Day - 5 MongoDB Assignment

Name : U Ganesh Chowdari
Email : ganeshchowdary077@gmail.com

CREATE DATABASE:

```
test> use InsuranceDB
switched to db InsuranceDB
```

CREATE COLLECTIONS:

```
InsuranceDB> db.createCollection('customers')
{ ok: 1 }
InsuranceDB> db.createCollection('policies')
{ ok: 1 }
```

INSERT / CREATE:

Insert multiple customer documents with name, email, age, city, active status, and created date so that queries return multiple rows.

```
InsuranceDB> db.customers.insertMany([
...   { firstName:"Rahul", lastName:"Sharma", email:"rahul@gmail.com", age:28, city:"Delhi", isActive:true, createdAt:new Date() },
...   { firstName:"Amit", lastName:"Verma", email:"amit@gmail.com", age:35, city:"Mumbai", isActive:true, createdAt:new Date() },
...   { firstName:"Neha", lastName:"Singh", email:"neha@gmail.com", age:22, city:"Delhi", isActive:false, createdAt:new Date() },
...   { firstName:"Pooja", lastName:"Iyer", email:"pooja@gmail.com", age:31, city:"Chennai", isActive:true, createdAt:new Date() },
...   { firstName:"Karan", lastName:"Mehta", email:"karan@gmail.com", age:42, city:"Mumbai", isActive:true, createdAt:new Date() }
... ]);
```

Output:

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6957ad9abcb9fcc4f71e2626'),
    '1': ObjectId('6957ad9abcb9fcc4f71e2627'),
    '2': ObjectId('6957ad9abcb9fcc4f71e2628'),
    '3': ObjectId('6957ad9abcb9fcc4f71e2629'),
    '4': ObjectId('6957ad9abcb9fcc4f71e262a')
  }
}
```

Insert multiple policy documents with different policy types, premium amounts, durations, and active status

```
InsuranceDB> db.policies.insertMany([
...   { policyName:"Health Secure", policyType:"Health", premiumAmount:18000, durationYears:1, isActive:true },
...   { policyName:"Health Plus", policyType:"Health", premiumAmount:30000, durationYears:2, isActive:true },
...   { policyName:"Life Shield", policyType:"Life", premiumAmount:25000, dudurationYears:10, isActive:true },
...   { policyName:"Life Premium", policyType:"Life", premiumAmount:40000, ddurationYears:20, isActive:false },
...   { policyName:"Budget Health", policyType:"Health", premiumAmount:12000, durationYears:1, isActive:true }
... ]);
```

Output:

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6957ad9abcb9fcc4f71e2626'),
    '1': ObjectId('6957ad9abcb9fcc4f71e2627'),
    '2': ObjectId('6957ad9abcb9fcc4f71e2628'),
    '3': ObjectId('6957ad9abcb9fcc4f71e2629'),
    '4': ObjectId('6957ad9abcb9fcc4f71e262a')
  }
}
```

BASIC READ (find / findOne)

Fetch all customers displaying only first name and city
InsuranceDB> db.customers.find({}, { firstName:1, city:1, _id:0 })

Output:

```
[
  { firstName: 'Rahul', city: 'Delhi' },
  { firstName: 'Amit', city: 'Mumbai' },
  { firstName: 'Neha', city: 'Delhi' },
  { firstName: 'Pooja', city: 'Chennai' },
  { firstName: 'Karan', city: 'Mumbai' }
]
```

Fetch a single customer using email and display only name and age
InsuranceDB> db.customers.findOne({ email:"rahul@gmail.com" },
{ firstName:1, age:1, _id:0 })

Output:

```
{ firstName: 'Rahul', age: 28 }
```

Fetch all policies displaying policy name and premium amount
InsuranceDB> db.policies.find({}, { policyName:1, premiumAmount:1,
_id:0 })

Output:

```
[
  { policyName: 'Health Secure', premiumAmount: 18000 },
  { policyName: 'Health Plus', premiumAmount: 30000 },
  { policyName: 'Life Shield', premiumAmount: 25000 },
  { policyName: 'Life Premium', premiumAmount: 40000 },
  { policyName: 'Budget Health', premiumAmount: 12000 }
]
```

COMPARISON OPERATORS

Fetch customers whose age is greater than 30 and display name and age

InsuranceDB> db.customers.find({ age: { $gt: 30 } }, { firstName:1, age:1, __id:0 })

Output:

```
[
  { firstName: 'Amit', age: 35 },
  { firstName: 'Pooja', age: 31 },
  { firstName: 'Karan', age: 42 }
]
```

