Week 8

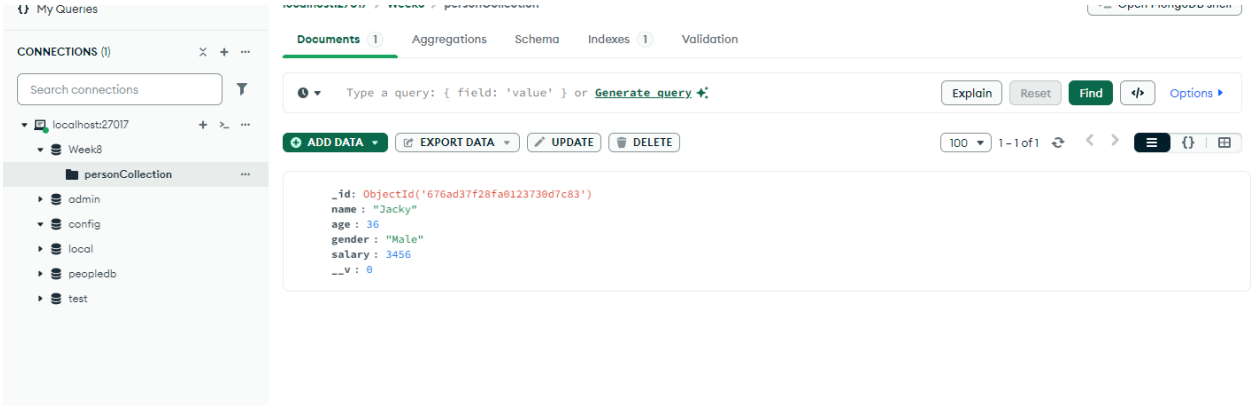
Task 1

In this task, we connected to a MongoDB database using Mongoose and created a schema named PersonSchema for a collection called personCollection. We then created a single document (doc1) and inserted it into the database using the .save() method. The .then() block was used to confirm the document was added successfully, and .catch() handled any errors during insertion.

Key Commands:

- .save(): Saves a single document to the collection.
- .then(): Used to handle the successful resolution of the promise.
- .catch(): Captures any errors that occur during the operation.

```
const mongoose = require('mongoose');
const MONGO_URI = 'mongodb://localhost:27017/Week8';

mongoose.connect(MONGO_URI, { useUnifiedTopology: true, useNewUrlParser: true });

const db = mongoose.connection;
db.on('error', (err) => console.log("Error occurred during connection: " + err));
db.once('connected', () => console.log(`Connected to ${MONGO_URI}`));


//Task 1

const PersonSchema = new mongoose.Schema({
    name: { type: String, required: true },
    age: Number,
    gender: String,
    salary: Number
});

const personModel = mongoose.model('Person', PersonSchema, 'personCollection');
const doc1 = new personModel({ name: 'Jacky', age: 36, gender: 'Male', salary: 3456 });

doc1.save()
    .then((doc1) => console.log("New document added:", doc1))
    .catch((err) => console.error(err));
```



