Slip.1.

Q1Consider the Worker table

Worker(WORKER_ID,FIRST_NAME,LAST_NAME,SALARY,JOINING_DATE,DEPARTMENT)Write a query to insert 5 rows in it as per the query requirements.1)List the department and the total salary for each department, sorted by total salary in descending order:2)Find the average salary for each department and display only those departments where the average salary is above $100,000, sorted by average salary in descending order

Ans: INSERT INTO Worker (WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT)

VALUES

(1, 'John', 'Doe', 120000, '2023-01-01', 'IT'),

(2, 'Jane', 'Smith', 110000, '2023-02-15', 'IT'),

(3, 'Michael', 'Johnson', 95000, '2023-03-20', 'Finance'),

(4, 'Emily', 'Williams', 105000, '2023-04-10', 'Finance'),

(5, 'David', 'Brown', 98000, '2023-05-05', 'HR');

1.SELECT DEPARTMENT, SUM(SALARY) AS TOTAL_SALARY

FROM Worker

GROUP BY DEPARTMENT

ORDER BY TOTAL_SALARY DESC;

2.SELECT DEPARTMENT, AVG(SALARY) AS AVERAGE_SALARY

FROM Worker

GROUP BY DEPARTMENT

HAVING AVG(SALARY) > 100000

ORDER BY AVERAGE_SALARY DESC;

Q2Model the following Property system as a document database. Consider a set of Property, Owner. One owner can buy many properties. 1. Assume appropriate attributes and collections as per the query requirements. 2. Insert at least 05 documents in each collection. 3. Answer the following Queries a)Display area wise property details. b)Display property owned by 'Mr.Patil' having minimum rate c)Give the details of owner whose property is at “Nashik”. d)Display area of property whose rate is less than 100000.

Property collection

[

- {"_id": 1, "area": "Downtown", "rate": 150000, "owner_id": 1},
- {"_id": 2, "area": "Suburb", "rate": 90000, "owner_id": 2},
- {"_id": 3, "area": "City Center", "rate": 120000, "owner_id": 1},
- {"_id": 4, "area": "Nashik", "rate": 80000, "owner_id": 3},
- {"_id": 5, "area": "Hillside", "rate": 110000, "owner_id": 2}
-]
- **Owner collection:-**
- [

{"_id": 1, "name": "Mr. Smith", "location": "New York"},

{"_id": 2, "name": "Mr. Patil", "location": "Mumbai"},

{"_id": 3, "name": "Ms. Gupta", "location": "Nashik"},

{"_id": 4, "name": "Mr. Johnson", "location": "Los Angeles"},

{"_id": 5, "name": "Ms. Lee", "location": "Seoul"}

]

A.

db.Properties.aggregate([

{$group: {_id: "$area", properties: {$push: "$$ROOT"}}}

])

B. db.Properties.find({"owner_id": 2}).sort({"rate": 1}).limit(1)

C. db.Owners.findOne({"location": "Nashik"})

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

Slip.2

Q1Consider the Employee table

Employee(emp_id,emp_name,job_name,manager_id,hire_date,salary)Write a query to insert 5 rows in it that as per the query requirements.1)List the average salary of each job title, sorted in descending order of average salary:2)List the department and the maximum salary among employees who joined after 1995, sorted by maximum salary in descending order.

Inserting 5rows:-

INSERT INTO Employee(emp_id, emp_name, job_name, manager_id, hire_date, salary)

VALUES

(1, 'John Doe', 'Manager', NULL, '1990-05-15', 70000),

(2, 'Jane Smith', 'Developer', 1, '1998-09-20', 60000),

(3, 'David Johnson', 'Developer', 1, '2001-03-10', 55000),

(4, 'Emily Brown', 'Analyst', 2, '2005-07-01', 50000),

(5, 'Michael Lee', 'Analyst', 2, '1999-11-12', 48000);

1. SELECT job_name, AVG(salary) AS average_salary

FROM Employee

GROUP BY job_name

ORDER BY average_salary DESC;

2. SELECT department, MAX(salary) AS max_salary

FROM Employee

WHERE hire_date > '1995-01-01'

GROUP BY department

ORDER BY max_salary DESC;

Q2Model the following system as a document database. Consider a database of newspaper, publisher, and city. Different publisher publishes various newspapers in different cities 2. Assume appropriate attributes and collections as per the query requirements. 3. Insert at least 5 documents in each collection. 4. Answer the following Queries. a. List all newspapers available "NASHIK" city b. List all the newspaper of "Marathi" language c. Count no. of publishers of "Gujrat" state d. Write a cursor to show newspapers with highest sale in Maharashtra state[4] with outputs

- Newspaper collection.
- [
- {"_id": 1, "name": "Times of India", "language": "English", "publisher_id": 1, "city_id": 1, "sales": 20000},
- {"_id": 2, "name": "Lokmat", "language": "Marathi", "publisher_id": 2, "city_id": 2, "sales": 18000},
- {"_id": 3, "name": "Gujarat Samachar", "language": "Gujarati", "publisher_id": 3, "city_id": 3, "sales": 22000},
- {"_id": 4, "name": "Maharashtra Times", "language": "Marathi", "publisher_id": 4, "city_id": 2, "sales": 25000},
- {"_id": 5, "name": "Navbharat Times", "language": "Hindi", "publisher_id": 5, "city_id": 4, "sales": 21000}
-]
- Publisher collection:-

- [
- {"_id": 1, "name": "Times Group", "state": "Maharashtra"},
- {"_id": 2, "name": "Lokmat Media", "state": "Maharashtra"},
- {"_id": 3, "name": "Gujarat Samachar Group", "state": "Gujarat"},
- {"_id": 4, "name": "Maharashtra Times Group", "state": "Maharashtra"},
- {"_id": 5, "name": "Navbharat Times Group", "state": "Delhi"}
-]
- Cities collection:-
- [
- {"_id": 1, "name": "Mumbai"},
- {"_id": 2, "name": "Pune"},
- {"_id": 3, "name": "Ahmedabad"},
- {"_id": 4, "name": "Nashik"}
-]

a.List all newspapers available in "Nashik" city.

db.Newspapers.find({"city_id": 4})

b. List all the newspapers of "Marathi" language:

db.Newspapers.find({"language": "Marathi"})

c. Count the number of publishers in the "Gujarat" state:

db.Publishers.count({"state": "Gujarat"})

d. Write a cursor to show newspapers with the highest sale in Maharashtra state:

db.Newspapers.find({"publisher_id": {$in: db.Publishers.find({"state": "Maharashtra"}, {_id: 1})}}).sort({"sales": -1}).limit(1)

SLIP.3

Q1.Consider the Worker table
Worker(WORKER_ID,FIRST_NAME,LAST_NAME,SALARY,JOINING_DA TE,DEPARTMENT)
Write a query to insert 5 rows in it as per the query requirements.
1)List the number of workers in each department who joined after January 1, 2021, sorted by department name:
2)Find the department with the highest number of workers earning a salary greater than $90,000, sorted by department name:

```
INSERT INTO Worker (WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT)
VALUES
  (1, 'John', 'Doe', 95000, '2021-03-15', 'Finance'),
  (2, 'Jane', 'Smith', 85000, '2022-02-20', 'IT'),
  (3, 'Alice', 'Johnson', 100000, '2021-07-10', 'Finance'),
  (4, 'Bob', 'Williams', 110000, '2023-01-08', 'IT'),
  (5, 'Emily', 'Brown', 105000, '2021-05-25', 'HR');
```

```
1. SELECT DEPARTMENT, COUNT(*) AS num_workers
FROM Worker
WHERE JOINING_DATE > '2021-01-01'
GROUP BY DEPARTMENT
ORDER BY DEPARTMENT;
2. SELECT DEPARTMENT, COUNT(*) AS num_high_salary_workers
FROM Worker
WHERE SALARY > 90000
GROUP BY DEPARTMENT
ORDER BY num_high_salary_workers DESC, DEPARTMENT;
```

Q2. 1. Model the following system as a document database. Consider employee and department's information. 2. Assume appropriate attributes and collections as per the query requirements. 3. Insert at least 5 documents in each collection. 4. Answer the following Queries. a. Display name of employee who has highest salary b. Display biggest department with max. no. of employees c. Write a cursor which shows department wise employee information d. List all the employees who work in Sales dept and salary > 50000.

-
- **Inserting documents.**
- [
- { "_id": 1, "name": "John Doe", "salary": 75000, "department_id": 1 },
- { "_id": 2, "name": "Jane Smith", "salary": 85000, "department_id": 2 },
- { "_id": 3, "name": "Alice Johnson", "salary": 95000, "department_id": 1 },
- { "_id": 4, "name": "Bob Williams", "salary": 110000, "department_id": 2 },
- { "_id": 5, "name": "Emily Brown", "salary": 105000, "department_id": 3 }
-]
-
- // Department collection
- [
- { "_id": 1, "name": "Finance", "employees": [1, 3] },
- { "_id": 2, "name": "IT", "employees": [2, 4] },
- { "_id": 3, "name": "Sales", "employees": [5] }
-]
- A.db.Employee.find({}, { "_id": 0, "name": 1, "salary": 1 }).sort({ "salary": -1 }).limit(1)
- B. db.Department.aggregate([
- {
- $project: {
- name: 1,
- numEmployees: { $size: "$employees" }
- }
- },
- {

- $sort: { numEmployees: -1 }
- },
- {
- $limit: 1
- }
-])
- C. var cursor = db.Department.find();
-
- while (cursor.hasNext()) {
- var department = cursor.next();
- var departmentName = department.name;
- var employeeIds = department.employees;
-
- print("Department: " + departmentName);
-
- var employeesCursor = db.Employee.find({ "_id": { $in: employeeIds } });
- while (employeesCursor.hasNext()) {
- var employee = employeesCursor.next();
- printjson(employee);
- }
- }
- D. db.Employee.find({
- "department_id": 3,
- "salary": { $gt: 50000 }
- })
-

