Here’s a database schema for an event management system implemented in Node.js with MongoDB. MongoDB is a NoSQL database, so it uses collections and documents rather than tables and rows. Here's how you could structure it:

**Schema Design**

**Collections**

1. **Users**
2. **Events**
3. **Tickets**
4. **Venues**
5. **Organizers**

**1. Users Collection**

For storing user information.

const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({

name: { type: String, required: true },

email: { type: String, required: true, unique: true },

password: { type: String, required: true },

role: { type: String, enum: ['attendee', 'organizer', 'admin'], default: 'attendee' },

createdAt: { type: Date, default: Date.now }

});

module.exports = mongoose.model('User', UserSchema);

**2. Events Collection**

For managing events.

const mongoose = require('mongoose');

const EventSchema = new mongoose.Schema({

name: { type: String, required: true },

description: { type: String, required: true },

organizerId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

venueId: { type: mongoose.Schema.Types.ObjectId, ref: 'Venue', required: true },

date: { type: Date, required: true },

time: { type: String, required: true },

ticketsAvailable: { type: Number, required: true },

ticketPrice: { type: Number, required: true },

categories: [{ type: String }], // Example: ['Music', 'Tech', 'Sports']

createdAt: { type: Date, default: Date.now }

});

module.exports = mongoose.model('Event', EventSchema);

**3. Tickets Collection**

For ticket purchases and management.

const mongoose = require('mongoose');

const TicketSchema = new mongoose.Schema({

eventId: { type: mongoose.Schema.Types.ObjectId, ref: 'Event', required: true },

userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

purchaseDate: { type: Date, default: Date.now },

quantity: { type: Number, required: true },

totalAmount: { type: Number, required: true }

});

module.exports = mongoose.model('Ticket', TicketSchema);

**4. Venues Collection**

For managing event locations.

const mongoose = require('mongoose');

const VenueSchema = new mongoose.Schema({

name: { type: String, required: true },

location: {

address: { type: String, required: true },

city: { type: String, required: true },

state: { type: String },

country: { type: String, required: true }

},

capacity: { type: Number, required: true },

contact: {

phone: { type: String },

email: { type: String }

},

createdAt: { type: Date, default: Date.now }

});

module.exports = mongoose.model('Venue', VenueSchema);

**5. Organizers Collection**

(Optional if separate from Users)

const mongoose = require('mongoose');

const OrganizerSchema = new mongoose.Schema({

userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

organizationName: { type: String, required: true },

contactInfo: {

phone: { type: String },

email: { type: String, required: true }

},

createdAt: { type: Date, default: Date.now }

});

module.exports = mongoose.model('Organizer', OrganizerSchema);

**Relationships**

1. **Users ↔ Events**:
   * organizerId in Events links to a User with a role of "organizer".
   * userId in Tickets links to Users.
2. **Events ↔ Venues**:
   * venueId in Events references a Venue.
3. **Events ↔ Tickets**:
   * eventId in Tickets references an Event.

**Example Queries**

* **Get all events by a specific organizer:**

Event.find({ organizerId: organizerId }).populate('venueId');

* **Get tickets purchased by a user:**

Ticket.find({ userId: userId }).populate('eventId');

* **Get event details including venue:**

Event.findById(eventId).populate('venueId');

This schema provides flexibility for adding more features later, like reviews, discounts, or notifications. Let me know if you'd like help refining it!