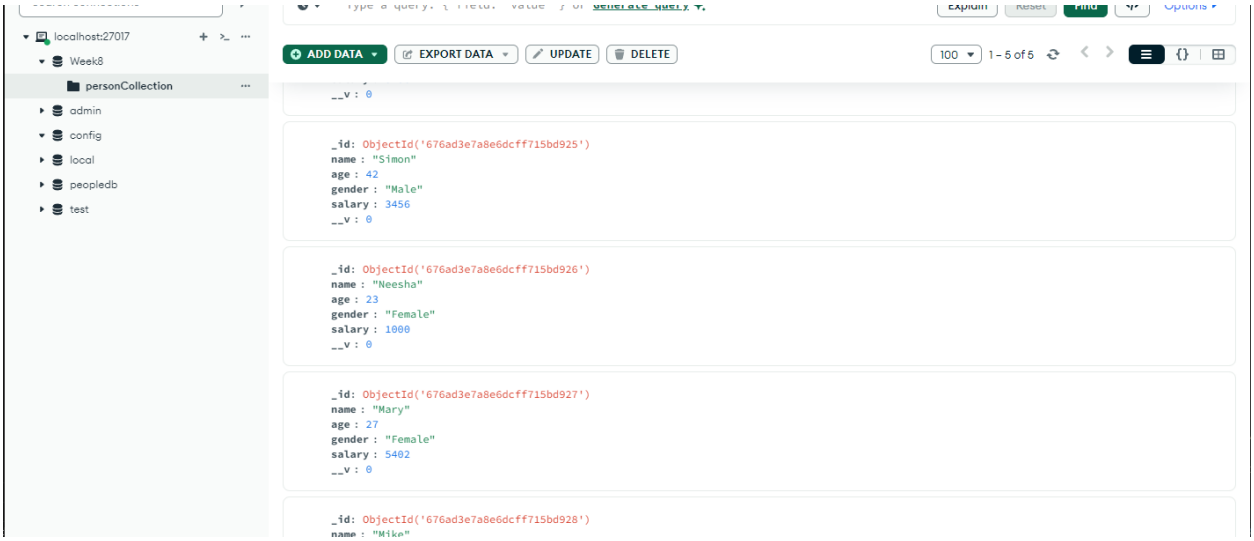
Task 2

We demonstrated how to insert multiple documents at once using the insertMany() method. An array of objects (manypersons) was passed, and the operation was completed within a .then() block for success and a .catch() block for error handling.

Key Commands:

- .insertMany(): Allows insertion of multiple documents in a single operation.

```
27
28  //Task 2
29
30  const manyPersons = [
31      { name: 'Simon', age: 42, gender: 'Male', salary: 3456 },
32      { name: 'Neesha', age: 23, gender: 'Female', salary: 1000 },
33      { name: 'Mary', age: 27, gender: 'Female', salary: 5402 },
34      { name: 'Mike', age: 40, gender: 'Male', salary: 4519 }
35  ];
36
37  personModel.insertMany(manyPersons)
38      .then(() => console.log("Data inserted"))
39      .catch((error) => console.log(error));
40
```



Task 3

We used the find() method to retrieve all documents from the collection. This demonstrated fetching records without any filtering criteria and limiting the result set to a specified number (e.g., 5). This operation allowed us to confirm the stored data.

Key Commands:

- .find(): Fetches documents from the collection.
- .limit(n): Limits the number of documents returned.

```
41  //Task 3
42
43  personModel.find().limit(5)
44      .then((docs) => console.log("Documents retrieved:", docs))
45      .catch((err) => console.error(err));
46
```

Output:

```
Documents retrieved: [
  {
    _id: new ObjectId('676ad37f28fa0123730d7c83'),
    name: 'Jacky',
    age: 36,
    gender: 'Male',
    salary: 3456,
    __v: 0
  },
  {
    _id: new ObjectId('676ad3e7a8e6dcff715bd925'),
    name: 'Simon',
    age: 42,
    gender: 'Male',
    salary: 3456,
    __v: 0
  },
  {
    _id: new ObjectId('676ad3e7a8e6dcff715bd926'),
    name: 'Neesha',
    age: 23,
    gender: 'Female',
    salary: 1000,
    __v: 0
  },
  {
```

We utilized the countDocuments() method to count the total number of records in the collection. This provided a quick way to confirm the size of the data stored in the collection.

Key Commands:

- .countDocuments(): Counts the total documents in the collection.

```
46
47
48   personModel.find({ gender: 'Female', age: { $gt: 25 } })
49       .then((docs) => console.log("Filtered documents:", docs))
50       .catch((err) => console.error(err));
51
```

Output

```
Connected to mongodb://localhost:27017/Week8
Filtered documents: [
  {
    _id: new ObjectId('676ad3e7a8e6dcff715bd927'),
    name: 'Mary',
    age: 27,
    gender: 'Female',
    salary: 5402,
    __v: 0
  }
]
```

Task 4

The find() method was used again, but this time with a filtering condition. We queried documents with Gender = Female and an age greater than a specific value. This task showcased how to retrieve specific subsets of data based on criteria.

