UNDERSTANDING RELATIONSHIPS IN MONGO DB

In MongoDB, a relationship refers to how documents are connected or associated with each other. Just like in relational databases (SQL), where tables are linked using foreign keys, MongoDB allows you to model connections between documents.

Since MongoDB is schema-less and stores data in BSON (Binary JSON) format, relationships are handled differently — either by nesting documents or by referencing them.

Relationships help you:

- **Organize data logically** (e.g., students and their addresses or courses)
- **Avoid duplication** (e.g., storing course details once and referencing them)
- **Improve query efficiency** (e.g., fetching related data in one go)
- **Maintain consistency** (e.g., updating a referenced document updates all linked data)

Types of Relationships in MongoDB

MongoDB supports three main types of relationships:

Relationship Type	Description	MongoDB Implementation
One-to-One	One document relates to one other document	Embedded or Reference
One-to-Many	One document relates to many others	Embedded Array or Reference
Many-to-Many	Many documents relate to many others	Reference + `$lookup`

Each of these can be implemented using either:

- **Embedded Documents**
- **References**
- **$lookup Aggregation** (for joins across collections)

What Is an Embedded Document?

An **embedded document** is a document nested inside another document.

Example:

```
{
  StudentId: "S101",
  Name: "Aarav",
  Address: {
    Street: "12 MG Road",
    City: "Bangalore",
    Pincode: 560001
  }
}
```

Here, `Address` is embedded inside the student document.

Advantages:

- Fast reads (all data in one place)
- Simple queries
- Good for tightly coupled data

Disadvantages:

- Document size can grow quickly
- Harder to update individual subdocuments
- Not ideal for data accessed independently

What Is a Reference Model?

A **reference model** stores related data in separate documents and links them using an identifier (like an `_id`).

Example:

```
// Address document
{
  _id: ObjectId("abc123"),
  Street: "12 MG Road",
  City: "Bangalore",
  Pincode: 560001
}

// Student document
{
  StudentId: "S101",
  Name: "Aarav",
  AddressId: ObjectId("abc123")
}
```

Here, the student document references the address document using `AddressId`.

Advantages:

- Better normalization
- Easier to manage large or independent data
- Avoids bloated documents

Disadvantages:

- Requires multiple queries or `$lookup` to join
- Slightly slower reads

When to Use Embedded vs Reference?

Criteria	Use Embedded	Use Reference
Data is always accessed together	✅ Yes	❌ No
Data grows independently	❌ No	✅ Yes
Document size is small	✅ Yes	❌ No
Need normalization	❌ No	✅ Yes
Frequent updates to subdata	❌ No	✅ Yes

What Is $lookup in MongoDB?

$lookup is an **aggregation stage** in MongoDB that allows you to **join documents from two collections**, similar to a SQL JOIN.

It's part of the **aggregation pipeline and** is typically used when you're working with **referenced data** across collections.

Why Use $lookup?

MongoDB doesn't support joins natively like relational databases. So when you store related data in separate collections (using references), $lookup helps you **combine** that data during queries.

1. One-to-One Relationship

Use Case:

Each customer has one shipping address.

◇ Embedded Document Model

```
db.customers.insertOne({
  CustomerId: "C001",
  Name: "Ritika",
  Email: "ritika@example.com",
  ShippingAddress: {
    Street: "22 Market Lane",
    City: "Pune",
    Pincode: 411001 }})
```

```
> db.customers.insertOne({
    CustomerId: "C001",
    Name: "Ritika",
    Email: "ritika@example.com",
    ShippingAddress: {
      Street: "22 Market Lane",
      City: "Pune",
      Pincode: 411001 }})

< {
    acknowledged: true,
    insertedId: ObjectId('688314b3f677d782b481f4ee')
  }
> db.customers.find(
    { CustomerId: "C001" },
    { Name: 1, ShippingAddress: 1, _id: 0 }
  )
< {
    Name: 'Ritika',
    ShippingAddress: {
      Street: '22 Market Lane',
      City: 'Pune',
      Pincode: 411001
    }
  }
```

- **Reference Model**

```
// Shipping address collection
const addrId = db.addresses.insertOne({
  Street: "22 Market Lane",
  City: "Pune",
  Pincode: 411001
}).insertedId

// Customer collection
db.customers.insertOne({
  CustomerId: "C002",
  Name: "Arjun",
  Email: "arjun@example.com",ShippingAddressId: addrId})
```

```
> db.customers.aggregate([
    {
      $lookup: {
        from: "addresses",
        localField: "AddressId",
        foreignField: "AddressId",
        as: "ShippingAddress"
      }
    },
    { $match: { CustomerId: "C002" } },
    {
      $project: {
        Name: 1,
        ShippingAddress: 1,
        _id: 0
      }
    }
  ])
< {
    Name: 'Arjun',
    ShippingAddress: [
      {
        _id: ObjectId('68831cccf677d782b481f4ef'),
        Street: '22 Market Lane',
        City: 'Pune',
        Pincode: 411001
      }
    ]
  }
```

2. One-to-Many Relationship

Use Case:

One customer places multiple order.

