Project Title: Understanding Data Relationships in MongoDB

Objective:

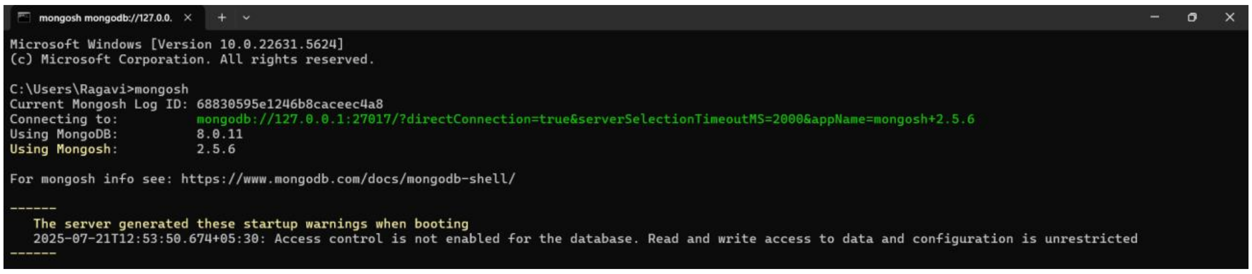
- ➪ This project demonstrates how **1-Many**, **Many-Many**, and **Many-1** relationships can be modeled and queried in **MongoDB** using the mongosh shell.
- ➪ MongoDB is a NoSQL database that doesn't use joins like SQL, so we model relationships using **embedding** (nesting documents) or **referencing** (using ObjectIds).

Table of Contents:

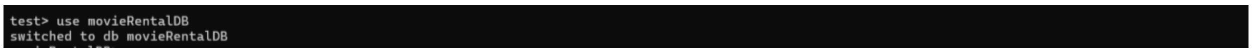
Tools Used:

- MongoDB
- mongosh (MongoDB shell)

1. Project Setup



```
Microsoft Windows [Version 10.0.22631.5624]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Ragavi>mongosh
Current Mongosh Log ID: 68830595e1246b8caceec4a8
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.6
Using MongoDB:          8.0.11
Using Mongosh:          2.5.6

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-07-21T12:53:50.674+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------
```



```
test> use movieRentalDB
switched to db movieRentalDB
```

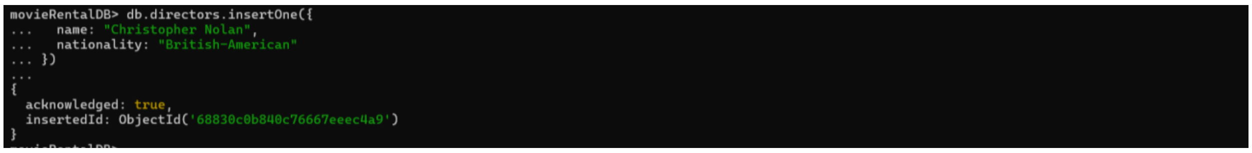
2. One-to-Many (1-Many) Relationship

Relationship: **One Director → Many Movies**

Concept:

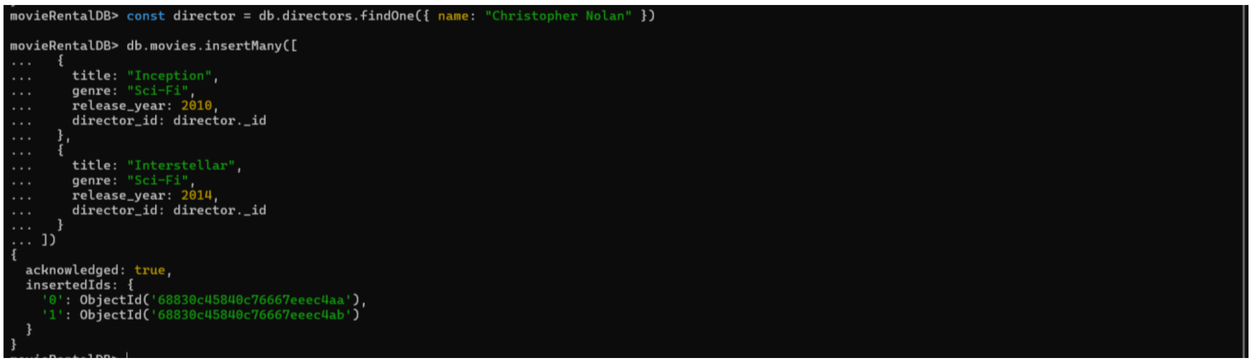
A single director can direct multiple movies, but each movie is directed by one director.