Key Commands:

- .find({ criteria }): Fetches documents matching the criteria.

```
53   //Task 4
54
55   personModel.countDocuments()
56       .then((count) => console.log("Total documents count:", count))
57       .catch((err) => console.error(err));
58
```

Output

```
Connected to mongodb://localhost:27017/Week8
Total documents count: 5
```

Task 5

In this task, we demonstrated how to remove records matching specific criteria. The deleteMany() method was used to delete all records where the age was greater than 25. The number of deleted records was printed in the console for verification.

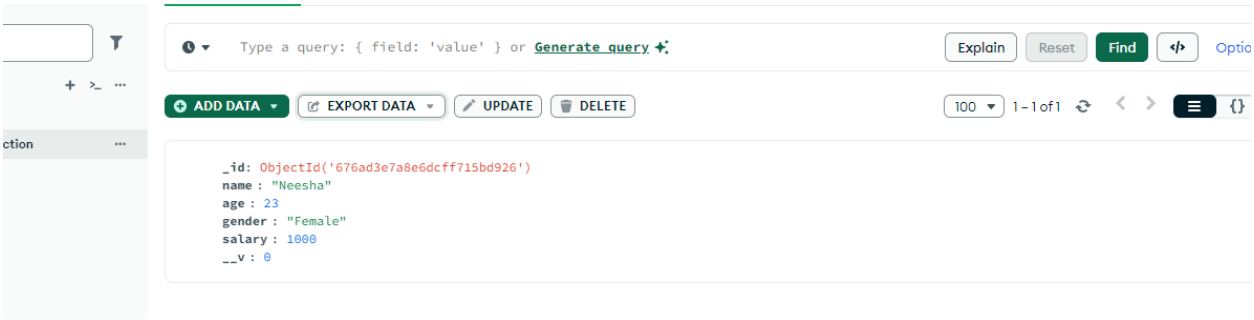
Key Commands:

- .deleteMany({ criteria }): Deletes all documents that match the criteria.

```
59
60  //Task 5
61
62  personModel.deleteMany({ age: { $gte: 25 } })
63      .then((docs) => console.log('Deleted documents:', docs))
64      .catch((error) => console.log(error));
65
```

Output

```
Connected to mongodb://localhost:27017/Week8
Deleted documents: { acknowledged: true, deletedCount: 4 }
```



Task 6

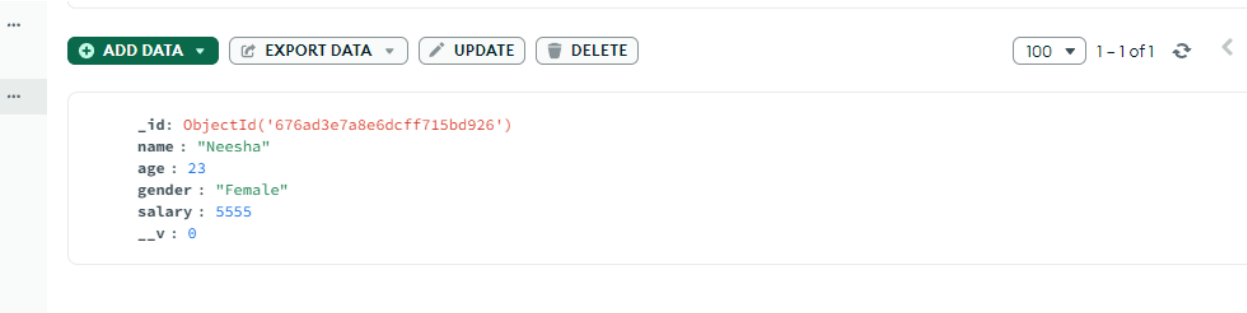
We performed a bulk update on the collection using the updateMany() method. All records where Gender = Female had their Salary field updated to 5555. The operation confirmed the number of updated records in the console.

Key Commands:

- .updateMany({ criteria }, { update }): Updates all documents matching the criteria.

```
66
67  // Task 6
68
69  personModel.updateMany({ gender: 'Female' }, { $set: { salary: 5555 } })
70      .then((docs) => console.log("Updated documents:", docs))
71      .catch((error) => console.log(error));
72
```

Output




```
Connected to mongodb://localhost:27017/Week8
Updated documents: {
  acknowledged: true,
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 1
}
```