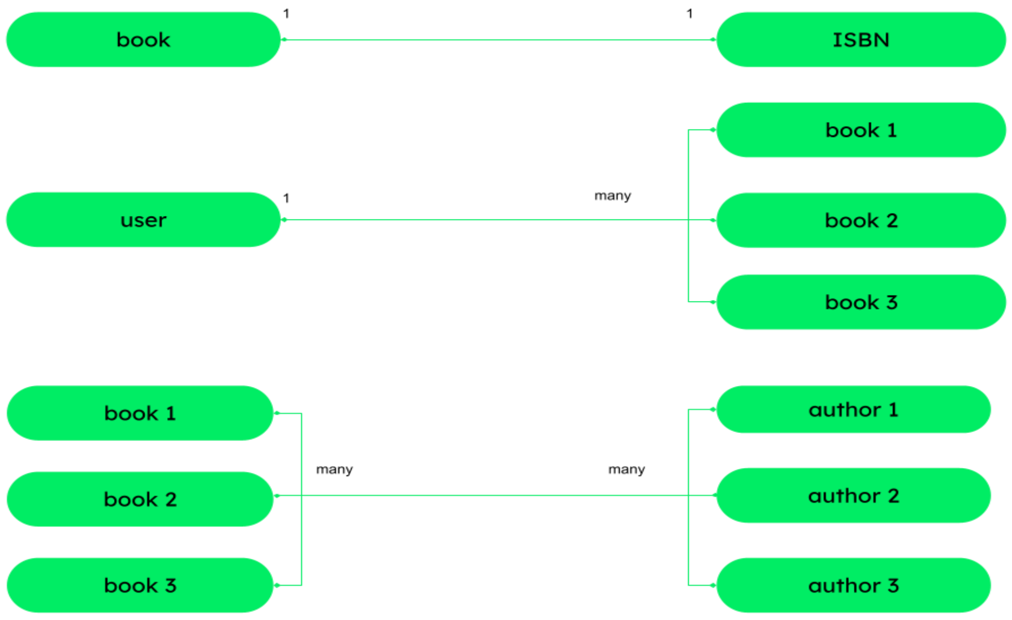
MongoDB Coding Evaluation

Relationship in MongoDB

In **MongoDB**, relationships between data can be managed using **embedded documents** and references:

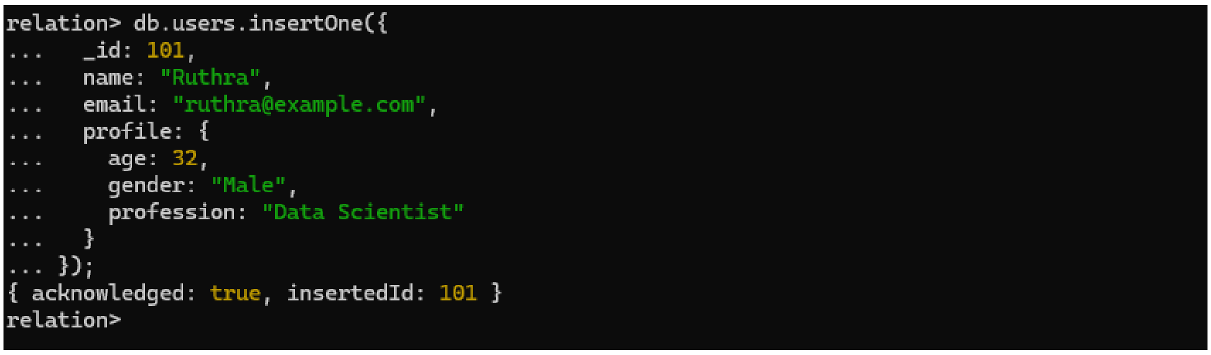
1. **Embedded Documents**: This approach stores related data within a single document, ideal for data that is frequently accessed together. It simplifies data retrieval and ensures **data locality**.
2. **Reference Model**: This method involves storing references to related documents using unique identifiers, suitable for large or independently accessed data. It allows for normalization and maintains **data consistency**.
3. **$lookup**: MongoDB's aggregation framework supports **joins** between collections using the **$lookup** stage, enabling complex **many-to-many** relationships.