Step 2.1: Insert a Director



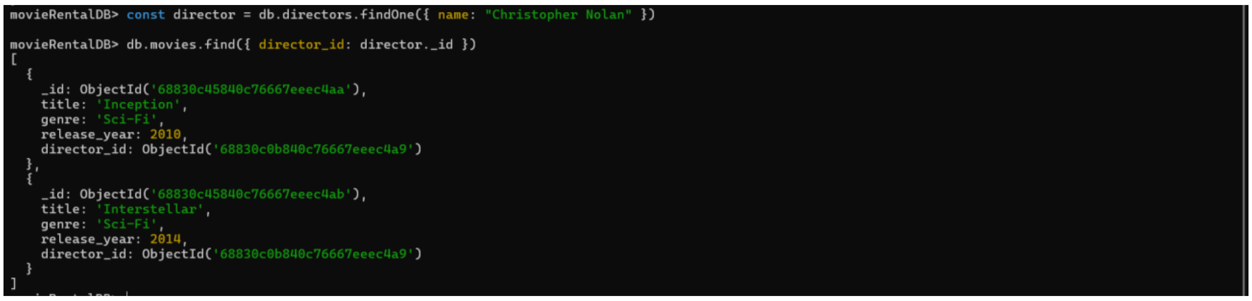
```
movieRentalDB> db.directors.insertOne({
...     name: "Christopher Nolan",
...     nationality: "British-American"
... })
...
{
  acknowledged: true,
  insertedId: ObjectId('68830c0b840c76667eeec4a9')
}
```

Step 2.2: Insert Movies Directed by Nolan



```
movieRentalDB> const director = db.directors.findOne({ name: "Christopher Nolan" })

movieRentalDB> db.movies.insertMany([
...     {
...       title: "Inception",
...       genre: "Sci-Fi",
...       release_year: 2010,
...       director_id: director._id
...     },
...     {
...       title: "Interstellar",
...       genre: "Sci-Fi",
...       release_year: 2014,
...       director_id: director._id
...     }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68830c45840c76667eeec4aa'),
    '1': ObjectId('68830c45840c76667eeec4ab')
  }
}
```

Step 2.3: Query – Find All Movies by Christopher Nolan



```
movieRentalDB> const director = db.directors.findOne({ name: "Christopher Nolan" })

movieRentalDB> db.movies.find({ director_id: director._id })
[
  {
    _id: ObjectId('68830c45840c76667eeec4aa'),
    title: 'Inception',
    genre: 'Sci-Fi',
    release_year: 2010,
    director_id: ObjectId('68830c0b840c76667eeec4a9')
  },
  {
    _id: ObjectId('68830c45840c76667eeec4ab'),
    title: 'Interstellar',
    genre: 'Sci-Fi',
    release_year: 2014,
    director_id: ObjectId('68830c0b840c76667eeec4a9')
  }
]
```

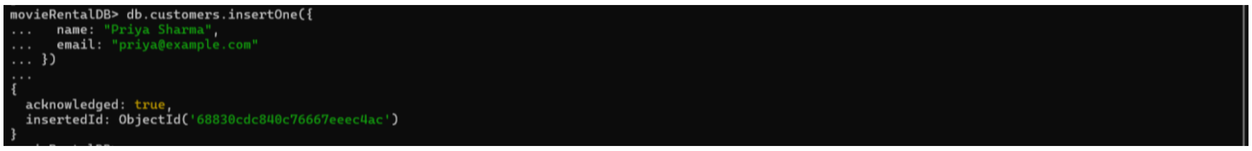
3. Many-to-One Relationship

Relationship: **Many Rentals → One Customer**

Concept:

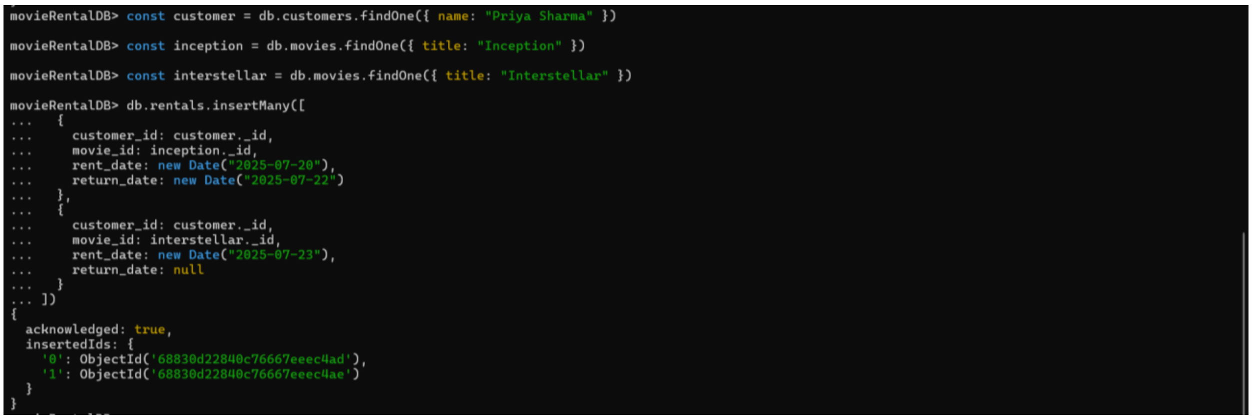
Each rental belongs to one customer, but one customer can have many rentals.

Step 3.1: Insert a Customer



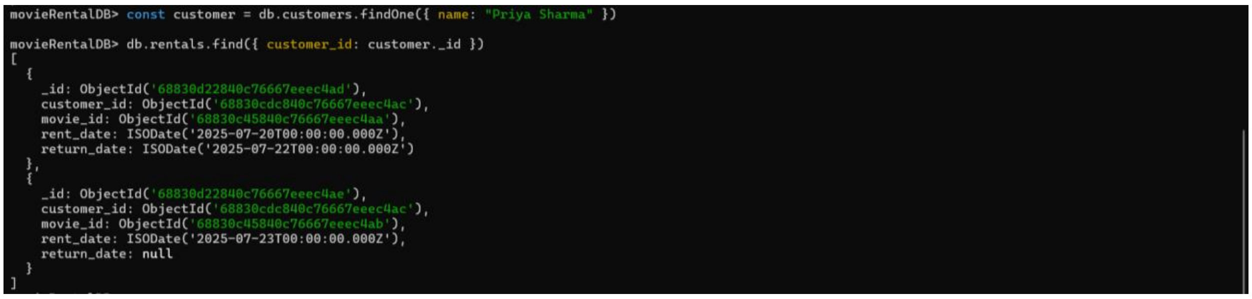
```
movieRentalDB> db.customers.insertOne({
...     name: "Priya Sharma",
...     email: "priya@example.com"
... })
...
{
  acknowledged: true,
  insertedId: ObjectId('68830cdc840c76667eeec4ac')
}
```

Step 3.2: Insert Rental Records



```
movieRentalDB> const customer = db.customers.findOne({ name: "Priya Sharma" })

movieRentalDB> const inception = db.movies.findOne({ title: "Inception" })

movieRentalDB> const interstellar = db.movies.findOne({ title: "Interstellar" })

movieRentalDB> db.rentals.insertMany([
...   {
...     customer_id: customer._id,
...     movie_id: inception._id,
...     rent_date: new Date("2025-07-20"),
...     return_date: new Date("2025-07-22")
...   },
...   {
...     customer_id: customer._id,
...     movie_id: interstellar._id,
...     rent_date: new Date("2025-07-23"),
...     return_date: null
...   }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68830d22840c76667eeec4ad'),
    '1': ObjectId('68830d22840c76667eeec4ae')
  }
}
```

Step 3.3: Query – Find All Rentals by Priya Sharma



```
movieRentalDB> const customer = db.customers.findOne({ name: "Priya Sharma" })

movieRentalDB> db.rentals.find({ customer_id: customer._id })
[
  {
    _id: ObjectId('68830d22840c76667eeec4ad'),
    customer_id: ObjectId('68830cdc840c76667eeec4ac'),
    movie_id: ObjectId('68830c45840c76667eeec4aa'),
    rent_date: ISODate('2025-07-20T00:00:00.000Z'),
    return_date: ISODate('2025-07-22T00:00:00.000Z')
  },
  {
    _id: ObjectId('68830d22840c76667eeec4ae'),
    customer_id: ObjectId('68830cdc840c76667eeec4ac'),
    movie_id: ObjectId('68830c45840c76667eeec4ab'),
    rent_date: ISODate('2025-07-23T00:00:00.000Z'),
    return_date: null
  }
]
```