Slip4

Q1Consider the Patient table Patient(PatientID,Name,DateOfBirth,Gender,admit_date,ward_no,City) Write a query to insert 5 rows in it that as per the query requirements. 1)List the number of patients admitted to each ward, sorted by ward number: 2)Find the average age of patients admitted to each ward, and display only those wards where the average age is below 40, sorted by average age in descending order.

```
INSERT INTO Patient (PatientID, Name, DateOfBirth, Gender, admit_date, ward_no, City)
VALUES
    (1, 'John Doe', '1985-05-15', 'Male', '2023-05-15', 101, 'New York'),
    (2, 'Jane Smith', '1978-09-20', 'Female', '2022-09-20', 102, 'Los Angeles'),
    (3, 'Alice Johnson', '1990-01-10', 'Female', '2023-01-10', 101, 'Chicago'),
    (4, 'Bob Williams', '1980-03-08', 'Male', '2022-03-08', 103, 'Houston'),
    (5, 'Emily Brown', '1992-07-25', 'Female', '2023-07-25', 101, 'Miami');
```

1. SELECT ward_no, COUNT(*) AS num_patients

```
FROM Patient
GROUP BY ward_no
ORDER BY ward_no;
```
2.
```
SELECT ward_no, AVG(YEAR(CURRENT_DATE) - YEAR(DateOfBirth)) AS avg_age
FROM Patient
GROUP BY ward_no
HAVING avg_age < 40
ORDER BY avg_age DESC;
```

Q2. 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] with all oututs

Inserting documents

```
// Hospital collection
[
  { "_id": 1, "name": "City Hospital", "city": "Nashik", "specializations": ["Pediatric", "Orthopedic"], "rating": 4.5 },
  { "_id": 2, "name": "Hope Hospital", "city": "Nashik", "specializations": ["Gynaec", "Orthopedic"], "rating": 4.2 },
  { "_id": 3, "name": "Sunrise Hospital", "city": "Mumbai", "specializations": ["Pediatric", "Gynaec"], "rating": 3.8 },
  { "_id": 4, "name": "Samaritan Hospital", "city": "Nashik", "specializations": ["Orthopedic"], "rating": 4.7 },
  { "_id": 5, "name": "Metro Hospital", "city": "Nashik", "specializations": ["Cardiology"], "rating": 4.0 }
]

// Specialization collection
[
  { "_id": 1, "name": "Pediatric" },
  { "_id": 2, "name": "Orthopedic" },
  { "_id": 3, "name": "Gynaec" },
  { "_id": 4, "name": "Cardiology" }
]
```

```
// Review collection
[
  { "_id": 1, "hospital_id": 1, "reviewer_name": "John Doe", "rating": 4.5 },
  { "_id": 2, "hospital_id": 2, "reviewer_name": "Jane Smith", "rating": 4.2 },
  { "_id": 3, "hospital_id": 3, "reviewer_name": "Alice Johnson", "rating": 3.8 },
  { "_id": 4, "hospital_id": 4, "reviewer_name": "Bob Williams", "rating": 4.7 },
  { "_id": 5, "hospital_id": 5, "reviewer_name": "Emily Brown", "rating": 4.0 }
]

// Doctor collection
[
  { "_id": 1, "name": "Dr. Deshmukh", "hospitals": [1, 2] },
  { "_id": 2, "name": "Dr. Patel", "hospitals": [3, 4] },
  { "_id": 3, "name": "Dr. Khan", "hospitals": [5] }
]
```

A. db.Hospital.find({ "specializations": "Orthopedic" }, { "_id": 0, "name": 1 })

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

C.
```
db.Doctor.findOne({ "name": "Dr. Deshmukh" }).hospitals.forEach(function(hospitalId) {
  var hospital = db.Hospital.findOne({ "_id": hospitalId });
  print(hospital.name);
});
```

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

Slip5

Q1. Consider the Patient table Patient(PatientID,Name,DateOfBirth,Gender,admit_date,ward_no,City) Write a query to insert 5 rows in it that as per the query requirements. 1)List the number of male and female patients admitted to each ward, sorted by ward number and gender: 2)Find the ward with the highest number of patients admitted, and display the top 3 wards with the highest number of patients, sorted by the number of patients in descending order

Inserting rows:-

```
INSERT INTO Patient (PatientID, Name, DateOfBirth, Gender, admit_date, ward_no, City)
VALUES
  (1, 'John Doe', '1990-05-15', 'Male', '2024-03-15', 101, 'Nashik'),
  (2, 'Jane Smith', '1985-09-20', 'Female', '2024-03-20', 102, 'Nashik'),
  (3, 'Alice Johnson', '1992-01-10', 'Female', '2024-03-10', 101, 'Nashik'),
  (4, 'Bob Williams', '1988-03-08', 'Male', '2024-03-08', 103, 'Nashik'),
  (5, 'Emily Brown', '1995-07-25', 'Female', '2024-03-25', 101, 'Nashik');
```

1. SELECT ward_no, Gender, COUNT(*) AS num_patients

```
FROM Patient
GROUP BY ward_no, Gender
ORDER BY ward_no, Gender;
```

2.
```
SELECT ward_no, COUNT(*) AS num_patients
FROM Patient
GROUP BY ward_no
ORDER BY num_patients DESC
LIMIT 3;
```

Q2. 1. **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]**

Inserting documents

// Employee collection
```
[
 { "_id": 1, "name": "John Doe", "projects": [1, 2] },
 { "_id": 2, "name": "Jane Smith", "projects": [2, 3] },
 { "_id": 3, "name": "Alice Johnson", "projects": [1, 3] },
 { "_id": 4, "name": "Bob Williams", "projects": [3, 4] },
 { "_id": 5, "name": "Emily Brown", "projects": [1, 4] }
]
```

// Project collection
```
[
 { "_id": 1, "name": "Project A", "duration": 4, "project_type": "Type 1" },
 { "_id": 2, "name": "Project B", "duration": 5, "project_type": "Type 2" },
 { "_id": 3, "name": "Project C", "duration": 3, "project_type": "Type 1" },
 { "_id": 4, "name": "Project D", "duration": 2, "project_type": "Type 2" },
 { "_id": 5, "name": "Project E", "duration": 6, "project_type": "Type 1" }
]
```

A. `db.Project.find({ "project_type": "Type 1" }, { "_id": 0, "name": 1 })`
B. `db.Project.find({ "duration": { $gt: 3 } })`
C. `db.Employee.find({ "projects": 1 }).count()`
D.
```
db.Employee.findOne({ "name": "Mr. Patil" }).projects.forEach(function(projectId) {
  var project = db.Project.findOne({ "_id": projectId });
print(project.name);
});
```

Slip 6

Q1 Consider the Worker table
Worker(WORKER_ID,FIRST_NAME,LAST_NAME,SALARY,JOINING_DATE,DEPARTMENT)
Write a query to insert 5 rows in it as per the query requirements.
1)List the department and the number of workers who joined in February 2021 and have a salary greater than $80,000, sorted by department name:
2)Find the department with the highest average salary among departments with at least 2 workers, sorted by average salary in descending order:

```
INSERT INTO Worker(WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE,
DEPARTMENT)
VALUES
  (1, 'John', 'Doe', 90000, '2021-02-05', 'Finance'),
  (2, 'Jane', 'Smith', 85000, '2021-02-10', 'Finance'),
  (3, 'David', 'Johnson', 95000, '2021-02-15', 'IT'),
  (4, 'Emily', 'Brown', 80000, '2021-02-20', 'IT'),
  (5, 'Michael', 'Lee', 82000, '2021-02-25', 'Marketing');

1. SELECT DEPARTMENT, COUNT(*) AS num_workers
FROM Worker
WHERE MONTH(JOINING_DATE) = 2 AND YEAR(JOINING_DATE) = 2021 AND SALARY >
80000
GROUP BY DEPARTMENT
ORDER BY DEPARTMENT;

2. SELECT DEPARTMENT, AVG(SALARY) AS avg_salary
FROM Worker
GROUP BY DEPARTMENT
HAVING COUNT(*) >= 2
ORDER BY avg_salary DESC;
```

Q2 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]

Customer collection.

```
[
{"_id": 1, "name": "Alice", "age": 30, "policies": [{"policy_id": 1}]},
{"_id": 2, "name": "Bob", "age": 35, "policies": [{"policy_id": 2}, {"policy_id": 3}]},
```

{"_id": 3, "name": "Charlie", "age": 40, "policies": [{"policy_id": 1}, {"policy_id": 4}]},

{"_id": 4, "name": "David", "age": 45, "policies": [{"policy_id": 2}]},

{"_id": 5, "name": "Eva", "age": 50, "policies": [{"policy_id": 3}]}

]

Policy collection

[

{"_id": 1, "name": "Komal Jeevan", "type": "Yearly", "premium_amount": 50000, "company_id": 1},

{"_id": 2, "name": "Health Plus", "type": "Monthly", "premium_amount": 2500, "company_id": 2},

{"_id": 3, "name": "Accident Shield", "type": "Half Yearly", "premium_amount": 10000, "company_id": 3},

{"_id": 4, "name": "Retirement Secure", "type": "Half Yearly", "premium_amount": 20000, "company_id": 1},

{"_id": 5, "name": "Term Life", "type": "Yearly", "premium_amount": 15000, "company_id": 2}

]

Companies collection

[

{"_id": 1, "name": "ABC Insurance", "location": "New York"},

{"_id": 2, "name": "XYZ Insurance", "location": "London"},

{"_id": 3, "name": "PQR Insurance", "location": "Tokyo"}

]

a. db.Customers.find({"policies.policy_id": 1})
b. db.Policies.aggregate([
{$group: {_id: null, avg_premium_amount: {$avg: "$premium_amount"}}}
])