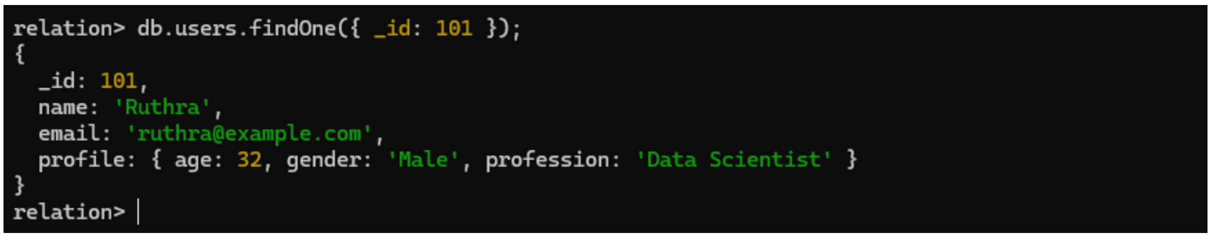


1. One-to-One Relationship

A. Embedded Approach

Example: User with embedded profile

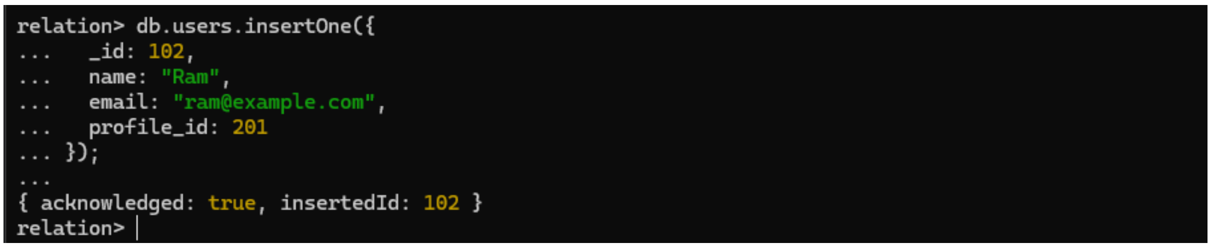

```
relation> db.users.insertOne({
...    _id: 101,
...    name: "Ruthra",
...    email: "ruthra@example.com",
...    profile: {
...      age: 32,
...      gender: "Male",
...      profession: "Data Scientist"
...    }
... });
{ acknowledged: true, insertedId: 101 }
relation>
```



```
relation> db.users.findOne({ _id: 101 });
{
  _id: 101,
  name: 'Ruthra',
  email: 'ruthra@example.com',
  profile: { age: 32, gender: 'Male', profession: 'Data Scientist' }
}
relation> 
```

