "Start Download."
  - A `div` for the progress bar, which contains another `div` that visually represents the progress.
- CSS Styles :
  - Basic styles for the body, button, and progress bar. The progress bar is styled to have a smooth transition effect.
- JavaScript Logic :
  - Button Click Event : When the button is clicked, it gets disabled to prevent multiple clicks.
  - Progress Bar Update : A set interval updates the width of the progress bar every second, increasing it by 10% until it reaches 100%.
  - Color Change : Another set interval changes the color of the progress bar every three seconds, cycling through an array of colors.

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 a the product.

1. Identify the labels and relationships, along with their properties, and draw a high-level Graph model. [3]
2. Create nodes and relationships, along with their properties, and visualize your actual Graph model.[3]
3. Answer the Queries

a. List the characteristics of…….. laptop. [3]

b. List the name of customers who bought a “DELL” company laptop [3]

c. List the customers who purchase a device on “26/01/2023” [4]

d. List the most recommended device.[4]

To model a Laptop Manufacturing Information System as a graph database, we can define labels for the primary entities and the relationships that connect them. Below is a detailed breakdown of the model, including queries to interact with the database using Cypher.

1. Identify Labels and Relationships

Labels

- **Laptop**
  - Properties: `laptop_id`, `brand`, `model`, `type`, `specifications`, `price`, `rating`
- Customer
  - Properties: `customer_id`, `name`, `email`, `purchase_date`

- **Recommendation**
  - Properties: `recommendation_id`, `rating`, `comment`

Relationships

- BOUGHT
  - From: Customer
  - To: Laptop
  - Properties: `purchase_date`

- RECOMMENDS
  - From: Customer
  - To: Laptop
  - Properties: `rating`, `comment`

High-Level Graph Model

```
(Customer)-[:BOUGHT {purchase_date: "YYYY-MM-DD"}]->(Laptop)
(Customer)-[:RECOMMENDS {rating: 1-5, comment: "..." }]->(Laptop)
```

2. Create Nodes and Relationships

Creating Laptop nodes

```
CREATE (l1:Laptop {laptop_id: 1, brand: "DELL", model: "Inspiron 15", type: "Notebook", specifications: "Intel i5, 8GB RAM", price: 50000, rating: 4.5}), (l2:Laptop {laptop_id: 2, brand: "HP", model: "Pavilion", type: "Notebook", specifications: "Intel i7, 16GB RAM", price: 70000, rating: 4.7}), (l3:Laptop {laptop_id: 3, brand: "Apple", model: "MacBook Pro", type: "Ultrabook", specifications: "M1 Chip, 8GB RAM", price: 120000, rating: 4.9}), (l4:Laptop {laptop_id: 4, brand: "ASUS", model: "ZenBook", type: "Ultrabook", specifications: "Intel i7, 16GB RAM", price: 90000, rating: 4.6}), (l5:Laptop {laptop_id: 5, brand: "LENOVO", model: "ThinkPad", type: "Business", specifications: "Intel i5, 8GB RAM", price: 60000, rating: 4.4});
```

// Creating Customer nodes

```
CREATE (c1:Customer {customer_id: 1, name: "Alice", email: "alice@example.com", purchase_date: "2023-01-26"}), (c2:Customer {customer_id: 2, name: "Bob", email: "bob@example.com", purchase_date: "2023-01-20"}), (c3:Customer {customer_id: 3, name: "Charlie", email: "charlie@example.com", purchase_date: "2023-01-26"}), (c4:Customer {customer_id: 4, name: "Diana", email: "diana@example.com", purchase_date: "2023-01-22"});
```

// Creating Relationships

```
CREATE (c1)-[:BOUGHT {purchase_date: "2023-01-26"}]->(l1), (c2)-[:BOUGHT {purchase_date: "2023-01-20"}]->(l2), (c3)-[:BOUGHT {purchase_date: "2023-01-26"}]->(l1), (c4)-[:BOUGHT {purchase_date: "2023-01-22"}]-
```