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

D. db.Customers.aggregate([

{$match: {"policies.type": "Half Yearly"}},

{$count: "num_customers"}

])

Slip7

Q1Consider the Employee table
Employee(emp_id,emp_name,job_name,manager_id,hire_date,salary)
Write a query to insert 5 rows in it that as per the query requirements.
1)Find the total number of employees in each department whose salary is above $2000, sorted by department name:
2)List the manager_id and the number of employees managed by each manager who manages more than one employee, sorted by manager_id:

```
INSERT INTO Employee (emp_id, emp_name, job_name, manager_id, hire_date, salary)
VALUES
  (1, 'John Doe', 'Developer', 101, '2023-01-01', 2500),
  (2, 'Jane Smith', 'Manager', 102, '2022-05-15', 3500),
  (3, 'Alice Johnson', 'Developer', 101, '2023-03-20', 2200),
  (4, 'Bob Williams', 'Developer', 102, '2023-02-10', 2800),
  (5, 'Emily Brown', 'Manager', 101, '2022-10-05', 4000);
```

1.
```
SELECT department, COUNT(*) AS num_employees
FROM Employee
WHERE salary > 2000
GROUP BY department
ORDER BY department;
```

2.
```
SELECT manager_id, COUNT(*) AS num_employees_managed
FROM Employee
WHERE job_name = 'Manager'
GROUP BY manager_id
HAVING COUNT(*) > 1
ORDER BY manager_id;
```

Q2. 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]

```
// Customers collection
[
```

```
 { "_id": 1, "firstName": "John", "lastName": "Doe", "branch": "Branch A" },
 { "_id": 2, "firstName": "Jane", "lastName": "Smith", "branch": "Branch B" },
 { "_id": 3, "firstName": "Alice", "lastName": "Johnson", "branch": "Branch A" },
 { "_id": 4, "firstName": "Bob", "lastName": "Williams", "branch": "Branch C" },
 { "_id": 5, "firstName": "Emily", "lastName": "Brown", "branch": "Branch A" }
]

// Accounts collection
[
 { "_id": 1, "customerId": 1, "accountType": "Savings", "openDate": "2020-01-01", "branch":
"Branch A" },
 { "_id": 2, "customerId": 2, "accountType": "Checking", "openDate": "2020-01-01", "branch":
"Branch B" },
 { "_id": 3, "customerId": 3, "accountType": "Savings", "openDate": "2020-02-15", "branch":
"Branch A" },
 { "_id": 4, "customerId": 4, "accountType": "Loan", "openDate": "2020-01-01", "branch":
"Branch C" },
 { "_id": 5, "customerId": 5, "accountType": "Savings", "openDate": "2020-01-01", "branch":
"Branch A" }
]

// Transactions collection
[
 { "_id": 1, "accountId": 1, "amount": 1000, "transactionType": "Deposit", "timestamp": "2020-
01-02T08:00:00Z" },
 { "_id": 2, "accountId": 2, "amount": 500, "transactionType": "Withdrawal", "timestamp":
"2020-01-03T10:00:00Z" },
 { "_id": 3, "accountId": 3, "amount": 1500, "transactionType": "Deposit", "timestamp": "2020-
02-16T09:00:00Z" },
 { "_id": 4, "accountId": 4, "amount": 2000, "transactionType": "Withdrawal", "timestamp":
"2020-01-05T11:00:00Z" },
 { "_id": 5, "accountId": 5, "amount": 3000, "transactionType": "Deposit", "timestamp": "2020-
01-04T12:00:00Z" }
]
```

A. db.Customers.find({ "firstName": /^S/i })
B. db.Accounts.find({ "openDate": "2020-01-01", "branch": "Branch A" })
C. var savingsCustomerIds = db.Accounts.find({ "accountType": "Savings" }, { "_id": 1 }).map(function(account) {
   return account.customerId;
   });
   db.Customers.find({ "_id": { $in: savingsCustomerIds } })

D. var loanAccountHoldersCount = db.Accounts.count({ "accountType": "Loan", "branch": "Branch C" });

loanAccountHoldersCount

Slip8

Q1. Consider the Employee table

Employee(emp_id,emp_name,job_name,manager_id,hire_date,salary)

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

1)Find the department with the highest average salary among departments with

at least 3 employees:

2)Find the top 3 departments with the highest total salary expenditure, sorted by

total salary expenditure in descending order

```
INSERT INTO Employee (emp_id, emp_name, job_name, manager_id, hire_date, salary, department_id)
VALUES
  (1, 'John Doe', 'Developer', 101, '2023-01-01', 5000, 1),
  (2, 'Jane Smith', 'Manager', 102, '2022-05-15', 6000, 2),
  (3, 'Alice Johnson', 'Developer', 101, '2023-03-20', 4500, 1),
  (4, 'Bob Williams', 'Developer', 102, '2023-02-10', 4800, 2),
  (5, 'Emily Brown', 'Manager', 101, '2022-10-05', 7000, 3);
```

```
1. SELECT department_id, AVG(salary) AS avg_salary
FROM Employee
GROUP BY department_id
HAVING COUNT(*) >= 3
ORDER BY avg_salary DESC
LIMIT 1;
2. SELECT department_id, SUM(salary) AS total_salary_expenditure
FROM Employee
GROUP BY department_id
ORDER BY total_salary_expenditure DESC
```

LIMIT 3;

Q2Model 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]

Inserting documents

// Items collection

```
[
  { "_id": 1, "name": "Chair", "quantity": 400, "status": "A", "height": 10, "tags": [1, 2] },
  { "_id": 2, "name": "Table", "quantity": 250, "status": "B", "height": 12, "tags": [1, 3, 4] },
  { "_id": 3, "name": "Planner", "quantity": 150, "status": "B", "height": 9, "tags": [2, 4] },
  { "_id": 4, "name": "Desk", "quantity": 350, "status": "A", "height": 8, "tags": [1, 3] },
  { "_id": 5, "name": "Lamp", "quantity": 200, "status": "A", "height": 6, "tags": [2] }
]
```

// Warehouses collection

```
[
  { "_id": 1, "name": "Warehouse A", "items": [{ "itemId": 1, "quantity": 150 }, { "itemId": 2, "quantity": 200 }] },
  { "_id": 2, "name": "Warehouse B", "items": [{ "itemId": 3, "quantity": 100 }, { "itemId": 4, "quantity": 300 }] },
  { "_id": 3, "name": "Warehouse C", "items": [{ "itemId": 2, "quantity": 50 }, { "itemId": 5, "quantity": 100 }] }
]
```

// Tags collection

```
[
```

```
  { "_id": 1, "name": "Furniture" },
  { "_id": 2, "name": "Office" },
  { "_id": 3, "name": "Wooden" },
  { "_id": 4, "name": "Plastic" },
  { "_id": 5, "name": "Metal" }
]
```

A. db.Items.find({ "quantity": { $gt: 300 } })
B. db.Items.find({ "tags": { $exists: true, $not: { $size: 5 } } })
C. db.Items.find({ $or: [{ "status": "B" }, { $and: [{ "quantity": { $lt: 50 } }, { "height": { $gt: 8 } }] }] })
D. var plannerItem = db.Items.findOne({ "name": "Planner" });
   var warehouses = db.Warehouses.find({ "items.itemId": plannerItem._id, "items.quantity": { $lt: 20 } });
   warehouses

Slip9

Q1Consider the Worker table Worker(WORKER_ID,FIRST_NAME,LAST_NAME,SALARY,JOINING_DATE,DEPARTMENT) Write a query to insert 5 rows in it that as per the query requirements.1)List the departments where the total salary expenditure is less than $300,000, sorted by total salary expenditure in ascending order 2)Find the worker with the highest salary in each department, sorted by department name:

```
INSERT INTO Worker (WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT)
VALUES
  (1, 'John', 'Doe', 50000, '2023-01-01', 'Finance'),
  (2, 'Jane', 'Smith', 60000, '2022-05-15', 'HR'),
  (3, 'Alice', 'Johnson', 45000, '2023-03-20', 'Finance'),
  (4, 'Bob', 'Williams', 48000, '2023-02-10', 'HR'),
  (5, 'Emily', 'Brown', 70000, '2022-10-05', 'IT');
```

```
1. SELECT DEPARTMENT, SUM(SALARY) AS TOTAL_SALARY
FROM Worker
GROUP BY DEPARTMENT
HAVING SUM(SALARY) < 300000
ORDER BY TOTAL_SALARY ASC;
```

2. SELECT *

FROM (

  SELECT *,

    ROW_NUMBER() OVER (PARTITION BY DEPARTMENT ORDER BY SALARY DESC) AS rn

  FROM Worker

) AS temp

WHERE rn = 1

ORDER BY DEPARTMENT;

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. 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]

Inserting documents

// Customers collection

[

{ "_id": 1, "name": "John Doe", "address": "123 Main St", "city": "Pune" },

{ "_id": 2, "name": "Jane Smith", "address": "456 Elm St", "city": "Pimpri" },

{ "_id": 3, "name": "Alice Johnson", "address": "789 Oak St", "city": "Pimpri" },

{ "_id": 4, "name": "Bob Williams", "address": "321 Pine St", "city": "Pune" },

{ "_id": 5, "name": "Emily Brown", "address": "654 Maple St", "city": "Pune" }

]

// Loans collection

[

{ "_id": 1, "customerId": 1, "loanType": "Personal", "loanAmount": 10000 },

{ "_id": 2, "customerId": 2, "loanType": "Home", "loanAmount": 200000 },

{ "_id": 3, "customerId": 3, "loanType": "Car", "loanAmount": 50000 },

{ "_id": 4, "customerId": 4, "loanType": "Personal", "loanAmount": 30000 },

{ "_id": 5, "customerId": 5, "loanType": "Education", "loanAmount": 100000 }

]