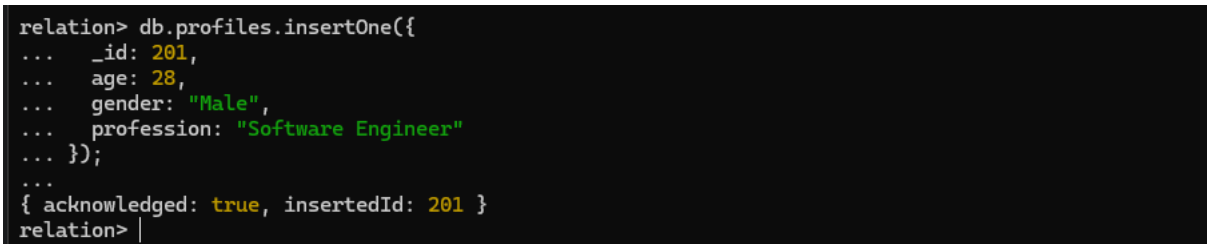
B. Referenced Approach

Users collection



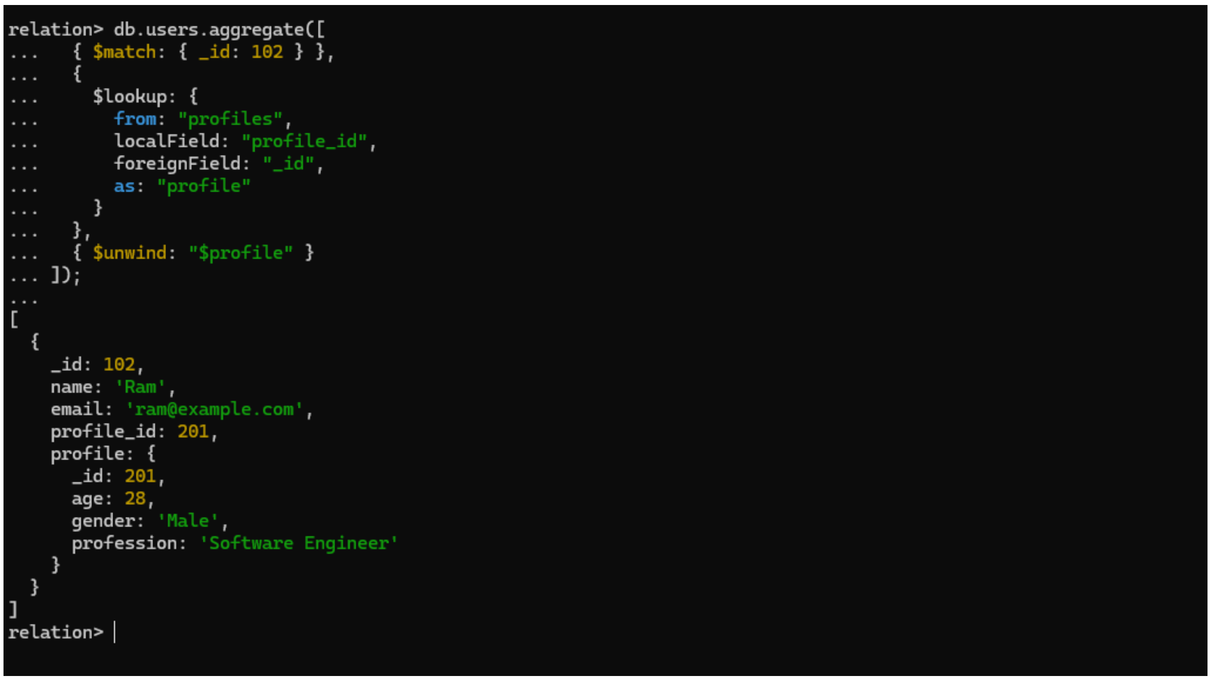
```
relation> db.users.insertOne({
...    _id: 102,
...    name: "Ram",
...    email: "ram@example.com",
...    profile_id: 201
... });
...
{ acknowledged: true, insertedId: 102 }
relation> 
```

Profiles collection



```
relation> db.profiles.insertOne({
...    _id: 201,
...    age: 28,
...    gender: "Male",
...    profession: "Software Engineer"
... });
...
{ acknowledged: true, insertedId: 201 }
relation> 
```

