Scenario 1:

db.Professor.insertMany([{profid:101,name:"Vijay",age:23,rank:2,speciality:"Python",Project:[202,203],grad_student:[501,502]},
{profid:102,name:"Vishal",age:23,rank:2,speciality:"Python",Project:[202,201,203,204],grad_student:[503,504]},
{profid:103,name:"Kunal",age:23,rank:2,speciality:"Python",Project:[204,201],grad_student:[505]}])
{

db.Project.insertMany([{pro_num:201,spo_name:"UGC",start_date:ISODate("2005-10-12"),end_date:ISODate("2010-11-12"),budget:200000,grad_student:[505,501],professor:103},
{pro_num:202,spo_name:"AICTE",start_date:ISODate("2006-10-12"),end_date:ISODate("2010-11-12"),budget:5000000,grad_student:[501,502,503],professor:101},
{pro_num:203,spo_name:"UGC",start_date:ISODate("2007-10-12"),end_date:ISODate("2011-11-12"),budget:30000,grad_student:[502,501],professor:101},
{pro_num:204,spo_name:"UGC",start_date:ISODate("2008-10-12"),end_date:ISODate("2013-11-12"),budget:30000,grad_student:[505,502],professor:103}])
{

db.grad_student.insertMany([{usn:501,g_name:"Anu",age:25,degree_program:"Bsc"},
{usn:502,g_name:"Varu",age:25,degree_program:"Bca"},
{usn:503,g_name:"Yashu",age:25,degree_program:"Bsc"},
{usn:504,g_name:"Arush;",age:25,degree_program:"Bsc"},
{usn:505,g_name:"Hanu",age:25,degree_program:"Bsc"}])

1) Professors without ongoing projects of more than 1 lakh

```
db.Project.aggregate([
  {
    $lookup: {
      from: "Professor",
      localField: "professor",
      foreignField: "profid",
      as: "professors"
    }
  },
  {
    $match: {
      $and: [
        { "end_date": { $lt: new Date() } },
        { "budget": { $lte: 100000 } }
      ]
    }
  },
  {
    $group: {
      _id: 0,
      professors: { $addToSet: "$professors.name" }
    }
  },
  {
    $project: {
      _id: 0,
      professor: "$professors"
    }
  }
```

])

2) Graduate students, their professors, and project sponsors

```
db.grad_student.aggregate([
   {
      $lookup: {
         from: "Project",
         localField: "usn",
         foreignField: "grad_student",
         as: "projects"
      }
   },
   {
      $lookup: {
         from: "Professor",
         localField: "usn",
         foreignField: "grad_student",
         as: "professors"
      }
   },
   {
      $project: {
         _id: 0,
         student_name: "$g_name",
         professor_name: "$professors.name",
         project_sponsor: "$projects.spo_name"
      }
   }
])
```

3) Professors and sum of budgets of projects started after 2005 but ended in 2010

```
db.Professor.aggregate([
   {
      $lookup: {
         from: "Project",
         localField: "profid",
         foreignField: "professor",
         as: "projects"
      }
   },
   {
      $unwind: "$projects"
   },
   {
      $match: {
         "projects.start_date": { $gte: ISODate("2005-01-01"), $lte: ISODate("2010-12-31") }
      }
   },
   {
      $group: {
         _id: "$name",
```

```
        total_budget: { $sum: "$projects.budget" }
      }
    }
])
```

5) Professors working on all projects

```
db.Professor.aggregate([
        {
                $lookup: {
                                from: "Project",
                                localField: "Project",
                               foreignField: "pro_num",
                               as: "projects"
                        }
         },
        {
                 $addFields: {
                                projectCount: {
                                        $size: "$projects"
        } } },
        {
                $match: {
                        projectCount: allProjectsCount
        } },
         {
                $project: {
                        _id: 0,
                         name: 1
        } }] )
```

Scenario 2)

1) List the details of customers who have a joint account and also have at least one loan

```
db.customers.aggregate([
  {
    $lookup: {
      from: "cust_ac",
      localField: "ssn",
      foreignField: "ssn",
      as: "account"
    }
  },
  {
    $lookup: {
      from: "cust_loan",
      localField: "ssn",
      foreignField: "ssn",
      as: "loan"
    }
  },
  {
    $match: {
      $and: [
        { "account.acc_no": { $in: db.account.distinct("ac_no", { ac_type: "joint" }) } },
        { "loan.lno": { $exists: true } }
      ]
    }
  },
  {
    $project: {
      ssn: 1,
      name: 1,
      address: 1,
      phone: 1
    }
  }
]);
```

2) List the details of the branch that has given the maximum loan

```
db.loan.aggregate([
        {
                $sort: {
                        amount: -1
        } },
         {
                $limit: 1
        },
        {
                $lookup: {
                        from: "branch",
```

```
                        localField: "bid",
                        foreignField: "_id",
                        as: "branchDetails"
        } },
        {
                $unwind: "$branchDetails"
        },
        {
                $project: {
                        _id: 0,
                        "Branch Name": "$branchDetails.name",
                        "Branch Address": "$branchDetails.address",
                        "Branch Phone": "$branchDetails.phone"
        }}] );
```

