# MongoDB Data Modelling: E-commerce

## What Is Data Modelling?

In MongoDB, **data modeling** is the process of designing the **structure of your documents and collections** to match your app's requirements and performance needs.

## Project Context: E-commerce App – E-commerce Monsta

### 1. Identify Collections

|  |  |
| --- | --- |
| Collection | Purpose |
| users | Customers using the platform |
| sellers | Merchants who list products |
| products | Items available for sale |
| inventory | Quantity and availability per product per seller |
| categories | Product groupings (e.g., Electronics, Fashion) |
| orders | Records of purchases |
| cart | Temporary items selected by a user |
| reviews | Product reviews by users |
| wishlists | Saved products by users for future interest |

### 2. Define Relationships

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | Relationship | Type | Description | Field Reference | Modeling Approach | Justification |
| 1 | **User ↔ Orders** | One-to-Many | A user can place multiple orders. Each order belongs to one user. | orders.userId → users.\_id | 🔗 **Reference** | Orders can grow large; better to separate for scalability. |
| 2 | **User ↔ Reviews** | One-to-Many | A user can write many reviews, each review belongs to one user. | reviews.userId → users.\_id | 🔗 **Reference** | Reviews are queryable across products; keep them separate. |
| 3 | **User ↔ Wishlist (Product)** | Many-to-Many | A user can wishlist many products, a product can appear in many wishlists. | users.wishlist[] → products.\_id | 🔗 **Reference** (store array of product IDs in user) | Efficient for user-centric wishlist display. |
| 4 | **Seller ↔ Products** | One-to-Many | A seller can list many products. Each product is posted by one seller. | products.sellerId → sellers.\_id | 🔗 **Reference** | Seller info is needed occasionally; referencing avoids duplication. |
| 5 | **Product ↔ Inventory** | One-to-One (per seller) | Each product entry has a stock managed in the inventory by seller. | inventory.productId, inventory.sellerId → products.\_id, sellers.\_id | 🔗 **Reference** | Keep inventory in a separate collection for flexible updates. |
| 6 | **Product ↔ Category** | Many-to-One | Each product belongs to one category; each category has many products. | products.categoryId → categories.\_id | 🔗 **Reference** | Categories are static and reusable; better to reference. |
| 7 | **Product ↔ Reviews** | One-to-Many | Each product can have multiple reviews. | reviews.productId → products.\_id | 🔗 **Reference** | Allows independent review management and aggregation. |
| 8 | **Order ↔ Items (Product snapshot)** | One-to-Many (embedded) | Each order includes multiple purchased products. | orders.items[] | ✅ **Embed product snapshot** | Product details at time of purchase are saved in order for accuracy. |
| 9 | **User ↔ Cart Items (Product snapshot)** | One-to-Many (embedded) | User can add multiple items to cart with qty & price. | cart.items[] | ✅ **Embed** | Cart is short-lived; embedding makes retrieval fast. |
| 10 | **Product ↔ Tags** | One-to-Many | A product can have multiple tags (e.g., ["mobile", "apple"]) | products.tags[] | ✅ **Embed** | Tags are small static strings; embedding is efficient. |

### 3. Embed vs Reference

|  |  |
| --- | --- |
| Use Case | Recommended Modeling |
| Order items with product snapshot | Embed |
| Product belongs to a seller | Reference sellerId |
| Product belongs to a category | Reference categoryId |
| Review made by a user | Reference userId |
| Cart items | Embed with product snapshot |

### 4. Sample Schemas

#### a. users

{

"\_id": ObjectId("..."),

"name": "Alice",

"email": "alice@example.com",

"address": {

"city": "Chennai",

"zip": "600001"

},

"wishlist": [ObjectId("product1"), ObjectId("product2")],

"createdAt": ISODate("2025-07-01T10:00:00Z"),

"updatedAt": ISODate("2025-07-08T15:00:00Z")

}

#### b. sellers

{

"\_id": ObjectId("..."),

"name": "TechWorld Inc.",

"email": "sales@techworld.com",

"gst": "33AAAPT1234A1ZV",

"address": {

"city": "Bangalore",

"zip": "560001"

},

"createdAt": ISODate("2025-07-01T10:00:00Z"),

"updatedAt": ISODate("2025-07-08T15:00:00Z")

}

#### c. products

{

"\_id": ObjectId("..."),

"name": "MacBook Air M3",

"price": 120000,

"categoryId": ObjectId("..."),

"sellerId": ObjectId("..."),

"tags": ["laptop", "apple", "electronics"],

"createdAt": ISODate("2025-07-01T10:00:00Z"),

"updatedAt": ISODate("2025-07-08T15:00:00Z")

}

#### d. inventory

{

"\_id": ObjectId("..."),

"productId": ObjectId("..."),

"sellerId": ObjectId("..."),

"quantityAvailable": 50,

"reorderLevel": 10,

"createdAt": ISODate("2025-07-01T10:00:00Z"),

"updatedAt": ISODate("2025-07-08T15:00:00Z")

}

#### e. orders

{

"\_id": ObjectId("..."),

"userId": ObjectId("..."),

"items": [

{

"productId": ObjectId("..."),

"name": "MacBook Air",

"priceAtPurchase": 120000,

"qty": 1

}

],

"total": 120000,

"status": "shipped",

"orderDate": ISODate("2025-07-09T10:00:00Z"),

"createdAt": ISODate("2025-07-09T10:00:00Z"),

"updatedAt": ISODate("2025-07-09T10:10:00Z")

}

#### f. reviews

{

"\_id": ObjectId("..."),

"productId": ObjectId("..."),

"userId": ObjectId("..."),

"rating": 4,

"comment": "Great value",

"createdAt": ISODate("2025-07-08T09:00:00Z"),

"updatedAt": ISODate("2025-07-08T09:00:00Z")

}

### 5. Query Scenarios to Think About

|  |  |
| --- | --- |
| Scenario | Fields to Use |
| Show all products sold by a seller | Filter by sellerId in products |
| Show product inventory | Join products and inventory on productId |
| Show all reviews for a product | Filter reviews by productId |
| Get user’s order history | Filter orders by userId |

### 6. Practice Task

After reviewing these examples, try designing the data model for your project.

**Your task:**

* Design the **collections** with relationships
* Include createdAt and updatedAt in every schema
* Seed sample documents in MongoDB
* Write 5 common queries your app will use