>(l3), (c1)-[:RECOMMENDS {rating: 5, comment: "Great performance!"}]->(l1), (c2)-[:RECOMMENDS {rating: 4, comment: "Good value for money."}]->(l2), (c3)-[:RECOMMENDS {rating: 5, comment: "Excellent laptop!"}]->(l1);

OR ALL In ONE

```
CREATE
(l1:Laptop {laptop_id: 1, brand: "DELL", model: "Inspiron 15", type: "Notebook", specifications: "Intel i5, 8GB RAM", price: 50000, rating: 4.5}),
(l2:Laptop {laptop_id: 2, brand: "HP", model: "Pavilion", type: "Notebook", specifications: "Intel i7, 16GB RAM", price: 70000, rating: 4.7}),
(l3:Laptop {laptop_id: 3, brand: "Apple", model: "MacBook Pro", type: "Ultrabook", specifications: "M1 Chip, 8GB RAM", price: 120000, rating: 4.9}),
(l4:Laptop {laptop  id: 4, brand: "ASUS", model: "ZenBook", type: "Ultrabook", specifications: "Intel i7, 16GB RAM", price: 90000, rating: 4.6}),
(l5:Laptop {laptop_id: 5, brand: "LENOVO", model: "ThinkPad", type: "Business", specifications: "Intel i5, 8GB RAM", price: 60000, rating: 4.4}),
(c1:Customer {customer  id: 1, name: "Alice", email: "alice@example.com", purchase  date: "2023-01-26"}),
(c2:Customer {customer_id: 2, name: "Bob", email: "bob@example.com", purchase_date: "2023-01-20"}),
(c3:Customer {customer_id: 3, name: "Charlie", email: "charlie@example.com", purchase_date: "2023-01-26"}),
(c4:Customer {customer_id: 4, name: "Diana", email: "diana@example.com", purchase_date: "2023-01-22"}),
(c1)-[:BOUGHT {purchase_date: "2023-01-26"}]->(l1),
(c2)-[:BOUGHT {purchase_date: "2023-01-20"}]->(l2),
(c3)-[:BOUGHT {purchase_date: "2023-01-26"}]->(l1),
(c4)-[:BOUGHT {purchase_date: "2023-01-22"}]->(l3),
(c1)-[:RECOMMENDS {rating: 5, comment: "Great performance!"}]->(l1),
(c2)-[:RECOMMENDS {rating: 4, comment: "Good value for money."}]->(l2),
(c3)-[:RECOMMENDS {rating: 5, comment: "Excellent laptop!"}]->(l1);
```

3. Answer the Queries

a. List the characteristics of a specific laptop

For example, to list the characteristics of the "Inspiron 15" laptop:

```
MATCH (l:Laptop {model: "Inspiron 15"}) RETURN l.brand, l.model, l.type, l.specifications, l.price, l.rating
```

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

```
MATCH (c:Customer)-[:BOUGHT]->(l:Laptop {brand: "DELL"}) RETURN c.name
```

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

```
MATCH (c:Customer) WHERE c.purchase_date = "2023-01-26" RETURN c.name
```

d. List the most recommended device

```
MATCH (c:Customer)-[r:RECOMMENDS]->(l:Laptop) RETURN l.model, COUNT(r) AS recommendations ORDER BY recommendations DESC LIMIT 1
```

Slip No. 20

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. [3]
2. Create nodes and relationships, along with their properties, and visualize your actual Graph model. [3]
3. Answer the following queries using Cypher:

a. List the types of plants from your graph model [3]

b. List the popular flowering plants. [3]
c. List the names of plants sold plant where qty>500 in last 2 days [4]
d. List the names of suppliers in decreasing order who supplies “Creepers”. [4]

To model a Nursery Management Information System as a graph database, we can identify the key entities and their relationships. Below is a detailed breakdown of the model, including queries to interact with the database using Cypher.