A. db.Customers.find({ "name": /^D/i })

B. db.Customers.find({ "city": "Pimpri" }).sort({ "name": -1 })

C.
```
var maxLoanAmount = db.Loans.aggregate([
  { $group: { _id: null, maxLoanAmount: { $max: "$loanAmount" } } }
]).next().maxLoanAmount;


db.Loans.aggregate([
  { $match: { "loanAmount": maxLoanAmount } },
  {
    $lookup: {
    from: "Customers",
  localField: "customerId",
    foreignField: "_id",
  as: "customer"
  }
  },
  { $unwind: "$customer" },
  { $project: { "customer.name": 1, "customer.address": 1, "customer.city": 1,
  "loanType": 1, "loanAmount": 1 } }
])
```

D.
```
db.Customers.updateMany(
  { "name": "Mr. Patil" },
  { $set: { "address": "<new_address>" } }
)
```

Slip10

Q1 Consider the Employee table
Employee(emp_id,emp_name,job_name,manager_id,hire_date,salary)

1)List the manager_id and the average salary of employees managed by each manager who manages at least 2 employees, sorted by average salary in descending order: 2)Find the departments where the average salary of employees is greater than the average salary of all employees, sorted by department name

1.SELECT manager_id, AVG(salary) AS average_salary

FROM Employee

WHERE manager_id IS NOT NULL

GROUP BY manager_id

HAVING COUNT(emp_id) >= 2

ORDER BY average_salary DESC;

2. SELECT department

FROM Employee

GROUP BY department

HAVING AVG(salary) > (SELECT AVG(salary) FROM Employee)

ORDER BY department;

Q2 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 names of products with brand which have highest rating. [4] D. Display customers who stay in …… city and billamt >50000 .[4]**

3. // Products collection
   [
   { "_id": 1, "name": "Laptop", "brand": "Dell", "warranty_period": 1, "rating": 4.5 },

{ "_id": 2, "name": "Smartphone", "brand": "Apple", "warranty_period": 2, "rating": 4.8 },

{ "_id": 3, "name": "Television", "brand": "Samsung", "warranty_period": 1, "rating": 4.3 },

{ "_id": 4, "name": "Refrigerator", "brand": "LG", "warranty_period": 2, "rating": 4.2 },
{ "_id": 5, "name": "Headphones", "brand": "Sony", "warranty_period": 1, "rating": 4.6 }
]

// Customers collection
[
{ "_id": 1, "name": "John Doe", "city": "New York", "purchase_date": "2023-08-15", "bill_amount": 60000 },
{ "_id": 2, "name": "Jane Smith", "city": "Los Angeles", "purchase_date": "2023-08-15", "bill_amount": 55000 },

{ "_id": 3, "name": "Alice Johnson", "city": "Chicago", "purchase_date": "2023-08-14", "bill_amount": 70000 },

{ "_id": 4, "name": "Bob Williams", "city": "Houston", "purchase_date": "2023-08-16", "bill_amount": 45000 },

{ "_id": 5, "name": "Emily Brown", "city": "Philadelphia", "purchase_date": "2023-08-15", "bill_amount": 80000 }

]

A. db.Products.find({ "warranty_period": 1 }, { "name": 1 })

B. db.Customers.find({ "purchase_date": "2023-08-15" })

C.var highestRatedBrand = db.Products.aggregate([
{ $group: { _id: "$brand", maxRating: { $max: "$rating" } } },
{ $sort: { maxRating: -1 } },
{ $limit: 1 }

]).next()._id;

db.Products.find({ "brand": highestRatedBrand }, { "name": 1 })

D. db.Customers.find({ "city": "<specific_city>", "bill_amount": { $gt: 50000 } })

Slip 11

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

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

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

Ans: SELECT publication_year, COUNT(*) AS num_books_published
FROM book
GROUP BY publication_year
ORDER BY publication_year ASC;

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

Ans: SELECT author_name, COUNT(*) AS total_books
FROM books
GROUP BY author_name

HAVING COUNT(*) > 1

ORDER BY total_books DESC;

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]

List all the orders along with their invoice for “Mr. Rajiv”.[4]

Ans:

Products Collection:

[

{"_id": 1, "name": "Laptop", "price": 1000, "description": "High-performance laptop", "quantity_available": 10},

More five just minor changes

]

Customers Collection:

[

{"_id": 1, "name": "Mr. Rajiv", "email": "rajiv@example.com", "address": "123 Main St", "phone": "123-456-7890"},

{"_id": 2, "name": "Ms. Sarah", "email": "sarah@example.com", "address": "456 Elm St", "phone": "987-654-3210"},

{"_id": 3, "name": "Mr. John", "email": "john@example.com", "address": "789 Oak St", "phone": "111-222-3333"},

{"_id": 4, "name": "Ms. Emily", "email": "emily@example.com", "address": "321 Pine St", "phone": "444-555-6666"},
{"_id": 5, "name": "Mr. Michael", "email": "michael@example.com", "address": "654 Cedar St", "phone": "777-888-9999"}
]

Orders Collection:

[
{"_id": 1, "customer_id": 1, "products": [{"product_id": 1, "quantity": 2}, {"product_id": 2, "quantity": 1}], "total_value": 2500, "processed": true},
{"_id": 2, "customer_id": 2, "products": [{"product_id": 3, "quantity": 3}, {"product_id": 4, "quantity": 1}], "total_value": 800, "processed": false},
{"_id": 3, "customer_id": 3, "products": [{"product_id": 5, "quantity": 2}], "total_value": 400, "processed": false},
{"_id": 4, "customer_id": 4, "products": [{"product_id": 1, "quantity": 1}], "total_value": 1000, "processed": true},
{"_id": 5, "customer_id": 5, "products": [{"product_id": 2, "quantity": 2}, {"product_id": 3, "quantity": 1}], "total_value": 1100, "processed": false}
]

Invoices Collection:

[
{"_id": 1, "order_id": 1, "total_value": 2500},
{"_id": 2, "order_id": 4, "total_value": 1000}
]

A. List all products in the inventory:

```
db.products.find({})
```

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

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

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

```
db.orders.find({ "processed": false })
```

D. List all the orders along with their invoice for "Mr. Rajiv":

```
db.orders.aggregate([
 {
 $match: { "customer_id": 1 } // Assuming Mr. Rajiv's customer ID is 1
 },
 {
 $lookup: {
 from: "invoices",
 localField: "_id",
 foreignField: "order_id",
 as: "invoice"
 }
 },
 {
 $unwind: "$invoice"
 },
 {
 $project: {
 "_id": 1,
 "total_value": 1,
 "invoice._id": 1,
 "invoice.total_value": 1
 }
 }
])
```

Slip 12

Consider the Worker table

Worker(WORKER_ID,FIRST_NAME,LAST_NAME,SALARY,JOINING_DA

TE,DEPARTMENT)

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

Ans : INSERT INTO Worker (WORKER_ID, FIRST_NAME, LAST_NAME,

SALARY, JOINING_DATE, DEPARTMENT)

VALUES

(1, 'John', 'Doe', 50000, '2021-02-15', 'IT'),

(2, 'Jane', 'Smith', 60000, '2020-12-20', 'HR'),

(3, 'Michael', 'Johnson', 55000, '2021-03-10', 'Finance'),

(4, 'Emily', 'Williams', 52000, '2021-01-05', 'Marketing'),

(5, 'David', 'Brown', 58000, '2021-04-25', 'IT');

1)Find the departments where the number of workers is equal to the number of

distinct first names, sorted by department name

Ans; SELECT DEPARTMENT

FROM Worker

GROUP BY DEPARTMENT

HAVING COUNT(WORKER_ID) = COUNT(DISTINCT FIRST_NAME)

ORDER BY DEPARTMENT;

2)List the number of workers in each department who joined after January 1,

2021, sorted by department name

Ans;

SELECT DEPARTMENT, COUNT(*) AS num_workers

FROM Worker

WHERE JOINING_DATE > '2021-01-01'

GROUP BY DEPARTMENT

ORDER BY DEPARTMENT;

Model the following University information system as a graph model, and

answer the following queries using Cypher.

University has various departments like Physics, Geography, Computer etc.

Each department conducts various courses and a course may be conducted

by multiple departments. Every course may have recommendations provided

by people.

1. Identify the labels and relationships, along with their properties, anddraw 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 :

A. List the details of all the departments in the university. [3]

B. List the names of the courses provided by Physics department. [3]

C. List the most recommended course in Geography department. [4]

List the names of common courses across Mathematics and computer department. [4]

ANS;

(Department)-[:DEPARTMENT_CONDUCTS_COURSE]->(Course)

(Course)-[:COURSE_RECOMMENDED_BY]->(Recommendation)

```
CREATE (:Department {name: 'Physics'}),
 (:Department {name: 'Geography'}),
 (:Department {name: 'Computer'});
MATCH (d:Department {name: 'Physics'})
CREATE (d)-[:DEPARTMENT_CONDUCTS_COURSE]->(:Course {name:
'Quantum Mechanics'}),
 (d)-[:DEPARTMENT_CONDUCTS_COURSE]->(:Course {name: 'Classical
Mechanics'});
MATCH (d:Department {name: 'Geography'})
CREATE (d)-[:DEPARTMENT_CONDUCTS_COURSE]->(:Course {name:
'Physical Geography'}),
 (d)-[:DEPARTMENT_CONDUCTS_COURSE]->(:Course {name: 'Human
Geography'});
MATCH (d:Department {name: 'Computer'})
CREATE (d)-[:DEPARTMENT_CONDUCTS_COURSE]->(:Course {name:
```

```
'Computer Programming'}),
 (d)-[:DEPARTMENT_CONDUCTS_COURSE]->(:Course {name: 'Data
Structures'});
MATCH (c:Course {name: 'Quantum Mechanics'})
CREATE (c)-[:COURSE_RECOMMENDED_BY]->(:Recommendation
{recommendation_text: 'Highly recommended by Prof. X'});
MATCH (c:Course {name: 'Human Geography'})
CREATE (c)-[:COURSE_RECOMMENDED_BY]->(:Recommendation
{recommendation_text: 'Great course, must take!'});
MATCH p=()-[r]->()
RETURN p;
```