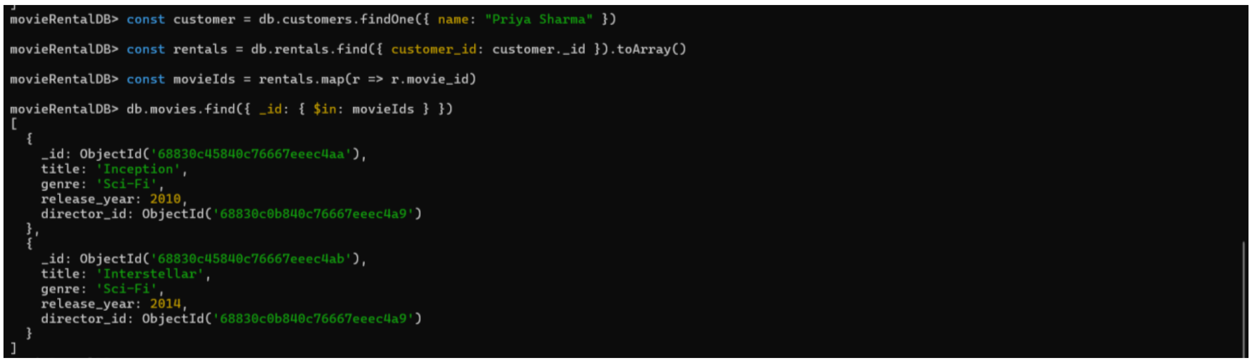
4. Many-to-Many Relationship

Relationship: **Customers ↔ Movies (via Rentals)**

Concept:

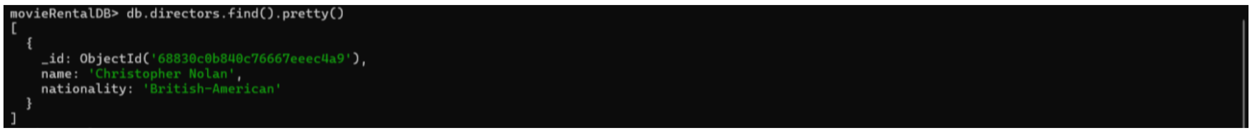
- A customer can rent many movies.
- A movie can be rented by many customers.
- This is modeled using a **rentals** mapping collection.

Step 4.1: Query – Find All Movies Rented by Priya Sharma

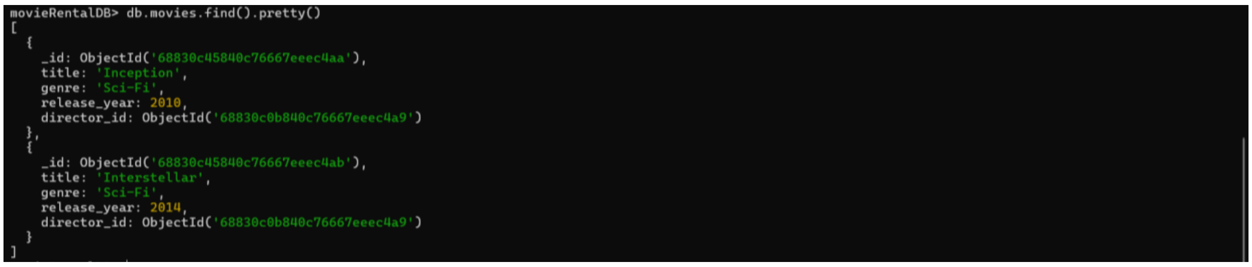


```
movieRentalDB> const customer = db.customers.findOne({ name: "Priya Sharma" })

movieRentalDB> const rentals = db.rentals.find({ customer_id: customer._id }).toArray()

movieRentalDB> const movieIds = rentals.map(r => r.movie_id)

movieRentalDB> db.movies.find({ _id: { $in: movieIds } })
[
  {
    _id: ObjectId('68830c45840c76667eeec4aa'),
    title: 'Inception',
    genre: 'Sci-Fi',
    release_year: 2010,
    director_id: ObjectId('68830c0b840c76667eeec4a9')
  },
  {
    _id: ObjectId('68830c45840c76667eeec4ab'),
    title: 'Interstellar',
    genre: 'Sci-Fi',
    release_year: 2014,
    director_id: ObjectId('68830c0b840c76667eeec4a9')
  }
]
```