1. Identify Labels and Relationships

Labels

- Plant
  - Properties: `plant_id`, `name`, `type`, `category`, `price`, `quantity`, `rating`

- Fertilizer
  - Properties: `fertilizer_id`, `name`, `type`, `price`

- Customer
  - Properties: `customer_id`, `name`, `email`

- Supplier
  - Properties: `supplier_id`, `name`, `contact_info`

Relationships

- SOLD
  - From: Plant
  - To: Customer
  - Properties: `purchase_date`, `quantity`

- **SUPPLIES**
  - From: Supplier
  - To: Plant
  - Properties: `supply_date`

- RATES
  - From: Customer
  - To: App (or the Nursery)
  - Properties: `rating`, `comment`

High-Level Graph Model

2. Create Nodes and Relationships

Sample Data Creation

// Creating Plant nodes

```
CREATE
 (p1:Plant {plant_id: 1, name: "Rose", type: "Flower", category: "Flowering", price: 100, quantity: 600, rating: 4.5}),
 (p2:Plant {plant_id: 2, name: "Tulip", type: "Flower", category: "Flowering", price: 120, quantity: 300, rating: 4.7}),
(p3:Plant {plant_id: 3, name: "Bamboo", type: "Grass", category: "Non-Flowering", price: 200, quantity: 1000, rating: 4.8}),
(p4:Plant {plant_id: 4, name: "Creeper", type: "Creeper", category: "Non-Flowering", price: 80, quantity: 700, rating: 4.3}),
```

```
(p5:Plant {plant_id: 5, name: "Cactus", type: "Succulent", category: "Non-Flowering", price: 50, quantity: 800, rating: 4.0});

// Creating Fertilizer nodes
CREATE
(f1:Fertilizer {fertilizer_id: 1, name: "Organic Fertilizer", type: "Organic", price: 150}),
(f2:Fertilizer {fertilizer_id: 2, name: "Chemical Fertilizer", type: "Chemical", price: 100});

// Creating Customer nodes
CREATE
(c1:Customer {customer_id: 1, name: "Alice", email: "alice@example.com"}),
(c2:Customer {customer_id: 2, name: "Bob", email: "bob@example.com"}),
(c3:Customer {customer_id: 3, name: "Charlie", email: "charlie@example.com"});

// Creating Supplier nodes
CREATE
(s1:Supplier {supplier_id: 1, name: "Greenhouse Supplies", contact_info: "greenhouse@example.com"}),
(s2:Supplier {supplier_id: 2, name: "Flora Supplies", contact_info: "flora@example.com"});

// Creating Relationships
CREATE
(c1)-[:SOLD {purchase_date: "2023-10-27", quantity: 5}]->(p1),
(c2)-[:SOLD {purchase_date: "2023-10-28", quantity: 10}]->(p2),
(c3)-[:SOLD {purchase_date: "2023-10-26", quantity: 15}]->(p4),
(s1)-[:SUPPLIES {supply_date: "2023-10-20"}]->(p4),
(s2)-[:SUPPLIES {supply_date: "2023-10-21"}]->(p1),
(c1)-[:RATES {rating: 5, comment: "Great service!"}]->(s1),
(c2)-[:RATES {rating: 4, comment: "Loved the variety."}]->(s2);
```

3. Answer the Queries

a. List the types of plants from your graph model

```
MATCH (p:Plant) RETURN DISTINCT p.type
```

b. List the popular flowering plants

Assuming "popular" means those with a rating greater than 4.5:

```
MATCH (p:Plant {category: "Flowering"}) WHERE p.rating > 4.5 RETURN p.name, p.rating
```

c. List the names of plants sold where quantity > 500 in the last 2 days

```
MATCH (c:Customer)-[s:SOLD]->(p:Plant) WHERE s.purchase_date >= "2023-10-26" AND p.quantity > 500
RETURN p.name
```

d. List the names of suppliers in decreasing order who supply “Creepers”

```
MATCH (s:Supplier)-[:SUPPLIES]->(p:Plant {name: "Creeper"}) RETURN s.name ORDER BY s.name DESC
```