Week 3 Day 15 Minutes of session

Day 15	Mongo DB	Introduction to NoSQL DB A non relational database that stores and retrieves data in a flexible, scalable way(Key-Value. Document, column or graph format) Best suited for unstructured and semi structured data.
		Introduction to MongoDB A popular open source No SQL Document dB that stores Data in JSON-like format.
		MongoDB CRUD Operations
		MongoDB Data Modeling,Indexing and Query Optimization
		Introduction to MongoDB Atlas
		Aggregation Framework in MongoDB

Case study based on 3NF normalization :-

ECommerce DB : Here we have a sales record table tracking customer purchase history.

Unormalized Table

Order ID	Customer Name	Customer Phone	Product Name	Product Price	Category Name
101	Amit	123456	Laptop	60,000	Electronics
102	Arshdeep	3216544	Sports Shoes	25,00	Clothing
103	Akhil	789658	Wireless Keyboard	1,000	Electronics

1. **Redundancy in customer and category info**
2. **Hard to update data consistently**

First Normal Form(1NF) : removing multivalued fields.
Second Normal Form(2NF): All non key attributes should fully depend on the primary key.
Order Table - > Order ID (P.K)
Customer table -> Customer ID(P.K) with Customer Name and Customer Phone Number
Product Table - > Product ID (P.K) with Product Name, Product Price and Category ID

Third Normal Form :(3NF) Moving category details into separate tables for removing transitive dependency.
Category Table we will CategoryID(P.K) with category Name
Final table looks like

orders:[OrderID, CustomerID, ProductID]
Customers : customerID, customerName,Customerphone

Products: [ProductID,ProductName,ProductPrice,CategoryID]
Categories: [CategoryID,Category Name]

Benefits of 3NF:

1. **No duplication of customer or category info.**
2. **Easier to update or delete without inconsistency.**
3. **Better data integrity and reduced storage.**

Case study based on HR Domain

EMPID, EMP ENAME,EmpPhone, DEpart D, DeptLocation and manager Name

1NF: No structural changes , make sure all values are atomic .

2NF: for removing partial dependencies we have to break table

Emp Table [EMpID, EMp name, EMP Phone no and Dept ID]

Department Table [Dept ID, Department, Department Location , Manager ID]

3NF: removing Transitive Dependencies - we have to move manager into its own table

Manager Table [manager ID, manager Name]

EmpID	EmpName	Role	Dept	DeptLocation	ManagerName	ManagerRole	ManagerDept
201	Alice Brown	HR Executive	HR	Mumbai	Priya Mehta	HR Manager	HR
202	Bob Smith	IT Analyst	IT	Pune	Karan Kapoor	IT Head	IT
203	John Davis	HR Executive	HR	Mumbai	Priya Mehta	HR Manager	HR
204	Riya Shah	HR Manager	HR	Mumbai	Arjun Sharma	HR Director	HR
205	Priya Mehta	HR Manager	HR	Mumbai	Arjun Sharma	HR Director	HR

Here are some hidden issues:-

-Role Overlap:

-Redundant Manager info

-Poor referential integrity

Implement 2NF and 3NF

What if an emp is also a manager ?

-Don't duplicate, use a common person ID concept or self referencing

What if two departments share the same manager ?

-Model many to one or use juntino table ManagerDeptRelation

What if the department location changes ?

- We have to implement a department table.

In 1NF : We have to remove repeating Groups and ensure values are atomic.

In 2NF : remove partial dependencies. Beak table on role/dept info.

Employee table : [Emp ID(P.K), Emp Name, Role ID, Dept ID, manager ID(Self ref F.K)]

Department Table : [Dept ID, Dept Name , Deptlocation]

Role table : [Role ID , Role name]

In 3NF : Remove transitive dependency: manager info should be abstracted into its its own entity

Manager Table : [Manager ID, Manager Name, Role ID, Dept ID]

limitation of RDBMS:

limitation of RDBMS	Benefits of NoSQL DB
Rigid Schema	Flexible schema - Schema less design allow dynamic fields and evolving data models
Poor horizontal Scalability	High scalability– built for horizontal scaling via sharding across multiple servers
Complex joins impact performance	Denormalized & Embedded Documents - reduces the need for joins
Limited for unstructured Data	Supports unstructured/semistructured data via storing XML, JSON and logs etc
Hard to handle Big Data Volumes	Designed for Big Data - capable of handling massive datasets distributed across nodes efficiently.
Slow development cycle due to strict schema	Agile Development_ No SQL adapts easily for fast changing requirements,
Difficult to achieve High availability	Built in replication- supports replica sets and failover
Limited Performance in Distributed Setups	Optimized for Distributed setups - Naive support for clustering and geographic replication.
Expensive scaling with licensing	OPen Source and cost effective scaling - No SQL option often free or lower total cost.

Step 1: Downloading Mongo DB

Step 2: Installing MOngo DB

Step 3: Setting Mongo DB Environment variables

Step 4: Crate a Data directory

Step 5: Start MongoDB Server using ‘mongod’ - this will start mongoDB daemon-

Note: In Mongo DB Mongo DB daemon is the core database process. It runs the mongo DB server, handles database operations and manages data storage, replication and requests from clients.