Fetch policies whose premium amount lies between 15000 and 30000

InsuranceDB> db.policies.find({ premiumAmount: { $gte:15000, $lte:30000 } }, { policyName:1, premiumAmount:1, _id:0 })

Output:

```
[
  { policyName: 'Health Secure', premiumAmount: 18000 },
  { policyName: 'Health Plus', premiumAmount: 30000 },
  { policyName: 'Life Shield', premiumAmount: 25000 }
]
```

Fetch customers who do not belong to Delhi

InsuranceDB> db.customers.find({ city: { $ne:"Delhi" } }, { firstName:1, citcity:1, _id:0 })

Output:

```
[
  { firstName: 'Amit', city: 'Mumbai' },
  { firstName: 'Pooja', city: 'Chennai' },
  { firstName: 'Karan', city: 'Mumbai' }
]
```

LOGICAL OPERATORS ($and, $or, $not)

Fetch active customers from Delhi

InsuranceDB> db.customers.find({ $and:[{city:"Delhi"}, {isActive:true}] }, { firstName:1, _id:0 })

Output:

```
[ { firstName: 'Rahul' } ]
```

Fetch customers who belong to either Delhi or Mumbai

InsuranceDB> db.customers.find({ $or:[{city:"Delhi"}, {city:"Mumbai"}] }, { firstName:1, city:1, _id:0 })

Output:

```
[
  { firstName: 'Rahul', city: 'Delhi' },
  { firstName: 'Amit', city: 'Mumbai' },
  { firstName: 'Neha', city: 'Delhi' },
  { firstName: 'Karan', city: 'Mumbai' }
]
```

Fetch policies whose premium is NOT greater than 30000

InsuranceDB> db.policies.find({ premiumAmount:{ $not:{ $gt:30000 }} }, { polpolicyName:1, premiumAmount:1, _id:0 })

Output:

```
[
  { policyName: 'Health Secure', premiumAmount: 18000 },
  { policyName: 'Health Plus', premiumAmount: 30000 },
  { policyName: 'Life Shield', premiumAmount: 25000 },
  { policyName: 'Budget Health', premiumAmount: 12000 }
]
```

IN / NOT IN OPERATORS

Fetch customers whose city belongs to a given list of cities

InsuranceDB> db.customers.find({ city:{ $in:["Delhi","Mumbai"] } }, { firstNfirstName:1, city:1, _id:0 })

Output:

```
[
  { firstName: 'Rahul', city: 'Delhi' },
  { firstName: 'Amit', city: 'Mumbai' },
  { firstName: 'Neha', city: 'Delhi' },
  { firstName: 'Karan', city: 'Mumbai' }
]
```

Fetch customers whose city does not belong to a given city list

InsuranceDB> db.customers.find({ city:{ $nin:["Delhi"] } }, { firstName:1, ccity:1, _id:0 })

Output:

```
[
  { firstName: 'Amit', city: 'Mumbai' },
  { firstName: 'Pooja', city: 'Chennai' },
  { firstName: 'Karan', city: 'Mumbai' }
]
```

PROJECTION

Fetch all customers displaying only first name and email

InsuranceDB> db.customers.find({}, { firstName:1, email:1, _id:0 })

Output:

```
[
  { firstName: 'Rahul', email: 'rahul@gmail.com' },
  { firstName: 'Amit', email: 'amit@gmail.com' },
  { firstName: 'Neha', email: 'neha@gmail.com' },
  { firstName: 'Pooja', email: 'pooja@gmail.com' },
  { firstName: 'Karan', email: 'karan@gmail.com' }
]
```

Fetch customer details excluding the email field

InsuranceDB> db.customers.find({}, { email:0 })