Query using $lookup

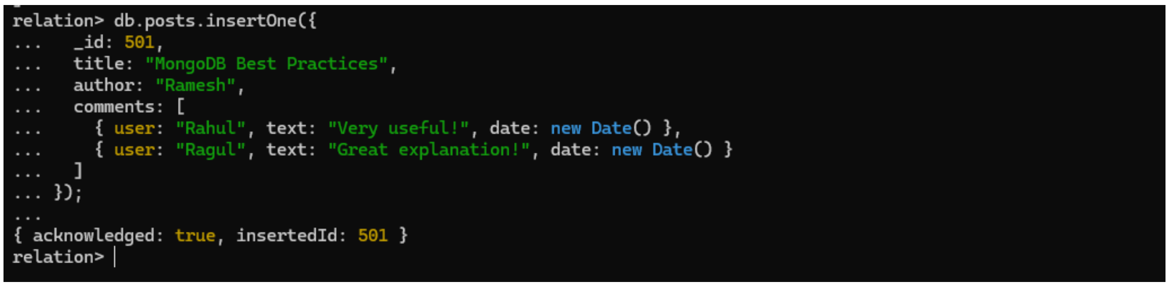


```
relation> db.users.aggregate([
...    { $match: { _id: 102 } },
...    {
...      $lookup: {
...        from: "profiles",
...        localField: "profile_id",
...        foreignField: "_id",
...        as: "profile"
...      }
...    },
...    { $unwind: "$profile" }
... ]);
...
[
  {
    _id: 102,
    name: 'Ram',
    email: 'ram@example.com',
    profile_id: 201,
    profile: {
      _id: 201,
      age: 28,
      gender: 'Male',
      profession: 'Software Engineer'
    }
  }
]
relation> 
```

2. One-to-Many Relationship

A. Embedded Approach

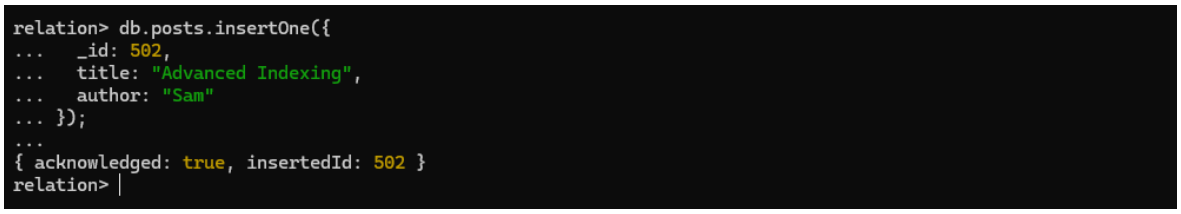
Example: Blog post with embedded comments



```
relation> db.posts.insertOne({
...   _id: 501,
...   title: "MongoDB Best Practices",
...   author: "Ramesh",
...   comments: [
...     { user: "Rahul", text: "Very useful!", date: new Date() },
...     { user: "Ragul", text: "Great explanation!", date: new Date() }
...   ]
... });
...
{ acknowledged: true, insertedId: 501 }
relation>
```

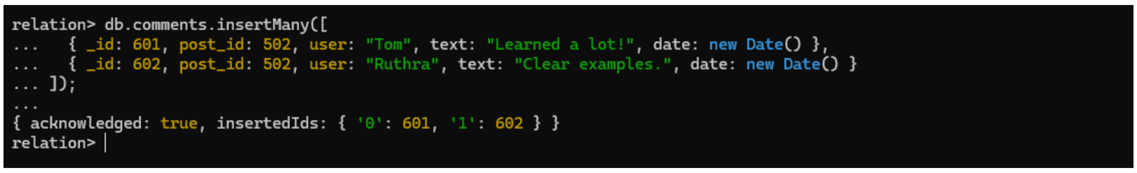
B. Referenced Approach

Posts collection (Author: Sam)