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

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

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

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

C. List the most recommended course in Geography department:

```
MATCH (:Department {name: 'Geography'})-
[:DEPARTMENT_CONDUCTS_COURSE]->(c:Course)-
[:COURSE_RECOMMENDED_BY]->(r:Recommendation)
RETURN c.name, COUNT(r) AS num_recommendations
ORDER BY num_recommendations DESC LIMIT 1;
```

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

```
MATCH (math:Department {name: 'Mathematics'})-
[:DEPARTMENT_CONDUCTS_COURSE]->(math_course:Course),
 (comp:Department {name: 'Computer'})-
```

[:DEPARTMENT_CONDUCTS_COURSE]->(comp_course:Course)

WHERE math_course.name = comp_course.name

RETURN math_course.name;s:

Slip 13

Consider the Book table book

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

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

ANS:

```
INSERT INTO Book (book_id, title, author, publication_year, language,
available_copies, total_copies)
VALUES
 (1, 'The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'English', 3, 5),
 (2, 'Don Quixote', 'Miguel de Cervantes', 1605, 'Spanish', 8, 10),
 (3, 'Les Misérables', 'Victor Hugo', 1862, 'French', 4, 6),
 (4, 'War and Peace', 'Leo Tolstoy', 1869, 'Russian', 2, 3),
 (5, 'One Hundred Years of Solitude', 'Gabriel García Márquez', 1967, 'Spanish',
6, 8);
```

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

ANS:

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

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

ANS:

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

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

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

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

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

3. Answer the following Queries :

A. List the books of type “text” [3]

B. List the name of student who bought a text and reference types books.[3]

C. List the most recommended book type. [4]

List the student who buy the more than one type of book [4]

Ans:

(Student)-[:STUDENT_BUYS_BOOK]->(Book)

(Book)-[:BOOK_RECOMMENDED_BY]->(Recommendation)

```
CREATE (:Book {title: 'Book A', type: 'text'}),
 (:Book {title: 'Book B', type: 'reference'}),
 (:Book {title: 'Book C', type: 'text'}),
 (:Book {title: 'Book D', type: 'bibliography'}),
 (:Book {title: 'Book E', type: 'text'});
```

```
CREATE (:Student {name: 'John'}),
 (:Student {name: 'Jane'}),
 (:Student {name: 'Alice'}),
 (:Student {name: 'Bob'}),
 (:Student {name: 'Eva'});
MATCH (b:Book {type: 'text'})
CREATE (b)-[:BOOK_RECOMMENDED_BY]->(:Recommendation
{recommendation_text: 'Highly recommended!'});
MATCH (b:Book {type: 'reference'})
CREATE (b)-[:BOOK_RECOMMENDED_BY]->(:Recommendation
{recommendation_text: 'Great reference material!'});
MATCH (s:Student {name: 'John'}), (b:Book {title: 'Book A'})
CREATE (s)-[:STUDENT_BUYS_BOOK]->(b);
MATCH (s:Student {name: 'Jane'}), (b:Book {title: 'Book A'})
CREATE (s)-[:STUDENT_BUYS_BOOK]->(b);
MATCH (s:Student {name: 'Jane'}), (b:Book {title: 'Book B'})
CREATE (s)-[:STUDENT_BUYS_BOOK]->(b);
MATCH (s:Student {name: 'Alice'}), (b:Book {title: 'Book C'})
CREATE (s)-[:STUDENT_BUYS_BOOK]->(b);
MATCH (s:Student {name: 'Bob'}), (b:Book {title: 'Book E'})
CREATE (s)-[:STUDENT_BUYS_BOOK]->(b);
MATCH (s:Student {name: 'Eva'}), (b:Book {title: 'Book D'})
CREATE (s)-[:STUDENT_BUYS_BOOK]->(b);
MATCH p=()-[r]->()
RETURN p;
```

A. List the books of type “text”:

```
MATCH (b:Book {type: 'text'})
RETURN b;
```

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

```
MATCH (s:Student)-[:STUDENT_BUYS_BOOK]->(:Book {type: 'text'}),
 (s)-[:STUDENT_BUYS_BOOK]->(:Book {type: 'reference'})
RETURN s.name;
```

C. List the most recommended book type:

```
MATCH (:Book)-[:BOOK_RECOMMENDED_BY]->(r:Recommendation)
RETURN COUNT(r) AS num_recommendations, 'text' AS book_type
ORDER BY num_recommendations DESC
LIMIT 1;
```

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

```
MATCH (s:Student)-[:STUDENT_BUYS_BOOK]->(b:Book)
WITH s, COUNT(DISTINCT b.type) AS num_types
WHERE num_types > 1
RETURN s.name;
```

Slip14

Consider the Book table book (book_id,title,author,publication_year,language,available_copies,total_copies)

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

Ans:

```
INSERT INTO book (book_id, title, author, publication_year, language,
available_copies, total_copies)
VALUES
 (1, 'Title 1', 'Author A', 2020, 'English', 10, 15),
 (2, 'Title 2', 'Author B', 2019, 'Spanish', 8, 12),
 (3, 'Title 3', 'Author A', 2018, 'English', 12, 20),
 (4, 'Title 4', 'Author C', 2021, 'French', 6, 10),
 (5, 'Title 5', 'Author D', 2020, 'English', 15, 18);
```

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

Ans:

```
SELECT language, COUNT(*) AS total_books
FROM book
GROUP BY language
ORDER BY total_books DESC;
```

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

Ans:

```
SELECT author, AVG(total_copies) AS avg_copies_available
FROM book
GROUP BY author
ORDER BY avg_copies_available DESC
LIMIT 3;
```

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

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

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

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

3. Answer the following Queries:

A. List the characteristics of four wheeler types. [3]

B. List the name of customers who bought a two wheeler vehicle. [3]

C. List the customers who bought more than one type of vehicle. [4]

List the most recommended vehicle type. [4]

Ans:

```
(Customer)-[:CUSTOMER_BUYS_VEHICLE]->(Vehicle)
(Vehicle)-[:VEHICLE_RECOMMENDED_BY]->(Recommendation)
CREATE (:Vehicle {type: 'Two-Wheeler', characteristics: 'Sleek design, fuel  efficient'}),
 (:Vehicle {type: 'Four-Wheeler', characteristics: 'Spacious, powerful
engine'}),
 (:Vehicle {type: 'SUV', characteristics: 'Off-road capability, large seating
capacity'}),
 (:Vehicle {type: 'Electric', characteristics: 'Environmentally friendly,
quiet operation'}),
 (:Vehicle {type: 'Luxury', characteristics: 'Premium features,
comfort'});
CREATE (:Customer {name: 'John'}),
 (:Customer {name: 'Jane'}),
 (:Customer {name: 'Alice'}),
 (:Customer {name: 'Bob'}),
 (:Customer {name: 'Eva'});
MATCH (v:Vehicle {type: 'Two-Wheeler'})
CREATE (v)-[:VEHICLE_RECOMMENDED_BY]->(:Recommendation
{recommendation_text: 'Great for city commuting!'});
MATCH (v:Vehicle {type: 'Four-Wheeler'})
CREATE (v)-[:VEHICLE_RECOMMENDED_BY]->(:Recommendation
{recommendation_text: 'Excellent for family trips!'});
MATCH (c:Customer {name: 'John'}), (v:Vehicle {type: 'Two-Wheeler'})
CREATE (c)-[:CUSTOMER_BUYS_VEHICLE]->(v);
MATCH (c:Customer {name: 'Jane'}), (v:Vehicle {type: 'Two-Wheeler'})
CREATE (c)-[:CUSTOMER_BUYS_VEHICLE]->(v);
MATCH (c:Customer {name: 'Jane'}), (v:Vehicle {type: 'Four-Wheeler'})
CREATE (c)-[:CUSTOMER_BUYS_VEHICLE]->(v);
MATCH (c:Customer {name: 'Alice'}), (v:Vehicle {type: 'SUV'})
```

```
CREATE (c)-[:CUSTOMER_BUYS_VEHICLE]->(v);
MATCH (c:Customer {name: 'Bob'}), (v:Vehicle {type: 'Electric'})
CREATE (c)-[:CUSTOMER_BUYS_VEHICLE]->(v);
MATCH (c:Customer {name: 'Eva'}), (v:Vehicle {type: 'Luxury'})
CREATE (c)-[:CUSTOMER_BUYS_VEHICLE]->(v);
MATCH p=()-[r]->()
RETURN p;
```

Queries:

A. List the characteristics of four-wheeler types:

```
MATCH (v:Vehicle {type: 'Four-Wheeler'})
RETURN v.characteristics;
```

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

```
MATCH (c:Customer)-[:CUSTOMER_BUYS_VEHICLE]->(:Vehicle {type:
'Two-Wheeler'})
RETURN c.name;
```

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

```
MATCH (c:Customer)-[:CUSTOMER_BUYS_VEHICLE]->(v:Vehicle)
WITH c, COUNT(DISTINCT v.type) AS num_types
WHERE num_types > 1
RETURN c.name;
```

D. List the most recommended vehicle type:

```
MATCH (:Vehicle)-[:VEHICLE_RECOMMENDED_BY]->(r:Recommendation)
RETURN COUNT(r) AS num_recommendations, 'Two-Wheeler' AS
vehicle_type
ORDER BY num_recommendations DESC
LIMIT 1;
```

Slip15

Consider the Book table book

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

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

ANS:

```
INSERT INTO book (book_id, title, author, publication_year, language,
available_copies, total_copies)
VALUES
(1, 'Book 1', 'Author A', 2010, 'English', 5, 10),
(2, 'Book 2', 'Author B', 2015, 'Spanish', 8, 12),
(3, 'Book 3', 'Author A', 2018, 'English', 12, 20),
(4, 'Book 4', 'Author C', 2020, 'French', 6, 10),
(5, 'Book 5', 'Author D', 2019, 'English', 15, 18);
```

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

ANS:

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

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

ANS:

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

Model the following Car Showroom information as a graph model,and

answer the queries using Cypher. Consider a car showroom with different models of cars like sofas Honda city, Skoda, Creta, Swift, Ertiga etc. Showroom is divided into different sections, onesection for each car model; each section is handled by a sales staff. Asales staff can handle one or more sections. Customer may enquire about car. An enquiry may result in a purchase by the customer.