3) List the details of saving accounts opened in the SBI branches located in Bangalore

```
db.account.find({
  ac_type: 'sb',
  bid: {
    $in: db.branch.find({
     b_code: 'SBI',
     address: /Bangalore/
    }).toArray().map(b => b._id)
  }
});
```

4) List the name of the branch along with its bank name and the total amount of loan given by it

```
db.branch.aggregate([
  {
    $lookup: {
      from: "bank",
      localField: "b_code",
      foreignField: "code",
      as: "bankDetails"
    }
  },
  {
    $lookup: {
      from: "loan",
      localField: "_id",
      foreignField: "bid",
      as: "branchLoans"
    }
  },
  {
    $project: {
      _id: 0,
      "Branch Name": "$name",
```

```
    "Bank Name": { $arrayElemAt: ["$bankDetails.name", 0] },
    "Total Loan Amount": { $sum: "$branchLoans.amount" }
   }
  }
]);
```

Scenario 3)

1) List the details of horror movies released in 2012 and directed by more than 2 directors.

```
db.movies.find({ genres: 'horror', yor: 2012, $expr: { $gt: [{ $size: '$directors' }, 2] } });
```

2) List the details of actors who acted in movies having same titles but released before 2000 and after 2010.db.movie.aggregate([

```
{$lookup:
        {from:"pcompany",
        localField:"pcompany",
        foreignField:"name",
        as:"pcom"}
},
{$group:
        {_id:"$pcompany",
        count:{$sum:1},
        pc:{$addToSet:"$pcom"}}
},
{$sort:{count:-1}},
{$limit:1},
{$project:
        {_id:0,
        "pc._id":0}
}])
```

```
db.movies.aggregate([
  {
    $match: {
      $or: [
        { yor: { $gt: 2010 } },
```

```
            { yor: { $lt: 2000 } }
          ]
        }
    },
    {
        $lookup: {
            from: "actors",
            localField: "title",
            foreignField: "name",
            as: "actors"
        }
    },
    {
        $unwind: "$actors"
    },
    {
        $match: {
            $expr: {
                $eq: ["$title", "$actors.name"]
            }
        }
    },
    {
        $project: {
            _id: 0,
            actorName: "$actors.name",
            movieName: "$title"
        }
    }
]);
```

3) List the details of production companies producing maximum movies (consider the scenario if 2 productions produced the same number of movies).

db.movies.aggregate([{ $lookup: { from: "production_companies", localField: "_id", foreignField: "movies", as: "pcom" } }, { $group: { _id: "$_id", count: { $sum: 1 }, pc: { $addToSet: "$pcom" } } }, { $sort: { count: -1 } }, { $limit: 1 }, { $project: { _id: 0, "pc._id": 0 } }])

4) List the details of movies where the actor and director have the same date of birth

```
db.movies.aggregate([ { $lookup: { from: "director", localField: "director", foreignField: "id", as: "directors" } }, { $lookup: { from: "actor", localField: "_id", foreignField: "movies", as: "actors" } }, { $match: { $expr: { $eq: ["$actors.dob", "$directors.dob"] } } }, { $project: { _id: 0, "directors._id": 0, "actors._id": 0 } }])
```

Scenario 4)

1) List the state name which has the maximum number of tourist places.

```
db.TouristPlaces.aggregate([
{ $group: { _id: "$state", count: { $sum: 1 } } },
{ $sort: { count: -1 } },
{ $limit: 1 }
]);
```

2) List details of Tourist places where the maximum number of tourists visited.

```
db.TouristVisits.aggregate([
{ $group: { _id: "$place_id", count: { $sum: 1 } } },
{ $sort: { count: -1 } },
{ $limit: 1 },
{
$lookup: {
from: "TouristPlaces",
localField: "_id",
foreignField: "_id",
```

```
as: "touristPlaceDetails"
}
},
{ $unwind: "$touristPlaceDetails" },
{ $project: { _id: "$touristPlaceDetails._id", name: "$touristPlaceDetails.name", state:
"$touristPlaceDetails.state", num_visits: "$count" } }])
```

3) List the details of tourists visiting all tourist places of the state “KARNATAKA”.

```
db.Tourists.find({
country: "India",
_id: {
$nin: db.TouristVisits.distinct("tourist_id", { place_id: { $in: db.TouristPlaces.find({ state:
"KARNATAKA" }).map(function(doc) { return doc._id; }) } })
}
});
```