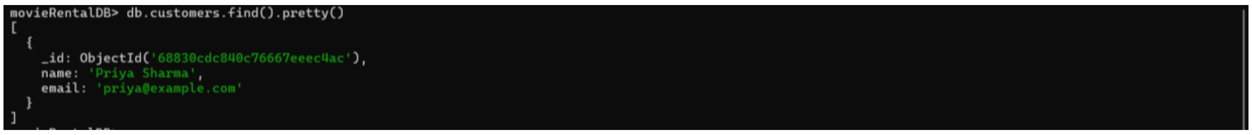
5. Check Data in Each Collection



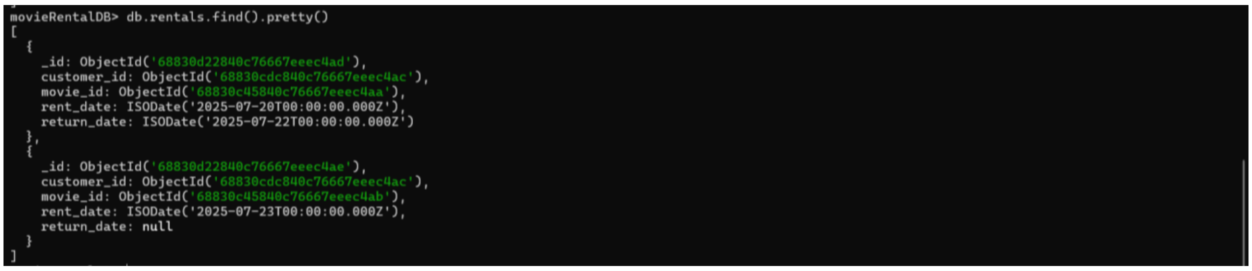
```
movieRentalDB> db.directors.find().pretty()
[
  {
    _id: ObjectId('68830c0b840c76667eeec4a9'),
    name: 'Christopher Nolan',
    nationality: 'British-American'
  }
]
```



```
movieRentalDB> db.movies.find().pretty()
[
  {
    _id: ObjectId('68830c45840c76667eeec4aa'),
    title: 'Inception',
    genre: 'Sci-Fi',
    release_year: 2010,
    director_id: ObjectId('68830c0b840c76667eeec4a9')
  },
  {
    _id: ObjectId('68830c45840c76667eeec4ab'),
    title: 'Interstellar',
    genre: 'Sci-Fi',
    release_year: 2014,
    director_id: ObjectId('68830c0b840c76667eeec4a9')
  }
]
```



```
movieRentalDB> db.customers.find().pretty()
[
  {
    _id: ObjectId('68830cdc840c76667eeec4ac'),
    name: 'Priya Sharma',
    email: 'priya@example.com'
  }
]
```



```
movieRentalDB> db.rentals.find().pretty()
[
  {
    _id: ObjectId('68830d22840c76667eeec4ad'),
    customer_id: ObjectId('68830cdc840c76667eeec4ac'),
    movie_id: ObjectId('68830c45840c76667eeec4aa'),
    rent_date: ISODate('2025-07-20T00:00:00.000Z'),
    return_date: ISODate('2025-07-22T00:00:00.000Z')
  },
  {
    _id: ObjectId('68830d22840c76667eeec4ae'),
    customer_id: ObjectId('68830cdc840c76667eeec4ac'),
    movie_id: ObjectId('68830c45840c76667eeec4ab'),
    rent_date: ISODate('2025-07-23T00:00:00.000Z'),
    return_date: null
  }
]
```

Conclusion:

⇨ Modeled a **One-to-Many** relationship:

One director → Many movies using director_id reference.

⇨ Created a **Many-to-One** relationship:

Many rentals → One customer using customer_id reference.

⇨ Demonstrated a **Many-to-Many** relationship:

Customers ↔ Movies through a rentals collection as a mapping table.

Collections Summary:

- directors → Stores director info
- movies → Stores movie info with director_id
- customers → Stores customer info
- rentals → Connects customers and movies with dates