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

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

3. Answer the following queries:

A. List the types of cars available in the showroom. [3]

B. List the sections handled by Mr. Narayan. [3]

C. List the names of customers who have done only enquiry but not made any purchase. [4]

List the highly sale car model. [4]

ANS:

(Customer)-[:ENQUIRY_ABOUT_CAR]->(Enquiry)

(Customer)-[:PURCHASED_BY_CUSTOMER]->(Purchase)

(SalesStaff)-[:SALES_STAFF_HANDLES_SECTION]->(CarModel)

```
CREATE (:CarModel {name: 'Honda City'}),
 (:CarModel {name: 'Skoda'}),
 (:CarModel {name: 'Creta'}),
 (:CarModel {name: 'Swift'}),
 (:CarModel {name: 'Ertiga'});
CREATE (:SalesStaff {name: 'Mr. Narayan'}),
 (:SalesStaff {name: 'Ms. Sharma'}),
 (:SalesStaff {name: 'Mr. Singh'}),
 (:SalesStaff {name: 'Ms. Patel'}),
```

```
 (:SalesStaff {name: 'Mr. Kumar'});
MATCH (s:SalesStaff {name: 'Mr. Narayan'}), (c:CarModel {name: 'Honda City'})
CREATE (s)-[:SALES_STAFF_HANDLES_SECTION]->(c);
MATCH (s:SalesStaff {name: 'Mr. Narayan'}), (c:CarModel {name: 'Swift'})
CREATE (s)-[:SALES_STAFF_HANDLES_SECTION]->(c);
MATCH (s:SalesStaff {name: 'Ms. Sharma'}), (c:CarModel {name: 'Skoda'})
CREATE (s)-[:SALES_STAFF_HANDLES_SECTION]->(c);
MATCH (s:SalesStaff {name: 'Ms. Patel'}), (c:CarModel {name: 'Creta'})
CREATE (s)-[:SALES_STAFF_HANDLES_SECTION]->(c);
MATCH (c:Customer {name: 'John'}), (car:CarModel {name: 'Honda City'})
CREATE (c)-[:ENQUIRY_ABOUT_CAR]->(:Enquiry),
 (c)-[:PURCHASED_BY_CUSTOMER]->(:Purchase);
MATCH (c:Customer {name: 'Jane'}), (car:CarModel {name: 'Skoda'})
CREATE (c)-[:ENQUIRY_ABOUT_CAR]->(:Enquiry);
MATCH (c:Customer {name: 'Alice'}), (car:CarModel {name: 'Creta'})
CREATE (c)-[:ENQUIRY_ABOUT_CAR]->(:Enquiry),
 (c)-[:PURCHASED_BY_CUSTOMER]->(:Purchase);
MATCH (c:Customer {name: 'Bob'}), (car:CarModel {name: 'Swift'})
CREATE (c)-[:ENQUIRY_ABOUT_CAR]->(:Enquiry);
MATCH (c:Customer {name: 'Eva'}), (car:CarModel {name: 'Ertiga'})
CREATE (c)-[:ENQUIRY_ABOUT_CAR]->(:Enquiry);
MATCH p=()-[r]->()
RETURN p;
```

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

```
MATCH (c:CarModel)
RETURN c.name;
```

B. List the sections handled by Mr. Narayan:

```
MATCH (s:SalesStaff {name: 'Mr. Narayan'})-
[:SALES_STAFF_HANDLES_SECTION]->(c:CarModel)
```

RETURN c.name;

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

MATCH (c:Customer)-[:ENQUIRY_ABOUT_CAR]->(e:Enquiry)

WHERE NOT (c)-[:PURCHASED_BY_CUSTOMER]->()

RETURN c.name;

D. List the highly sale car model:

MATCH (c:CarModel)<-[:PURCHASED_BY_CUSTOMER]-(:Customer)

RETURN c.name, COUNT(*) AS sales

ORDER BY sales DESC

LIMIT 1;

Slip 16

Consider the Course table

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

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

ANS:

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

VALUES

(1, 'Introduction to Python', 'John Smith', 'Programming', 49.99, '4 weeks', 100),

(2, 'Machine Learning Fundamentals', 'Alice Johnson', 'Data Science', 69.99, '6 weeks', 80),

(3, 'Web Development with JavaScript', 'Michael Brown', 'Web Development', 59.99, '5 weeks', 120),

(4, 'Financial Accounting Basics', 'Emily Davis', 'Finance', 79.99, '8 weeks', 90),

(5, 'Graphic Design Principles', 'Daniel Wilson', 'Design', 54.99, '4 weeks', 110);

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

ANS:

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

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

ANS:

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

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

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

1. Identify the labels and relationships, along with their properties, and drawa 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 names of different medicines considered in your graph[3]

B. List the medicine that are highly Used in Rajasthan. [3]

C. List the highly used tablet in Gujarat. [4]

D. List the medicine names manufacturing “Powder” [4]

ANS:

```
(MedicineBrand)-[:BRAND_MANUFACTURES_MEDICINE]->(Medicine)
(Medicine)-[:MEDICINE_HAS_TYPE]->(MedicineType)
(Medicine)-[:MEDICINE_USED_IN_STATE]->(State)
CREATE (:MedicineBrand {name: 'Dr. Reddy'}),
 (:MedicineBrand {name: 'Cipla'}),
 (:MedicineBrand {name: 'SunPharma'});
CREATE (:Medicine {name: 'Medicine A'}),
 (:Medicine {name: 'Medicine B'}),
 (:Medicine {name: 'Medicine C'}),
 (:Medicine {name: 'Medicine D'}),
 (:Medicine {name: 'Medicine E'});
CREATE (:MedicineType {name: 'Tablet'}),
 (:MedicineType {name: 'Syrup'}),
 (:MedicineType {name: 'Powder'});
CREATE (:State {name: 'Rajasthan'}),
 (:State {name: 'Gujarat'}),
 (:State {name: 'Maharashtra'});
MATCH (b:MedicineBrand {name: 'Dr. Reddy'}),
 (m:Medicine {name: 'Medicine A'})
CREATE (b)-[:BRAND_MANUFACTURES_MEDICINE]->(m);
MATCH (b:MedicineBrand {name: 'Cipla'}),
 (m:Medicine {name: 'Medicine B'})
CREATE (b)-[:BRAND_MANUFACTURES_MEDICINE]->(m);
MATCH (b:MedicineBrand {name: 'SunPharma'}),
 (m:Medicine {name: 'Medicine C'})
CREATE (b)-[:BRAND_MANUFACTURES_MEDICINE]->(m);
MATCH (m:Medicine {name: 'Medicine A'}),
 (t:MedicineType {name: 'Tablet'})
CREATE (m)-[:MEDICINE_HAS_TYPE]->(t);
```

```
MATCH (m:Medicine {name: 'Medicine B'}),
 (t:MedicineType {name: 'Syrup'})
CREATE (m)-[:MEDICINE_HAS_TYPE]->(t);
MATCH (m:Medicine {name: 'Medicine C'}),
 (t:MedicineType {name: 'Powder'})
CREATE (m)-[:MEDICINE_HAS_TYPE]->(t);
MATCH (m:Medicine {name: 'Medicine A'}),
 (s:State {name: 'Rajasthan'})
CREATE (m)-[:MEDICINE_USED_IN_STATE {PercentageUse: 95}]->(s);
MATCH (m:Medicine {name: 'Medicine B'}),
 (s:State {name: 'Gujarat'})
CREATE (m)-[:MEDICINE_USED_IN_STATE {PercentageUse: 85}]->(s);
MATCH (m:Medicine {name: 'Medicine C'}),
 (s:State {name: 'Gujarat'})
CREATE (m)-[:MEDICINE_USED_IN_STATE {PercentageUse: 60}]->(s);
MATCH (m:Medicine {name: 'Medicine D'}),
 (s:State {name: 'Maharashtra'})
CREATE (m)-[:MEDICINE_USED_IN_STATE {PercentageUse: 40}]->(s);
MATCH p=()-[r]->()
RETURN p;
```

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

```
MATCH (m:Medicine)
RETURN m.name;
```

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

```
MATCH (m:Medicine)-[r:MEDICINE_USED_IN_STATE]->(s:State {name:
'Rajasthan'})
WHERE r.PercentageUse >= 90
RETURN m.name;
```

C. List the highly used tablet in Gujarat:

```
MATCH (m:Medicine)-[r:MEDICINE_USED_IN_STATE]->(s:State {name:
'Gujarat'}),
 (m)-[:MEDICINE_HAS_TYPE]->(t:MedicineType {name: 'Tablet'})
WHERE r.PercentageUse >= 90
RETURN m.name;
```

D. List the medicine names manufacturing “Powder”:

```
MATCH (m:Medicine)-[:MEDICINE_HAS_TYPE]->(t:MedicineType {name:
'Powder'})
RETURN m.name;
```