- **Embedded Document Model**

```
db.customers.insertOne({
  CustomerId: "C002",
  Name: "Arjun",
  Orders: [
   { OrderId: "O101", Total: 2500, Status: "Shipped" },
   { OrderId: "O102", Total: 1800, Status: "Processing"}]})
```

```
> db.customers.find(
    { CustomerId: "C002" },
    { Name: 1, Orders: 1, _id: 0 }
  )
< {
    Name: 'Arjun'
  }
  {
    Name: 'Arjun',
    Orders: [
      {
        OrderId: 'O101',
        Total: 2500,
        Status: 'Shipped'
      },
      {
        OrderId: 'O102',
        Total: 1800,
        Status: 'Processing'
      }
    ]
  }
```

◇ Reference Model

```
// Orders collection
db.orders.insertMany([
  { OrderId: "O103", CustomerId: "C001", Total: 2500,
Status: "Shipped" },
  { OrderId: "O104", CustomerId: "C001", Total: 1800,
Status: "Processing" }
])

// Customer document
db.customers.insertOne({
  CustomerId: "C001",
  Name: "Ritika",
  OrderIds: ["O103", "O104"]})
```

```
> // Orders collection
  db.orders.insertMany([
    { OrderId: "O103", CustomerId: "C001", Total: 2500, Status: "Shipped" },
    { OrderId: "O104", CustomerId: "C001", Total: 1800, Status: "Processing" }
  ])

  // Customer document
  db.customers.insertOne({
    CustomerId: "C001",
    Name: "Ritika",
    OrderIds: ["O103", "O104"]})
< {
    acknowledged: true,
    insertedId: ObjectId('68831e4cf677d782b481f4f4')
  }
> db.orders.find(
    { CustomerId: "C001" },
    { OrderId: 1, Total: 1, Status: 1, _id: 0 }
  )
< {
    OrderId: 'O103',
    Total: 2500,
    Status: 'Shipped'
  }
  {
    OrderId: 'O104',
    Total: 1800,
    Status: 'Processing'
  }
```

3. Many-to-Many Relationship

Use Case:

Each order contains multiple products, and each product can appear in multiple orders.

- **Reference Model + `$lookup`**

```
// Products collection
db.products.insertMany([
  { ProductId: "P001", Name: "Laptop", Price: 60000 },
  { ProductId: "P002", Name: "Headphones", Price: 3000 }])

// Orders collection
db.orders.insertOne({
  OrderId: "O105",
  CustomerId: "C003",
  ProductIds: ["P001", "P002"],
  Total: 63000})
```

- **Aggregation with `$lookup`**

```
db.orders.aggregate([
  {
    $lookup: {
      from: "products",
      localField: "ProductIds",
      foreignField: "ProductId",
      as: "ProductDetails"}}]). pretty()
```

```
> // Products collection
  db.products.insertMany([
    { ProductId: "P001", Name: "Laptop", Price: 60000 },
    { ProductId: "P002", Name: "Headphones", Price: 3000 }])

  // Orders collection
  db.orders.insertOne({
    OrderId: "O105",
    CustomerId: "C003",
    ProductIds: ["P001", "P002"],
    Total: 63000})
< {
    acknowledged: true,
    insertedId: ObjectId('68831ea0f677d782b481f4f7')
  }
```

```
> db.products.aggregate([
    {
      $lookup: {
        from: "orders",
        localField: "ProductId",
        foreignField: "ProductIds",
        as: "OrderDetails"
      }
    },
    {
      $project: {
        _id: 0,
        ProductId: 1,
        Name: 1,
        Price: 1,
        OrderDetails: 1
      }
    }
  ])
```

```
< {
    ProductId: 'P001',
    Name: 'Laptop',
    Price: 60000,
    OrderDetails: [
      {
        _id: ObjectId('68831ea0f677d782b481f4f7'),
        OrderId: 'O105',
        CustomerId: 'C003',
        ProductIds: [
          'P001',
          'P002'
        ],
        Total: 63000
      }
    ]
  }
  {
    ProductId: 'P002',
    Name: 'Headphones',
    Price: 3000,
    OrderDetails: [
      {
        _id: ObjectId('68831ea0f677d782b481f4f7'),
        OrderId: 'O105',
        CustomerId: 'C003',
        ProductIds: [
          'P001',
          'P002'
        ],
        Total: 63000
      }
    ]
  }
```

```
> db.orders.aggregate([
    {
      $lookup: {
        from: "products",
        localField: "ProductIds",
        foreignField: "ProductId",
        as: "ProductDetails"
      }
    },
    {
      $project: {
        _id: 0,
        OrderId: 1,
        CustomerId: 1,
        Total: 1,
        ProductDetails: 1
      }
    }
  ])
```

```
< {
    OrderId: 'O103',
    CustomerId: 'C001',
    Total: 2500,
    ProductDetails: []
  }
  {
    OrderId: 'O104',
    CustomerId: 'C001',
    Total: 1800,
    ProductDetails: []
  }
  {
    OrderId: 'O105',
    CustomerId: 'C003',
    Total: 63000,
    ProductDetails: [
      {
        _id: ObjectId('68831ea0f677d782b481f4f5'),
        ProductId: 'P001',
        Name: 'Laptop',
        Price: 60000
      },
      {
        _id: ObjectId('68831ea0f677d782b481f4f6'),
        ProductId: 'P002',
        Name: 'Headphones',
        Price: 3000
      }
    ]
  }
```

SUMMARY

Relationship	Example Entities	Embedded Model Example	Reference Model Example	Use $lookup?
One-to-One	Customer → ShippingAddress	Embed address in customer	Customer → AddressId	Optional
One-to-Many	Customer → Orders	Embed orders in customer	Orders → CustomerId	Optional
Many-to-Many	Orders ↔ Products	Not ideal	Orders → ProductIds	✅ Yes