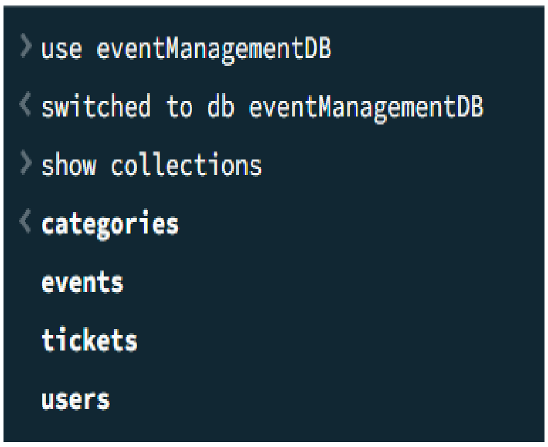
# Assignment 1 :

Event Management and Ticketing system

Submitted by: Komal Joshi



```
> use eventManagementDB
< switched to db eventManagementDB
> show collections
< categories
  events
  tickets
  users
```

This assignment focuses on designing a MongoDB-based Event Management and ticketing System.It includes schema design for managing events, ticketing, and user interactions such as browsing, booking and managing tickets.

The MongoDB database named eventManagementDB consists of four collections: Users, Events, Tickets, and categories.CRUD operations are performed on these collections, and aggregation pipelines are used to generate reports.

## MongoDB collections and schema definitions

1. **Users collection (users)**

```
{
  userId: String,
  name: String,
  email: String,
  phone: String,
  role: "attendee" or "organizer",
  createdAt: Date
}
```

2. **Events collection (events)**

```
{
  eventId: String,
  title: String,
  description: String,
  category: String,
  dateTime: Date,
  venue: String,
  organizerId: String,
  price: Number,
  totalTickets: Number,
  availableTickets: Number,
  status: "upcoming" or "cancelled" or "completed"
}
```

3. **Tickets collection (tickets)**

```
{
  ticketId: String,
  eventId: String,
  userId: String,
  bookingDate: Date,
  quantity: Number,
  totalAmount: Number,
  status: "booked" or "cancelled"
}
```

4. **Categories collection (categories)**

```
{
  categoryId: String,
  name: String,
  description: String
}
```

# Sample Data

1. **Users**

```
db.users.insertMany([
 {
  "userId": "U01",
  "name": "Aarav Sharma",
  "email": "aarav@gmail.com",
  "phone": "9000000001",
  "role": "organizer",
  "createdAt": "2025-01-01"
},
{
  "userId": "U06",
  "name": "Simran Kaur",
  "email": "simran@gmail.com",
  "phone": "9000000006",
  "role": "attendee",
  "createdAt": "2025-01-06"
}
])
```

Similarly other users' documents were added.

2. **Events**

```
db.events.insertMany([
{
  "eventId": "E01",
  "title": "Tech Innovators Summit",
  "description": "Conference on emerging technologies",
  "category": "Technology",
  "dateTime": "2026-02-10T10:00:00",
  "venue": "Noida Expo Centre",
  "organizerId": "U01",
  "price": 1500,
  "totalTickets": 300,
  "availableTickets": 180,
  "status": "upcoming"
}
])
```

Similarly,20 distinct events are created with unique eventId.

3. **Tickets**

```
db.tickets.insertMany([
{
  "ticketId": "T01",
  "eventId": "E01",
  "userId": "U06",
  "bookingDate": "2026-01-05",
  "quantity": 2,
  "totalAmount": 3000,
  "status": "booked"
}
])
```

4. **Categories**

```
db.categories.insertMany([
{
  "categoryId": "C01",
  "name": "Music",
  "description": "Concerts and live music events"
}
])
```

Similarly 20 different categories are inserted.

# CRUD Operations

These operations allow creation, retrieval, updating, and deletion of data in collections.

- createCollection() to create the collections
- insertMany() to enter 20 documents per collection
- find(),updateOne(),deleteOne() for further modifications.