Slip 17

Consider the Course table

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

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

ANS:

```
INSERT INTO Courses (CourseID, Title, Instructor, Category, Price,
Duration, EnrollmentCount)
VALUES
 (1, 'Java Programming', 'John Doe', 'Programming', 49.99, '10 weeks', 100),
 (2, 'Data Science Fundamentals', 'Alice Smith', 'Data Science', 69.99, '6 weeks',
80),
 (3, 'Web Development with JavaScript', 'Michael Brown', 'Web Development',
59.99, '8 weeks', 120),
 (4, 'Financial Accounting Basics', 'Emily Davis', 'Finance', 79.99, '12 weeks',
90),
 (5, 'Graphic Design Principles', 'Daniel Wilson', 'Design', 54.99, '4 weeks',
110);
```

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

ANS:

```
SELECT Category, COUNT(*) AS NumCourses
FROM Courses
GROUP BY Category
HAVING COUNT(*) > 5
ORDER BY NumCourses DESC;
```

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

ANS:

```
SELECT Category, AVG(CAST(SUBSTRING(Duration, 1, POSITION(' ' IN
Duration) - 1) AS DECIMAL)) AS AvgDurationWeeks
FROM Courses
GROUP BY Category
HAVING AVG(CAST(SUBSTRING(Duration, 1, POSITION(' ' IN Duration) - 1)
AS DECIMAL)) < 8
ORDER BY Category;
```

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]

ANS:

(Customer)-[:PURCHASED]->(Plant)

(Customer)-[:PURCHASED]->(Fertilizer)

(Customer)-[:PURCHASED]->(Product)

(Customer)-[:RECOMMENDED]->(App)

(Supplier)-[:SUPPLIES]->(Plant)

(Supplier)-[:SUPPLIES]->(Fertilizer)

(Supplier)-[:SUPPLIES]->(Product)

```
CREATE (:Plant {name: 'Rose', type: 'Flowering', quantity: 700}),
 (:Plant {name: 'Lily', type: 'Flowering', quantity: 600}),
 (:Plant {name: 'Tulip', type: 'Flowering', quantity: 800}),
 (:Plant {name: 'Fern', type: 'Non-Flowering', quantity: 400}),
 (:Plant {name: 'Palm', type: 'Non-Flowering', quantity: 300});
CREATE (:Fertilizer {name: 'NPK', type: 'Granular', quantity: 1000}),
 (:Fertilizer {name: 'Compost', type: 'Organic', quantity: 800}),
 (:Fertilizer {name: 'Urea', type: 'Chemical', quantity: 1200}),
 (:Fertilizer {name: 'Bone Meal', type: 'Organic', quantity: 500}),
 (:Fertilizer {name: 'Potassium Nitrate', type: 'Chemical', quantity:
900});
CREATE (:Product {name: 'Garden Shears', type: 'Tool', quantity: 300}),
 (:Product {name: 'Plant Pots', type: 'Accessory', quantity: 400}),
 (:Product {name: 'Gardening Gloves', type: 'Accessory', quantity:
600}),
 (:Product {name: 'Watering Can', type: 'Tool', quantity: 350}),
 (:Product {name: 'Fertilizer Sprayer', type: 'Tool', quantity: 250});
CREATE (:Customer {name: 'John'}),
```

```
 (:Customer {name: 'Alice'}),
 (:Customer {name: 'Bob'}),
 (:Customer {name: 'Eva'}),
 (:Customer {name: 'Michael'});
CREATE (:Supplier {name: 'Green World'}),
 (:Supplier {name: 'Nature's Bounty'}),
 (:Supplier {name: 'Plant Paradise'}),
 (:Supplier {name: 'Blossom Garden'}),
 (:Supplier {name: 'Floral Land'});
MATCH (c:Customer {name: 'John'}), (p:Plant {name: 'Rose'})
CREATE (c)-[:PURCHASED {purchase_date: date('2024-03-27')}]->(p);
MATCH (c:Customer {name: 'Alice'}), (p:Plant {name: 'Lily'})
CREATE (c)-[:PURCHASED {purchase_date: date('2024-03-28')}]->(p);
MATCH (c:Customer {name: 'Bob'}), (p:Plant {name: 'Tulip'})
CREATE (c)-[:PURCHASED {purchase_date: date('2024-03-26')}]->(p);
MATCH (c:Customer {name: 'Eva'}), (p:Plant {name: 'Fern'})
CREATE (c)-[:PURCHASED {purchase_date: date('2024-03-27')}]->(p);
MATCH (c:Customer {name: 'Michael'}), (p:Plant {name: 'Palm'})
CREATE (c)-[:PURCHASED {purchase_date: date('2024-03-28')}]->(p);
MATCH (s:Supplier {name: 'Green World'}), (p:Plant {name: 'Rose'})
CREATE (s)-[:SUPPLIES]->(p);
MATCH (s:Supplier {name: 'Nature\'s Bounty'}), (p:Plant {name: 'Lily'})
CREATE (s)-[:SUPPLIES]->(p);
MATCH (s:Supplier {name: 'Plant Paradise'}), (p:Plant {name: 'Tulip'})
CREATE (s)-[:SUPPLIES]->(p);
MATCH (s:Supplier {name: 'Blossom Garden'}), (p:Plant {name: 'Fern'})
CREATE (s)-[:SUPPLIES]->(p);
MATCH (s:Supplier {name: 'Floral Land'}), (p:Plant {name: 'Palm'})
CREATE (s)-[:SUPPLIES]->(p);
```

```
MATCH p=()-[r]->()
RETURN p;
```

A. List the types of plants from your graph model:

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

B. List the popular flowering plants:

```
MATCH (p:Plant {type: 'Flowering'})-[r:PURCHASED]->(:Customer)
RETURN p.name, COUNT(r) AS popularity
ORDER BY popularity DESC;
```

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

```
MATCH (p:Plant)
WHERE p.quantity > 500 AND date() - duration('P2D') <= p.purchase_date
RETURN p.name;
```

D. List the names of suppliers in decreasing order who supplies "Creepers":

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

Slip 18

Consider the Course table

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

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

ANS:

```
INSERT INTO Courses (CourseID, Title, Instructor, Category, Price, Duration,
EnrollmentCount)
VALUES
 (1, 'Introduction to Python', 'John Doe', 'Programming', 49.99, '4 weeks',
1000),
 (2, 'Data Science Fundamentals', 'Alice Smith', 'Data Science', 69.99, '6
```

weeks', 800),

(3, 'Web Development with JavaScript', 'Michael Brown', 'Web Development', 59.99, '5 weeks', 600),

(4, 'Financial Accounting Basics', 'Emily Davis', 'Finance', 79.99, '8 weeks', 1200),

(5, 'Graphic Design Principles', 'Daniel Wilson', 'Design', 54.99, '4 weeks', 900);

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

ANS:

```
SELECT Instructor, COUNT(*) AS NumCourses
FROM Courses
GROUP BY Instructor
HAVING SUM(EnrollmentCount) > 500
ORDER BY NumCourses DESC;
```

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

ANS:

```
SELECT Category, AVG(EnrollmentCount) AS AvgEnrollment
FROM Courses
GROUP BY Category
HAVING AVG(EnrollmentCount) > 700
ORDER BY AvgEnrollment DESC;
```

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 buya 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]

ANS:

(LaptopManufacturer)-[:MANUFACTURES]->(LaptopModel)

(Customer)-[:BOUGHT]->(LaptopModel)

(Customer)-[:RECOMMENDS]->(LaptopModel)

```
CREATE (:LaptopManufacturer {name: 'DELL'}),
 (:LaptopManufacturer {name: 'HP'}),
 (:LaptopManufacturer {name: 'Apple'}),
 (:LaptopManufacturer {name: 'Lenovo'}),
 (:LaptopManufacturer {name: 'Asus'});
CREATE (:LaptopModel {name: 'DELL XPS 13', characteristics: 'Ultra-slim design, high-resolution display'}),
 (:LaptopModel {name: 'HP Spectre x360', characteristics: 'Convertible design, long battery life'}),
 (:LaptopModel {name: 'MacBook Pro', characteristics: 'Retina display, powerful performance'}),
 (:LaptopModel {name: 'Lenovo ThinkPad X1 Carbon', characteristics: 'Business-oriented, durable design'}),
 (:LaptopModel {name: 'Asus ROG Zephyrus G14', characteristics: 'Gaming laptop, high-performance hardware'});
CREATE (:Customer {name: 'John'}),
```

```
 (:Customer {name: 'Alice'}),
 (:Customer {name: 'Bob'}),
 (:Customer {name: 'Eva'}),
 (:Customer {name: 'Michael'});
MATCH (c:Customer {name: 'John'}), (l: LaptopModel {name: 'DELL XPS
13'})
CREATE (c)-[:BOUGHT {purchase_date: date('2023-01-26')}]->(l);
MATCH (c:Customer {name: 'Alice'}), (l: LaptopModel {name: 'HP Spectre
x360'})
CREATE (c)-[:BOUGHT {purchase_date: date('2023-01-26')}]->(l);
MATCH (c:Customer {name: 'Bob'}), (l: LaptopModel {name: 'MacBook
Pro'})
CREATE (c)-[:BOUGHT {purchase_date: date('2023-01-25')}]->(l);
MATCH (c:Customer {name: 'Eva'}), (l: LaptopModel {name: 'Lenovo
ThinkPad X1 Carbon'})
CREATE (c)-[:BOUGHT {purchase_date: date('2023-01-27')}]->(l);
MATCH (c:Customer {name: 'Michael'}), (l: LaptopModel {name: 'Asus
ROG Zephyrus G14'})
CREATE (c)-[:BOUGHT {purchase_date: date('2023-01-27')}]->(l);
MATCH (c:Customer {name: 'John'}), (l: LaptopModel {name: 'DELL XPS
13'})
CREATE (c)-[:RECOMMENDS {rating: 5}]->(l);
MATCH (c:Customer {name: 'Alice'}), (l: LaptopModel {name: 'HP Spectre
x360'})
CREATE (c)-[:RECOMMENDS {rating: 4}]->(l);
MATCH (c:Customer {name: 'Bob'}), (l: LaptopModel {name: 'MacBook
Pro'})
CREATE (c)-[:RECOMMENDS {rating: 5}]->(l);
MATCH (c:Customer {name: 'Eva'}), (l: LaptopModel {name: 'Lenovo
```

