

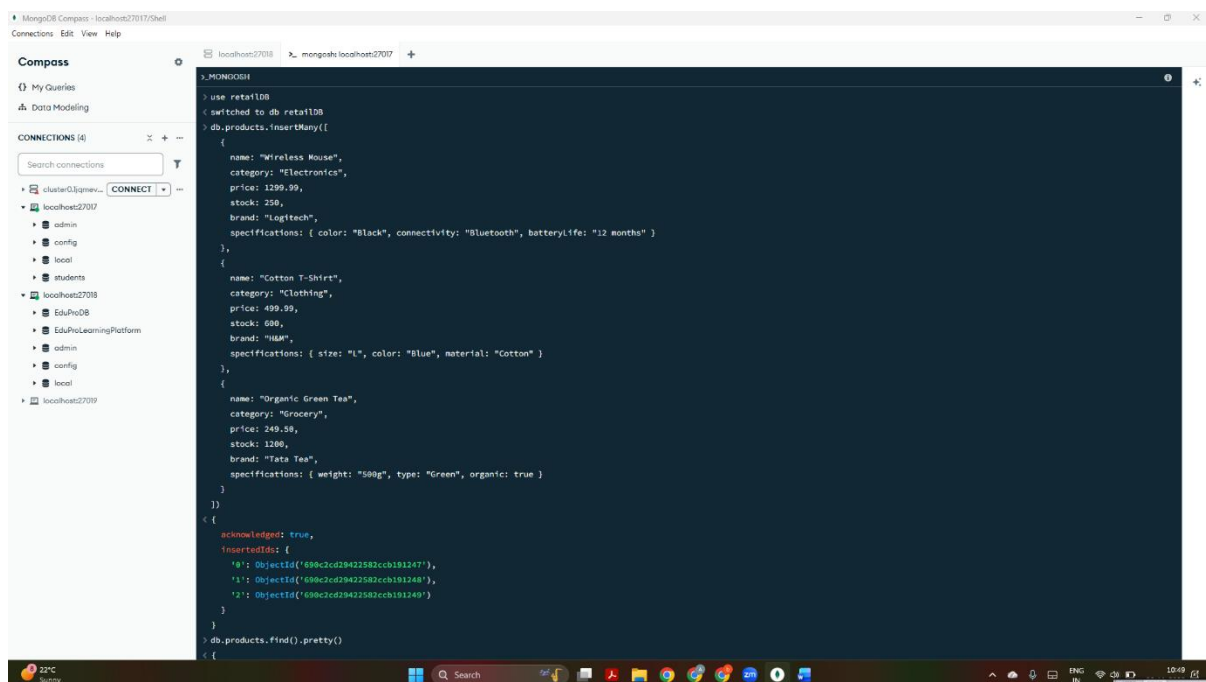
Day 7 Practice question

Project Description

This project focuses on migrating an online retail company's existing product catalog system from a relational database (MySQL) to MongoDB to address scalability and performance challenges. The company's platform manages millions of products with diverse attributes, along with customer orders, payments, and user accounts.

The MongoDB-based solution is designed to efficiently store and manage product information, customer orders, and user authentication details using a flexible document model. Proper indexing strategies are implemented to optimize query performance for common operations such as retrieving products by category and finding orders for specific users. The project also includes sample datasets, schema designs, and performance comparisons before and after indexing to demonstrate improved efficiency.

Output:

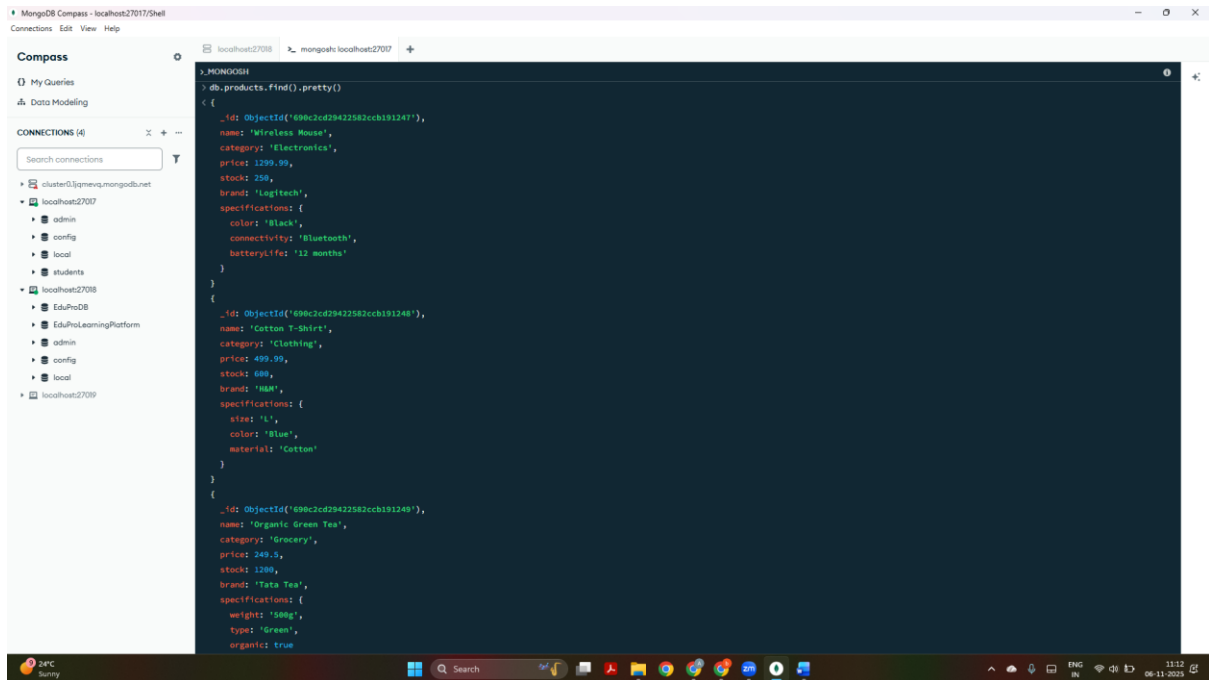


The screenshot shows the MongoDB Compass interface. On the left, the 'CONNECTIONS' panel lists several connections, including 'localhost:27018'. The main window displays the MongoDB shell output for a series of commands. The commands executed are: `use retailDB`, `switched to db retailDB`, `db.products.insertMany([`, followed by three product documents, `])`, `{ acknowledged: true, insertedIds: { '0': ObjectId('6092cd942382ccb191d17'), '1': ObjectId('6092cd942382ccb191d18'), '2': ObjectId('6092cd942382ccb191d19') } }`, and `db.products.find().pretty()`. The output shows the details of the three inserted products: a 'Wireless Mouse', a 'Cotton T-Shirt', and 'Organic Green Tea'.