**CRUD operations performed:**

**db.users.find().limit(1).pretty();**

```
{
 _id: ObjectId('696c8238122cc461ed1872c5'),
 userId: 'U01',
 name: 'Aarav Sharma',
 email: 'aarav@gmail.com',
 phone: '9000000001',
 role: 'organizer',
 createdAt: '2025-01-01'
}
```

**db.tickets.find({totalAmount : { $gt : 3000}}).pretty();**

```
{
  _id: ObjectId('696ca064122cc461ed1872ef'),
  ticketId: 'T03',
  eventId: 'E02',
  userId: 'U08',
  bookingDate: '2026-01-01',
  quantity: 3,
  totalAmount: 3600,
  status: 'booked'
}
```

**db.categories.find({},{"_id" : 0}).limit(2).skip(3).pretty()**

```
{
  categoryId: 'C04',
  name: 'Arts',
  description: 'Art exhibitions and creative workshops'
}
{
  categoryId: 'C05',
  name: 'Education',
  description: 'Educational seminars'
}
```

**db.events.updateOne({ eventId : "E03" }, { $set : { venue : "Mumbai theatre" }})**

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

# Aggregation pipelines

### 1.Top 5 events by ticket sales:

Since ticket collection has eventId as well as the quantity of ticket sold we will use ticket collection for pipeline. Firstly we will filter only the event ticket whose status is booked(removing cancelled one). Then group them with eventId and find the total ticket Quantity per event using sum operator and finally sorting in descending order and limiting to top 5.

```
db.tickets.aggregate([
  { $match : { status : "booked" } },
  { $group : {
      _id : "$eventId",
      totalTicketQuantity : { $sum : "$quantity" }
  }},
  { $sort : { totalTicketQuantity : -1 } },
  { $limit : 5 }
]);
```

### 2.Total revenue earned by an organizer:

In this we have total revenue as total amount in ticket collection and organizerId and details in event collection hence after filtering the booked tickets we used lookup operator to join the two collections.unwind is used to break the array.

```
db.tickets.aggregate([
  { $match : { status : "booked" } },
  { $lookup : {
      from : "events",
      localField : "eventId",
      foreignField : "eventId",
      as : "eventDetails"
  }},
  { $unwind : "$eventDetails" },
  { $group : {
      _id : "$eventDetails.organizerId",
      totalRevenueOfOrganizer : { $sum : "$totalAmount" }
  }},
  { $sort : { _id: 1 } }
]);
```

### 3. Number of attendees per event:

Again, we used ticket collections and used eventId as id for grouping as we have to find number of attendees per event.then sum of the quantity for each event.

```
db.tickets.aggregate([
  { $match : { status : "booked" } },
  { $group : {
      _id : "$eventId",
      attendees : { $sum : "$quantity" }
  }},
  { $sort : { _id : 1 } }
]);
```

### 4. Events grouped by category and status

In this we used two fields as id for grouping as asked in Question.new field created counts all the events with some category and status. We used $project to remove the id field for output.(to make it look nicer)

```
db.events.aggregate([
  { $group : {
      _id : { category : "$category", status : "$status" },
      groupedEvent : { $sum : 1 }
  }},
  { $project : {
      _id : 0,
      category : "$_id.category",
      status : "$_id.status",
      groupedEvent : 1
  }}
])
```

## Indexes

Indexes improve performance for frequent event searches by date or category. So we can create indexes:

*db.events.createIndex({ category: 1 })*
*db.events.createIndex({ dateTime: 1 })*

## Ticket booking and cancellation logic

1. **Ticket booking:** we create a new document for the ticket using insertOne() and put the status field as booked. Then we will decrease the number of available tickets as the quantity of tickets booked in the event collection.

```
db.tickets.insertOne({
  ticketId: "T21",
  eventId: "E03",
  userId: "U21",
  bookingDate: new Date(),
  quantity: 2,
  totalAmount: 3000,
  status: "booked"
});

db.events.updateOne(
  { eventId: "E03" },
  { $inc: { availableTickets: -2 } }
);
```

2. **Ticket cancellation:** for cancellation we update the status for ticket to be cancelled in ticket collection and increase the available tickets is event collection.

```
db.tickets.updateOne(
  { ticketId: "T21" },
  { $set: { status: "cancelled" } }
);

db.events.updateOne(
  { eventId: "E03" },
  { $inc: { availableTickets: 2 } }
);
```

“When a ticket is booked, available tickets are reduced. When cancelled, tickets are restored.”

*This project demonstrates the use of MongoDB for building an Event Management and Ticketing System using well-structured collections, CRUD operations, aggregation pipelines, and indexes. The database design efficiently handles event creation, ticket booking and cancellation, and reporting requirements. Through the use of aggregation queries and indexing, the system supports meaningful data analysis and optimized query performance.*