ThinkPad X1 Carbon'})

CREATE (c)-[:RECOMMENDS {rating: 4}]->(l);

MATCH (c:Customer {name: 'Michael'}), (l: LaptopModel {name: 'Asus

ROG Zephyrus G14'})

CREATE (c)-[:RECOMMENDS {rating: 5}]->(l);

MATCH p=()-[r]->()

RETURN p;

A. List the characteristics of laptop:

MATCH (l:LaptopModel)

RETURN l.characteristics;

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

MATCH (c:Customer)-[:BOUGHT]->(l:LaptopModel)-[:MANUFACTURES]-

>(m:LaptopManufacturer {name: 'DELL'})

RETURN DISTINCT c.name;

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

MATCH (c:Customer)-[r:BOUGHT]->(:LaptopModel)

WHERE r.purchase_date = date('2023-01-26')

RETURN DISTINCT c.name;

D. List the most recommended device:

MATCH (l:LaptopModel)<-[:RECOMMENDS]-()

WITH l, COUNT(*) AS recommendation_count

RETURN l.name, recommendation_count

ORDER BY recommendation_count DESC

LIMIT 1;

Slip19

Consider the Course table

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

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

ANS:

```
INSERT INTO Courses (CourseID, Title, Instructor, Category, Price, Duration,
EnrollmentCount)
VALUES
 (1, 'Introduction to Python', 'John Doe', 'Programming', 49.99, '4 weeks', 100),
 (2, 'Data Science Fundamentals', 'Alice Smith', 'Data Science', 69.99, '6 weeks', 80),
 (3, 'Web Development with JavaScript', 'Michael Brown', 'Web Development',
39.99, '5 weeks', 120),
 (4, 'Financial Accounting Basics', 'Emily Davis', 'Finance', 79.99, '8 weeks', 90),
 (5, 'Graphic Design Principles', 'Daniel Wilson', 'Design', 54.99, '4 weeks', 110);
```

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

ANS:

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

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

ANS:

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

ANS:

```
(Doctor)-[:VISITS]->(Hospital)
(Doctor)-[:OWNS]->(Clinic)
```

(Person)-[:RECOMMENDS]->(Doctor)

```
CREATE (:Doctor {name: 'Dr. John', specialization: 'Pediatric'}),
 (:Doctor {name: 'Dr. Alice', specialization: 'Gynaec'}),
 (:Doctor {name: 'Dr. Bob', specialization: 'Heart Specialist'}),
 (:Doctor {name: 'Dr. Eva', specialization: 'Orthopedic'}),
 (:Doctor {name: 'Dr. Michael', specialization: 'ENT'});
CREATE (:Hospital {name: 'Ruby Hospital', location: 'Seren Medows'}),
```

Model the following Doctor's information system as a graph model, and answer the following queries using Cypher.

Consider the doctors in and around Pune. Each Doctor is specialized in some stream like Pediatric, Gynaec, Heart Specialist, Cancer Specialist, ENT, etc. A doctor may be a visiting doctor across many hospitals or he may own a clinic. A person can provide a review/can recommend a doctor.

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

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

3. Answer the Queries

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

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

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

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

```
 (:Hospital {name: 'Sunshine Hospital', location: 'XYZ Area'}),
 (:Hospital {name: 'Unity Hospital', location: 'ABC Area'});
CREATE (:Clinic {name: 'Ortho Care Clinic', location: 'PQR Area'}),
 (:Clinic {name: 'Kids Health Clinic', location: 'Seren Medows'}),
 (:Clinic {name: 'Heartbeat Clinic', location: 'LMN Area'});
MATCH (d:Doctor {name: 'Dr. John'}), (h:Hospital {name: 'Ruby Hospital'})
CREATE (d)-[:VISITS]->(h);
```

```
MATCH (d:Doctor {name: 'Dr. Alice'}), (h:Hospital {name: 'Sunshine
Hospital'})
CREATE (d)-[:VISITS]->(h);
MATCH (d:Doctor {name: 'Dr. Bob'}), (h:Hospital {name: 'Unity Hospital'})
CREATE (d)-[:VISITS]->(h);
MATCH (d:Doctor {name: 'Dr. Eva'}), (c:Clinic {name: 'Ortho Care Clinic'})
CREATE (d)-[:OWNS]->(c);
MATCH (d:Doctor {name: 'Dr. Michael'}), (c:Clinic {name: 'Kids Health
Clinic'})
CREATE (d)-[:OWNS]->(c);
MATCH p=()-[r]->()
RETURN p;
```

A. List the Orthopedic doctors in a specific area:

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

B. List the doctors who has specialization in a specific area:

```
MATCH (d:Doctor {specialization: 'Pediatric'})
RETURN d.name;
```

C. List the most recommended Pediatrics in a specific area:

```
MATCH (d:Doctor {specialization: 'Pediatric'})<-[r:RECOMMENDS]-
(:Person)
WHERE r.rating = 5 AND d.location = 'Seren Medows'
RETURN d.name;
```

D. List all the doctors who visits more than 2 hospitals:

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

Slip20

Consider the Course table

Courses(CourseID,Title,Instructor,Category,Price,Duration,EnrollmentCount) Write a query to insert 5 rows in it that as per the query requirements.

ANS:

```
INSERT INTO Courses (CourseID, Title, Instructor, Category, Price, Duration,
EnrollmentCount)
VALUES
 (1, 'Introduction to Python', 'John Doe', 'Programming', 49.99, '4 weeks', 1000),
 (2, 'Data Science Fundamentals', 'Alice Smith', 'Data Science', 69.99, '6 weeks', 800),
 (3, 'Web Development with JavaScript', 'Michael Brown', 'Web Development', 59.99, '5
weeks', 1200),
 (4, 'Financial Accounting Basics', 'Emily Davis', 'Finance', 79.99, '8 weeks', 1500),
 (5, 'Graphic Design Principles', 'Daniel Wilson', 'Design', 54.99, '4 weeks', 1000);
```

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

ANS:

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

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

ANS:

```
SELECT Category
FROM (
 SELECT Category, COUNT(DISTINCT CourseID) AS CourseCount,
COUNT(DISTINCT Instructor) AS InstructorCount
```

```
 FROM Courses
 GROUP BY Category
) AS CategoryCounts
WHERE CourseCount = InstructorCount
ORDER BY Category;
```

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]

ANS:

(Author)-[:WRITES]->(Book)

(Publisher)-[:PUBLISHES]->(Book)

(Reader)-[:READS]->(Book)

(Reader)-[:RECOMMENDS]->(Book)

(Reader)-[:PROVIDES_REVIEW]->(Book)

```
CREATE (:Author {name: 'Author1'})
CREATE (:Author {name: 'Author2'})
CREATE (:Author {name: 'Author3'})
CREATE (:Book {title: 'Book1', category: 'Comics', rating: 4.5})
CREATE (:Book {title: 'Book2', category: 'Fiction', rating: 3.8})
```

```
CREATE (:Book {title: 'Book3', category: 'Comics', rating: 4.0})
CREATE (:Publisher {name: 'Publisher1'})
CREATE (:Publisher {name: 'Publisher2'})
CREATE (:Publisher {name: 'Publisher3'})
CREATE (:Reader {name: 'Reader1'})
CREATE (:Reader {name: 'Reader2'})
CREATE (:Reader {name: 'Reader3'})
MATCH (a:Author {name: 'Author1'}), (b:Book {title: 'Book1'})
CREATE (a)-[:WRITES]->(b)
MATCH (a:Author {name: 'Author2'}), (b:Book {title: 'Book2'})
CREATE (a)-[:WRITES]->(b)
MATCH (a:Author {name: 'Author3'}), (b:Book {title: 'Book3'})
CREATE (a)-[:WRITES]->(b)
MATCH (p:Publisher {name: 'Publisher1'}), (b:Book {title: 'Book1'})
CREATE (p)-[:PUBLISHES]->(b)
MATCH (p:Publisher {name: 'Publisher2'}), (b:Book {title: 'Book2'})
CREATE (p)-[:PUBLISHES]->(b)
MATCH (p:Publisher {name: 'Publisher3'}), (b:Book {title: 'Book3'})
CREATE (p)-[:PUBLISHES]->(b)
MATCH (r:Reader {name: 'Reader1'}), (b:Book {title: 'Book1'})
CREATE (r)-[:READS]->(b)
MATCH (r:Reader {name: 'Reader2'}), (b:Book {title: 'Book2'})
CREATE (r)-[:READS]->(b)
MATCH (r:Reader {name: 'Reader3'}), (b:Book {title: 'Book3'})
CREATE (r)-[:READS]->(b)
```

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

```
MATCH (a:Author)-[:WRITES]->(b:Book)
WHERE b.category = 'Comics'
RETURN DISTINCT a.name;
```

B. Count no. of readers of book published by “Sage”.

```
MATCH (p:Publisher {name: 'Sage'})-[:PUBLISHES]->(b:Book)<-[:READS]-(r:Reader)
RETURN COUNT(DISTINCT r) AS reader_count;
```

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

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

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

```
MATCH (r:Reader)-[rv:PROVIDES_REVIEW]->(b:Book)
WHERE rv.rating >= 3
RETURN DISTINCT r.name;
```