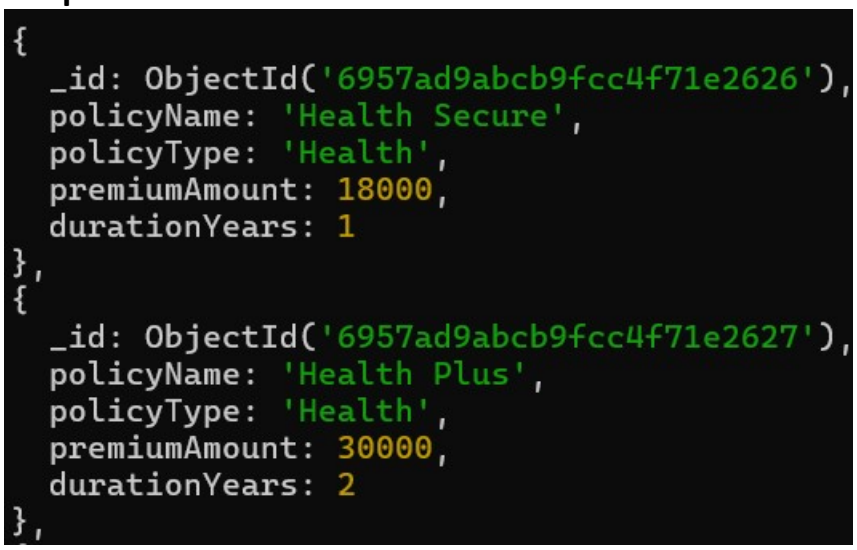
Output:

```
{
  _id: ObjectId('6957ad8fbcb9fcc4f71e2621'),
  firstName: 'Rahul',
  lastName: 'Sharma',
  age: 28,
  city: 'Delhi',
  isActive: true,
  createdAt: ISODate('2026-01-02T11:35:43.094Z')
},
{
  _id: ObjectId('6957ad8fbcb9fcc4f71e2622'),
  firstName: 'Amit',
  lastName: 'Verma',
  age: 35,
  city: 'Mumbai',
  isActive: true,
  createdAt: ISODate('2026-01-02T11:35:43.094Z')
},
```

Fetch policy details excluding internal fields like created date and active status

InsuranceDB> db.policies.find({}, { createdAt:0, isActive:0 })

Output:



SORT, LIMIT, PAGINATION

Fetch customers sorted by age in ascending order

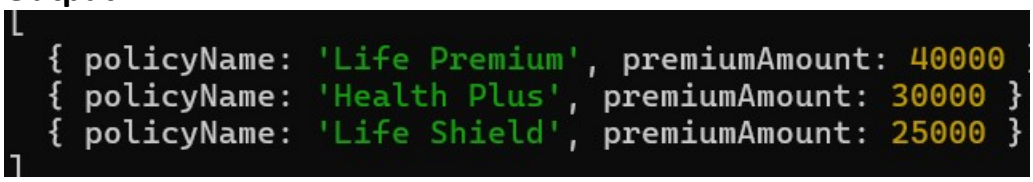
InsuranceDB> db.customers.find({}, { firstName:1, age:1, _id:0 }).sort({age:1 })

Output:

Fetch top three policies with the highest premium amounts

InsuranceDB> db.policies.find({}, { policyName:1, premiumAmount:1, _id:0 }).sort({ premiumAmount:-1 }).limit(3)

Output:



Fetch customers using pagination with skip and limit

InsuranceDB> db.customers.find({}, { firstName:1, _id:0 }).skip(2).limit(2)

Output:

```
[ { firstName: 'Neha' }, { firstName: 'Pooja' } ]
```

UPDATE OPERATIONS

Update the city of a customer using email as identifier

InsuranceDB> db.customers.updateOne({ email:"rahul@gmail.com" },
{ $set:{ cicity:"Bangalore" } })

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Increase premium amount of all Health policies by 10 percent

InsuranceDB> db.policies.updateMany({ policyType:"Health" },
{ $mul:{ premiupremiumAmount:1.1 } })

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
```

Deactivate customers whose age is below 25

InsuranceDB> db.customers.updateMany({ age:{ $lt:25 } },
{ $set:{ isActive:false } })

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
```

DELETE OPERATIONS

Delete a customer using email

```
InsuranceDB> db.customers.deleteOne({ email:"test@gmail.com" })
```

Output:

```
{ acknowledged: true, deletedCount: 0 }
```

Delete all inactive customers

```
InsuranceDB> db.customers.deleteMany({ isActive:false })
```

Output:

```
{ acknowledged: true, deletedCount: 1 }
```

Delete policies whose premium amount is less than 10000

```
InsuranceDB> db.policies.deleteMany({ premiumAmount:{ $lt:10000 } })
```

Output:

```
{ acknowledged: true, deletedCount: 0 }
```

AGGREGATION

Count the number of customers in each city and display clean output

```
InsuranceDB> db.customers.aggregate([
...   { $group:{ _id:"$city", count:{ $sum:1 } } },
...   { $project:{ _id:0, city:"$_id", count:1 } }
... ])
```

Output:

```
[
  { count: 1, city: 'Chennai' },
  { count: 1, city: 'Bangalore' },
  { count: 2, city: 'Mumbai' }
]
```

Calculate the average premium amount for each policy type

```
InsuranceDB> db.policies.aggregate([
...   { $group:{ _id:"$policyType",
avgPremium:{ $avg:"$premiumAmount" } } },
...   { $project:{ _id:0, policyType:"$_id", avgPremium:1 } }
... ])
```