5th program

```
=import pymongo

client = pymongo.MongoClient("mongodb://127.0.0.1:27017/")
db = client["mydatabase"]
collection = db["employee"]

def create_record():
    emp_id = input("Enter Employee ID: ")
    name = input("Enter Employee Name: ")
    position = input("Enter Employee Position: ")
    salary = float(input("Enter Employee Salary: "))
    record = {"emp_id": emp_id, "name": name, "position": position, "salary": salary}
    collection.insert_one(record)
    print("Record created successfully!")

def read_records():
    for record in collection.find():
```

```
        emp_id = record.get("emp_id", "N/A")
        name = record.get("name", "N/A")
        position = record.get("position", "N/A")
        salary = record.get("salary", "N/A")
        print(f"Employee ID: {emp_id}, Name: {name}, Position: {position}, Salary: {salary}")

def update_record():
    emp_id_to_update = input("Enter the Employee ID to update: ")
    print("\nSelect the Field to update:")
    print("1. Employee ID")
    print("2. Name")
    print("3. Position")
    print("4. Salary")
    print("5. Exit")

    choice = input("Enter your choice: ")

    if choice == "1":
        new_emp_id = input("Enter the new Employee ID: ")
        collection.update_one({"emp_id": emp_id_to_update}, {"$set": {"emp_id": new_emp_id}})
    elif choice == "2":
        new_name = input("Enter the new Name: ")
        collection.update_one({"emp_id": emp_id_to_update}, {"$set": {"name": new_name}})
    elif choice == "3":
        new_position = input("Enter the new Position: ")
        collection.update_one({"emp_id": emp_id_to_update}, {"$set": {"position": new_position}})
    elif choice == "4":
        new_salary = float(input("Enter the new Salary: "))
        collection.update_one({"emp_id": emp_id_to_update}, {"$set": {"salary": new_salary}})
    elif choice == "5":
        pass
    else:
        print("Invalid choice")

def delete_record():
    emp_id_to_delete = input("Enter the Employee ID to delete: ")
    collection.delete_one({"emp_id": emp_id_to_delete})
    print("Record deleted successfully!")
```

```
while True:
    print("\nEmployee Database Menu:")
    print("1. Create Employee Record")
    print("2. Read Employee Records")
    print("3. Update Employee Record")
    print("4. Delete Employee Record")
    print("5. Exit")

    choice = input("Enter your choice: ")

    if choice == "1":
        create_record()
    elif choice == "2":
        read_records()
    elif choice == "3":
        update_record()
    elif choice == "4":
        delete_record()
    elif choice == "5":
        break
    else:
        print("Invalid choice.")
```

==========

```
import pymongo

# Establish a connection to MongoDB
client = pymongo.MongoClient("mongodb://127.0.0.1:27017/")
db = client["mydatabase"]
collection = db["student"]

def create_record():
        usn = input("Enter USN: ")
        name = input("Enter name: ")
        age = int(input("Enter age: "))
        address = input("Enter Address: ")
```

```
    addhar = input("Enter Aadhar Number: ")
    record = {"usn": usn, "name": name, "age": age, "address": address, "addhar": addhar}
    collection.insert_one(record)
    print("Record created successfully!")

def read_records():
    for record in collection.find():
        if 'usn' in record:
            usn = record['usn']
        else:
            usn = "N/A"

        if 'name' in record:
            name = record['name']
        else:
            name = "N/A"

        if 'age' in record:
            age = record['age']
        else:
            age = "N/A"

        if 'address' in record:
            address = record['address']
        else:
            address = "N/A"

        if 'addhar' in record:
            addhar = record['addhar']
        else:
            addhar = "N/A"

        print(f"USN: {usn}, Name: {name}, Age: {age}, Address: {address}, Aadhar: {addhar}")

def update_record():
    usn_to_update = input("Enter the USN to update: ")
    print("\t")
    while True:
```

```
        print("\nSelect the Field to update:")
        print("1. usn")
        print("2. name")
        print("3. age")
        print("4. address")
        print("5. aadhar")
        print("6. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            new_usn = input("Enter the new usn: ")
            collection.update_one({"usn": usn_to_update}, {"$set": {"usn": new_usn}})
        elif choice == "2":
            new_name = input("Enter the new Name: ")
            collection.update_one({"usn": usn_to_update}, {"$set": {"name": new_name}})

        elif choice == "3":
            new_age = input("Enter the new Age: ")
            collection.update_one({"usn": usn_to_update}, {"$set": {"age": new_age}})

        elif choice == "4":
            new_address = input("Enter the new Address: ")
            collection.update_one({"usn": usn_to_update}, {"$set": {"address": new_address}})

        elif choice == "5":
            new_addhar = input("Enter the new Addhar: ")
            collection.update_one({"usn": usn_to_update}, {"$set": {"addhar": new_addhar}})

        elif choice == "6":
            break
        else:
            print("Invalid choice")
    print("Successfully Updated")
def delete_record():
    usn_to_delete = input("Enter the USN to delete: ")
    collection.delete_one({"usn": usn_to_delete})
    print("Record deleted successfully!")
```

```
while True:
    print("\nMenu:")
    print("1. Create Record")
    print("2. Read Records")
    print("3. Update Record")
    print("4. Delete Record")
    print("5. Exit")

    choice = input("Enter your choice: ")

    if choice == "1":
        create_record()
    elif choice == "2":
        read_records()
    elif choice == "3":
        update_record()
    elif choice == "4":
        delete_record()
    elif choice == "5":
        break
    else:
        print("Invalid choice.")
```