Step 6: Opening mongo Shell and inserting collection

```
Use testDB
db.testCollection.insertOne({ name: “TOM Cruise”, Age: 55 })
db.testCollection.find()
```

MongoDB COmpass/Atlas

Case Study: E-Commerce Inventory & Order Management System (MongoDB-Based)

Problem Statement

You are a backend engineer working for **ShopEx**, an emerging e-commerce platform aiming to transition from a relational database to **MongoDB** to support faster product updates, better scalability, and real-time analytics.

Your task is to design, implement, and optimize a NoSQL-based solution for the platform that efficiently handles **product catalog, customer data, orders, and transactions** using **MongoDB**.

User Role	User Story
Customer	As a customer, I should be able to register, update my profile, and place orders for products.
Admin	As an admin, I should be able to add/update/remove product details from the catalog.
Order Manager	As an order manager, I want to query order status by customer and date, and see average order value.
Analyst	As an analyst, I want to view product sales aggregated by category and by region.

Steps for approaching solution :
Step 1: Create or switch to Database using use **ShopExDB**
Step 2: Create a collection with some data

Create a product collection	db.products.insertMany([
	  {
	  productName: "Wireless Mouse",
	  category: "Electronics",
	  price: 899,
	  stock: 120,
	  rating: 4.5
	  },
	  {
	  productName: "Cotton T-Shirt",
	  category: "Fashion",
	  price: 499,
	  stock: 60,
	  rating: 4.2
	  }
	])

Create a customer collection	db.customers.insertOne({
	  name: "Ravi Kumar",
	  email: "ravi@example.com",
	  city: "Mumbai",
	  registeredOn: new Date("2022-01-01")
	})

Create an order collection	db.orders.insertOne({
	  customerId: ObjectId("<<CustomerID>>"),
	  orderDate: new Date("2023-08-01"),
	  items: [
	  { productId: ObjectId("<<ProductID>>"), quantity: 2, price: 899 }
	  ],
	  totalAmount: 1798,
	  shippingCity: "Mumbai",
	  status: "Delivered"
	})

Step 3: Performing CRUD operations

Updating Stock After Order	db.products.updateOne({ _id: ObjectId("<<ProductID>>") }, { $inc: { stock: -2 } })
Read orders	db.orders.find({ customerId: ObjectId("<<CustomerID>>") })
Delete inactive customer	db.customers.deleteMany({ registeredOn: { $lt: new Date("2021-01-01") } })

Step 4: INdexing and Query optimisation

Create an index on product category	db.products.createIndex({category : 1})
Query with explain Plan	sb.products.find({caegory: “Electronics”}).explain(“executionStats”)

Step 5: Aggregation queries

Average Order Value by City	db.orders.aggregate([{ $group: { _id: "$shippingCity", avgOrder: { $avg: "$totalAmount" } } }])
Top 3 Products by quantity sold	db.orders.aggregate([{ $unwind: "$items" }, { $group: { _id: "$items.productId", totalSold: { $sum: "$items.quantity" } } }, { $sort: { totalSold: -1 } }, { $limit: 3 }])

Step 6: Exporting report in JSON format

Mongoexport –collection = products –db=ShopExDB – out = products.json

Case Study: HR Employee Management System Using MongoDB

📌 Problem Statement

Design and implement an HR Employee Management System using MongoDB for a mid-sized organization. The system should allow efficient management of employee records, departments, attendance, and performance using MongoDB features like CRUD operations, data modeling, indexing, aggregation, and deployment on MongoDB Atlas.

User Stories

As an HR Manager, I want to:

1. **Perform CRUD Operations**
    - Add new employees with their details (name, age, department, salary, city).
    - Update employee details when they get promoted or transferred.
    - Delete records of employees who leave the organization.
    - View employee records filtered by department, city, or salary range.
2. **Model Data Efficiently**
    - Design embedded documents to represent department and performance history inside employee documents.
    - Use referencing for maintaining separate attendance collections.
3. **Implement Indexing & Query Optimization**
    - Create indexes on fields like department, city, and salary to improve search efficiency.
    - Use .explain("executionStats") to verify query performance.

4. **Use Aggregation Framework**
    - Calculate the average salary of employees by department.
    - Count the number of employees in each city.
    - Find top 3 highest-paid employees per department.
5. **Use MongoDB Atlas for Cloud-Based Deployment**
    - Host the database on MongoDB Atlas.
    - Import dummy data to Atlas using Compass or CLI.
    - Provide access to the technical lead for read/write operations.

Dummy Data for Practice

employees collection

```
[
  { "name": "Alice", "age": 29, "department": "HR", "salary": 52000, "city": "Bangalore" },
  { "name": "Bob", "age": 35, "department": "IT", "salary": 75000, "city": "Mumbai" },
  { "name": "Carol", "age": 28, "department": "IT", "salary": 67000, "city": "Mumbai" },
  { "name": "Dave", "age": 40, "department": "Admin", "salary": 45000, "city": "Delhi" },
  { "name": "Eve", "age": 31, "department": "HR", "salary": 50000, "city": "Bangalore" }
]
```

attendance collection

```
[
  { "empName": "Alice", "date": "2025-08-01", "status": "Present" },
  { "empName": "Bob", "date": "2025-08-01", "status": "Absent" },
  { "empName": "Carol", "date": "2025-08-01", "status": "Present" }
]
```

Assessment Questions (For Review)

1. How can you embed department details inside the employee document?
2. Demonstrate how to fetch the top 2 highest paid employees from IT department.
3. How would you model attendance as a separate referenced collection?
4. What indexes will you create to optimize filter-based queries on city and salary?
5. How can MongoDB Atlas help in secure deployment of the above HR system?