Output:

```
[
  { avgPremium: 22000, policyType: 'Health' }
  { avgPremium: 32500, policyType: 'Life' }
]
```

HAVING EQUIVALENT (match after group)

Display only those cities where customer count is greater than one

```
InsuranceDB> db.customers.aggregate([
...   { $group:{ _id:"$city", count:{ $sum:1 } } },
...   { $match:{ count:{ $gt:1 } } },
...   { $project:{ _id:0, city:"$_id", count:1 } }
... ])
```

Output:

```
[ { count: 2, city: 'Mumbai' } ]
```

CONDITIONAL LOGIC (CASE equivalent)

Categorize customers into age groups such as Young and Adult

```
InsuranceDB> db.customers.aggregate([
...   {
...     $project:{
...       firstName:1,
...       ageGroup:{
...         $cond:[ { $lt:["$age",30] }, "Young", "Adult" ]
...       }
...     }
...   }
... ])
```

Output:

```
[
  {
    _id: ObjectId('6957ad8fbcb9fcc4f71e2621'),
    firstName: 'Rahul',
    ageGroup: 'Young'
  },
  {
    _id: ObjectId('6957ad8fbcb9fcc4f71e2622'),
    firstName: 'Amit',
    ageGroup: 'Adult'
  },
  {
    _id: ObjectId('6957ad8fbcb9fcc4f71e2624'),
    firstName: 'Pooja',
    ageGroup: 'Adult'
  },
  {
    _id: ObjectId('6957ad8fbcb9fcc4f71e2625'),
    firstName: 'Karan',
    ageGroup: 'Adult'
  }
```

Categorize policies into High, Medium, and Low premium categories

```
InsuranceDB> db.policies.aggregate([
...   {
...     $project:{
...       policyName:1,
...       category:{
...         $switch:{
...           branches:[
...             { case:{ $gt:["$premiumAmount",30000] }, then:"High" },
...             { case:{ $gt:["$premiumAmount",15000] }, then:"Medium" }
...           ],
...           default:"Low"
...         }
...       }
...     }
...   }
... ])
```

Output:

```
{
  _id: ObjectId('6957ad9abcb9fcc4f71e2626'),
  policyName: 'Health Secure',
  category: 'Medium'
},
{
  _id: ObjectId('6957ad9abcb9fcc4f71e2627'),
  policyName: 'Health Plus',
  category: 'High'
},
{
  _id: ObjectId('6957ad9abcb9fcc4f71e2628'),
  policyName: 'Life Shield',
  category: 'Medium'
},
{
  _id: ObjectId('6957ad9abcb9fcc4f71e2629'),
  policyName: 'Life Premium',
  category: 'High'
},
{
  _id: ObjectId('6957ad9abcb9fcc4f71e262a'),
  policyName: 'Budget Health',
  category: 'Low'
}
```

EXISTS / TYPE / REGEX OPERATORS

Fetch customers where the age field exists

InsuranceDB> db.customers.find({ age:{ $exists:true } }, { firstName:1, _id:0 })

Output:

```
[
  { firstName: 'Rahul' },
  { firstName: 'Amit' },
  { firstName: 'Pooja' },
  { firstName: 'Karan' }
]
```

Fetch customers whose email contains the word "gmail"

InsuranceDB> db.customers.find({ email:{ $regex:"gmail", $options:"i" } }, { firstName:1, email:1, _id:0 })

Output:

```
[
  { firstName: 'Rahul', email: 'rahul@gmail.com' },
  { firstName: 'Amit', email: 'amit@gmail.com' },
  { firstName: 'Pooja', email: 'pooja@gmail.com' },
  { firstName: 'Karan', email: 'karan@gmail.com' }
]
```

Fetch customers where the age field is of integer type

InsuranceDB> db.customers.find({ age:{ $type:"int" } }, { firstName:1, age:1, _id:0 })

Output:

```
[
  { firstName: 'Rahul', age: 28 },
  { firstName: 'Amit', age: 35 },
  { firstName: 'Pooja', age: 31 },
  { firstName: 'Karan', age: 42 }
]
```

INDEXING

Create a unique index on customer email

InsuranceDB> db.customers.createIndex({ email:1 }, { unique:true })

Output:

```
email_1
```

Create an index on city field for faster filtering
InsuranceDB> db.customers.createIndex({ city:1 })

Output:
```
city_1
```

Create an index on policy type for performance improvement
InsuranceDB> db.policies.createIndex({ policyType:1 })

Output:
```
policyType_1
```