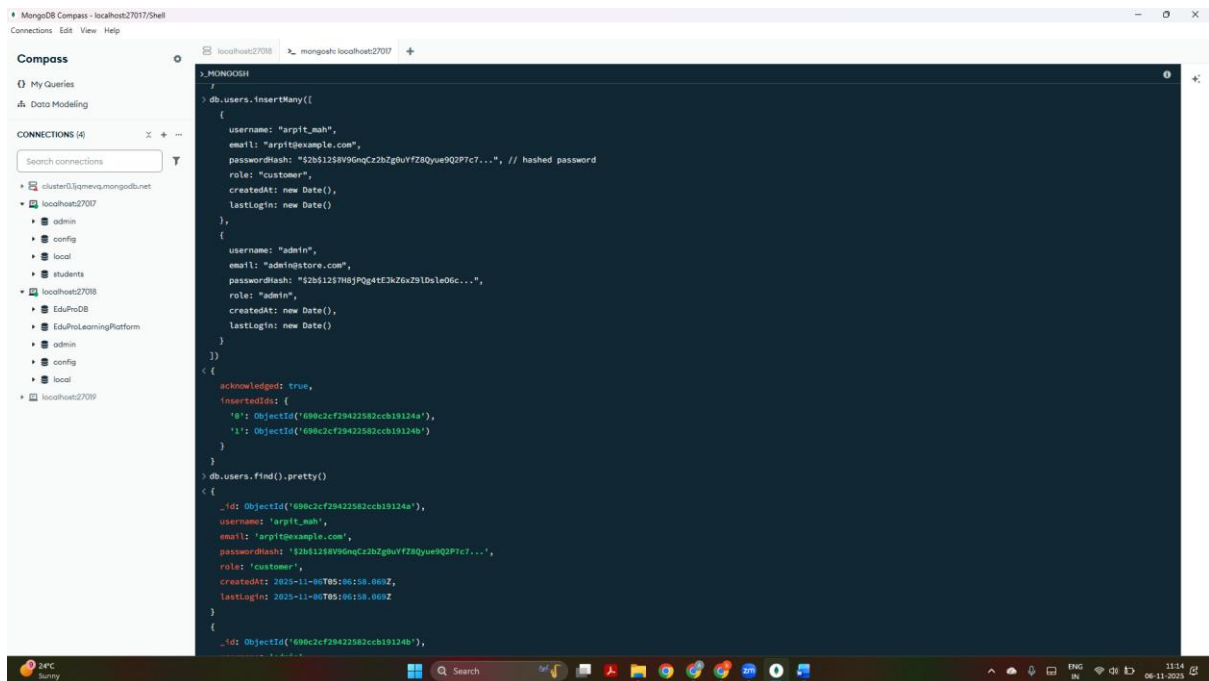
```
> use retailDB
switched to db retailDB
> db.products.insertMany([
  {
    name: "Wireless Mouse",
    category: "Electronics",
    price: 1299.99,
    stock: 250,
    brand: "Logitech",
    specifications: { color: "Black", connectivity: "Bluetooth", batteryLife: "12 months" }
  },
  {
    name: "Cotton T-Shirt",
    category: "Clothing",
    price: 499.99,
    stock: 600,
    brand: "H&M",
    specifications: { size: "L", color: "Blue", material: "Cotton" }
  },
  {
    name: "Organic Green Tea",
    category: "Grocery",
    price: 249.99,
    stock: 1200,
    brand: "Tata Tea",
    specifications: { weight: "500g", type: "Green", organic: true }
  }
])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6092cd942382ccb191d17'),
    '1': ObjectId('6092cd942382ccb191d18'),
    '2': ObjectId('6092cd942382ccb191d19')
  }
}
> db.products.find().pretty()
[
  {
    name: 'Wireless Mouse',
    category: 'Electronics',
    price: 1299.99,
    stock: 250,
    brand: 'Logitech',
    specifications: { color: 'Black', connectivity: 'Bluetooth', batteryLife: '12 months' }
  },
  {
    name: 'Cotton T-Shirt',
    category: 'Clothing',
    price: 499.99,
    stock: 600,
    brand: 'H&M',
    specifications: { size: 'L', color: 'Blue', material: 'Cotton' }
  },
  {
    name: 'Organic Green Tea',
    category: 'Grocery',
    price: 249.99,
    stock: 1200,
    brand: 'Tata Tea',
    specifications: { weight: '500g', type: 'Green', organic: true }
  }
]
```

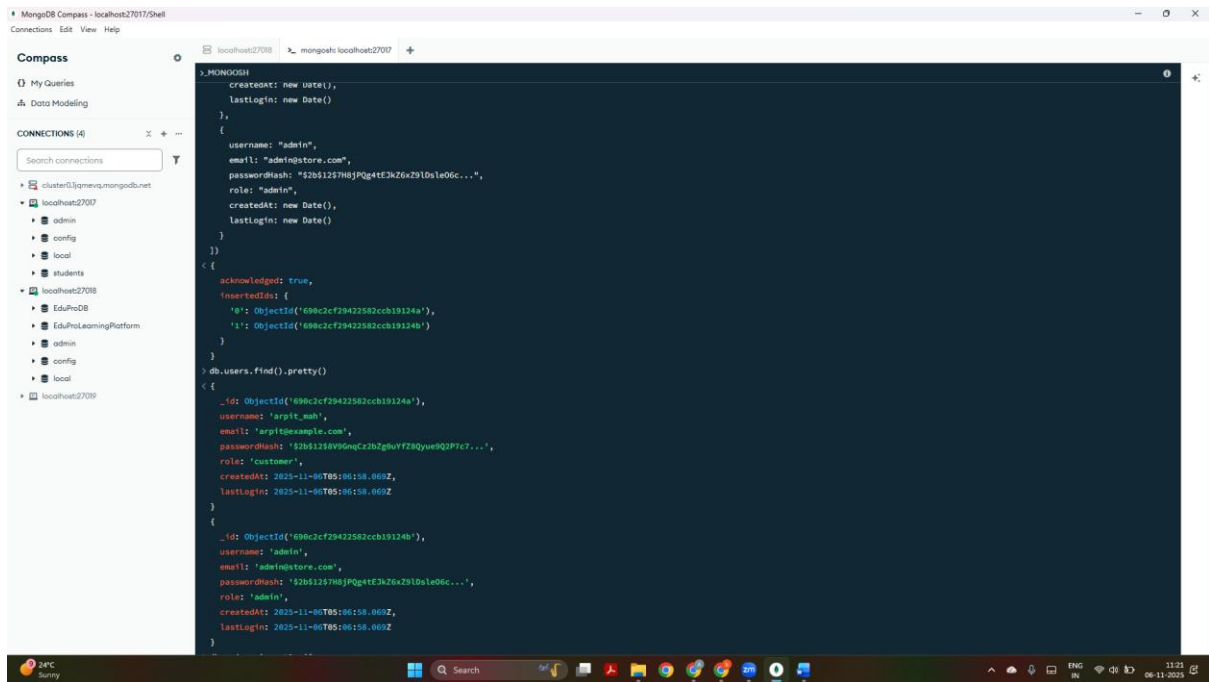
This displays a database **retailDB** is created with collections such as “products”, “users” and “orders” are created and data is inserted inside “products collection. Here, each product document contains details like:

- Basic info (name, category, price)
- Inventory info (stock)
- Nested attributes (brand, specifications)

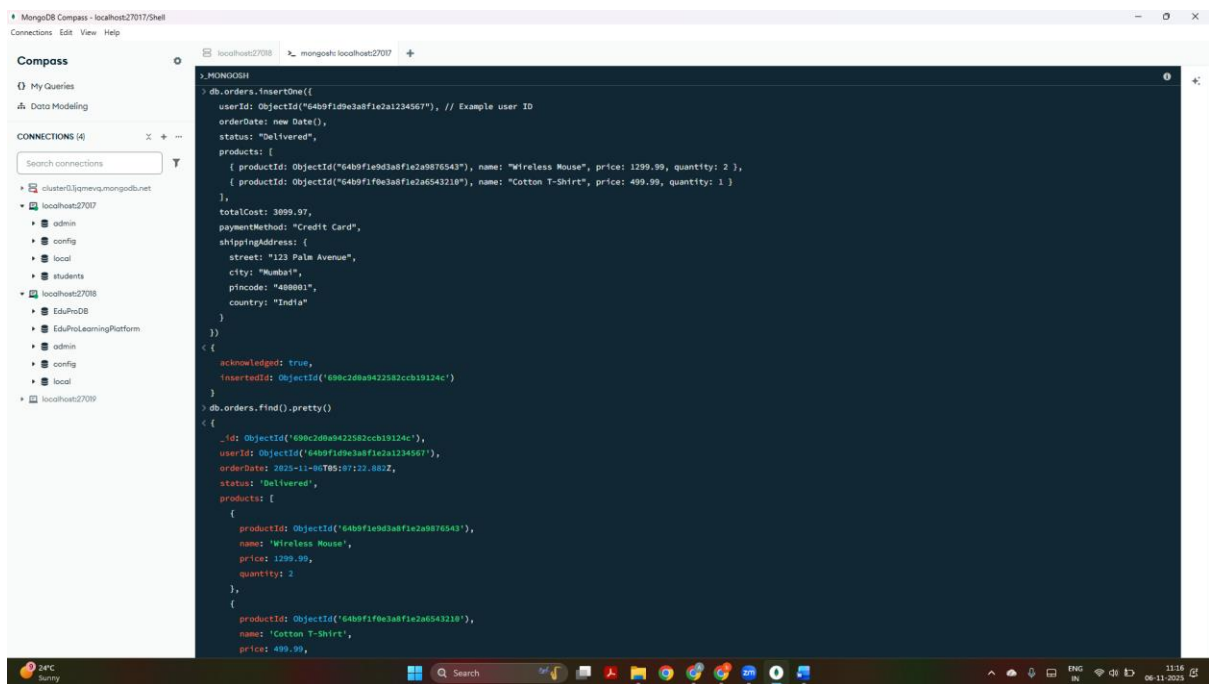


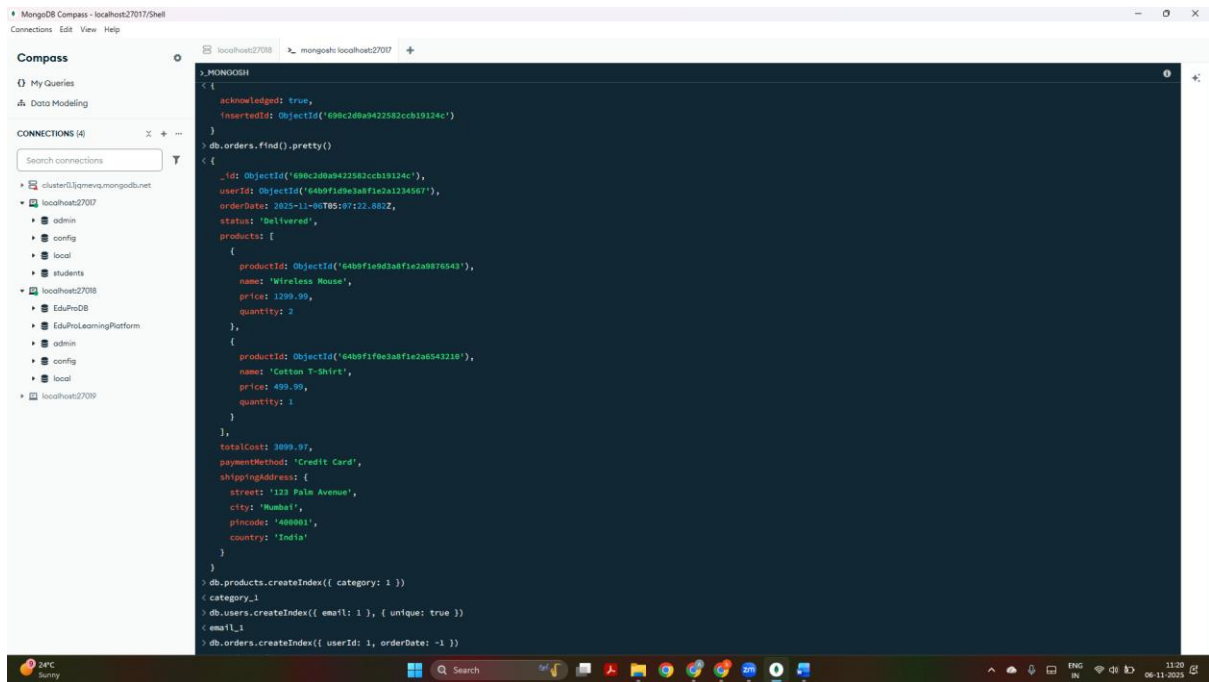
The data is seen in “**products**” collection using-
db.products.find().pretty() query.





The above 2 screenshots display data is inserted inside “users” collection and it is fetched as seen in the output.





The above 2 screenshots displays that data is inserted in “**orders**” collection that:

- References the user via **userId**
- Embeds purchased products (product snapshot)
- Records total cost, payment method, and shipping details

```
db.products.createIndex({ category: 1 })
category_1
db.users.createIndex({ email: 1 }, { unique: true })
email_1
db.orders.createIndex({ userId: 1, orderDate: -1 })
userId_1_orderDate_-1
db.products.getIndexes()
db.users.getIndexes()
db.orders.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  {
    v: 2,
    key: { userId: 1, orderDate: -1 },
    name: 'userId_1_orderDate_-1'
  }
]
```

1) **db.products.createIndex({ category: 1 })**

This command creates an **index** on the category field of the products collection.
The 1 indicates an **ascending order index**.

Purpose:

It helps MongoDB quickly find products that belong to a particular category (for example, “Electronics”) without scanning the entire collection.
This improves the speed of queries like:

db.products.find({ category: "Electronics" })

2) **db.users.createIndex({ email: 1 }, { unique: true })**

This creates an index on the email field in the “users” collection.
The { **unique: true** } option ensures that no two users can have the same email address.

Purpose:

- Makes user lookups by email very fast (for example, during login).
- Prevents duplicate user accounts with the same email.

3) **db.orders.createIndex({ userId: 1, orderDate: -1 })**

This creates a compound index on two fields — userId (ascending) and orderDate (descending).

Purpose:

It speeds up queries that need to:

- Find all orders for a specific user, and
- Sort them by the most recent order date.

For example:

db.orders.find({ userId: ObjectId("...") }).sort({ orderDate: -1 })

4) **db.products.getIndexes(), db.users.getIndexes(), and db.orders.getIndexes()**

These commands list all the indexes currently present in each collection.

Every collection automatically has one default index on `_id`.
Then we see:

{ key: { _id: 1 }, name: '_id_' }

5) This default index ensures each document’s `_id` value is unique and can be found quickly.

When you run `getIndexes()` after creating your own indexes, you see an output like:

```
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { userId: 1, orderDate: -1 }, name: 'userId_1_orderDate_-1' }
]
```

Explanation of the output fields:

- **v: 2** → The internal version number of the index format used by MongoDB. You can ignore it; it's just metadata.
- **key** → Shows which fields the index is built on, and in which order.
For example, `{ userId: 1, orderDate: -1 }` means:
 - The index sorts `userId` in ascending order.
 - Within each user, it sorts `orderDate` in descending order.
- **name** → The system-generated name for the index.
MongoDB automatically names indexes by combining field names and sort orders (like `userId_1_orderDate_-1`).
You can assign your own name if you want when creating the index.