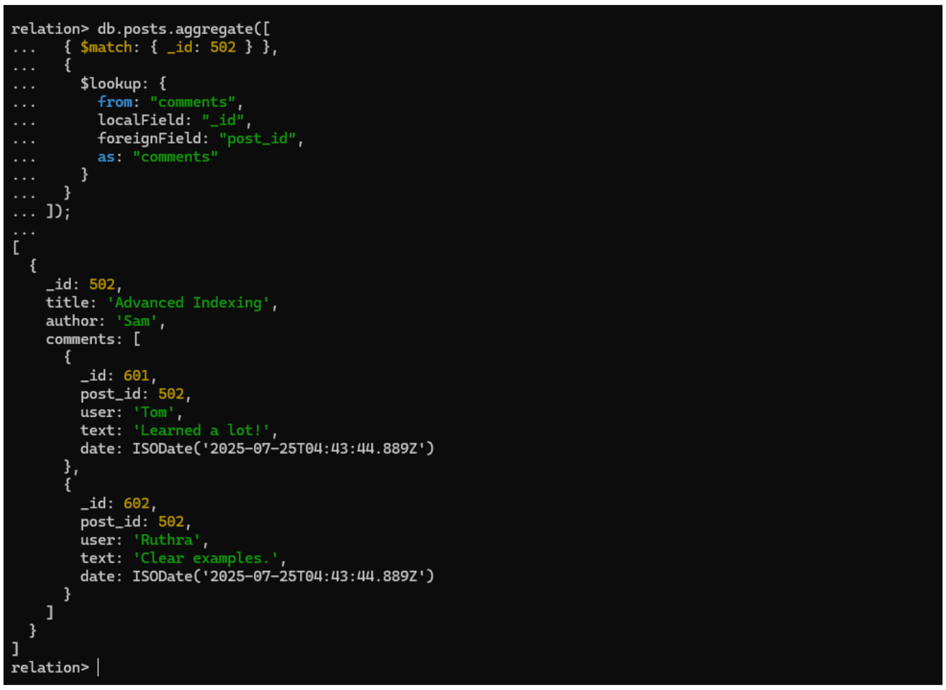
```
relation> db.posts.insertOne({
...   _id: 502,
...   title: "Advanced Indexing",
...   author: "Sam"
... });
...
{ acknowledged: true, insertedId: 502 }
relation>
```

Comments collection



```
relation> db.comments.insertMany([
...   { _id: 601, post_id: 502, user: "Tom", text: "Learned a lot!", date: new Date() },
...   { _id: 602, post_id: 502, user: "Ruthra", text: "Clear examples.", date: new Date() }
... ]);
...
{ acknowledged: true, insertedIds: { '0': 601, '1': 602 } }
relation>
```

Query with $lookup

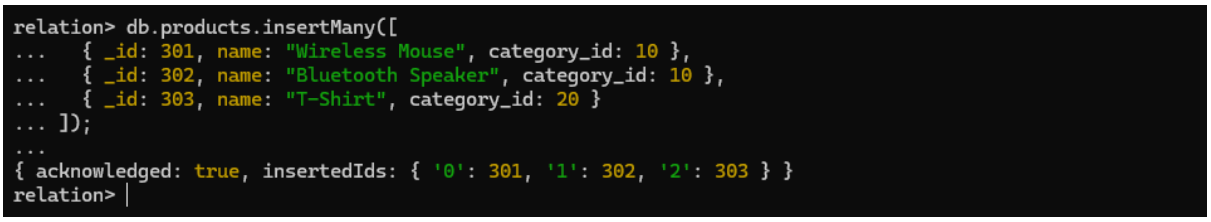


```
relation> db.posts.aggregate([
...   { $match: { _id: 502 } },
...   {
...     $lookup: {
...       from: "comments",
...       localField: "_id",
...       foreignField: "post_id",
...       as: "comments"
...     }
...   }
... ]);
...
[
  {
    _id: 502,
    title: 'Advanced Indexing',
    author: 'Sam',
    comments: [
      {
        _id: 601,
        post_id: 502,
        user: 'Tom',
        text: 'Learned a lot!',
        date: ISODate('2025-07-25T04:43:44.889Z')
      },
      {
        _id: 602,
        post_id: 502,
        user: 'Ruthra',
        text: 'Clear examples.',
        date: ISODate('2025-07-25T04:43:44.889Z')
      }
    ]
  }
]
relation>
```

3. Many-to-One Relationship

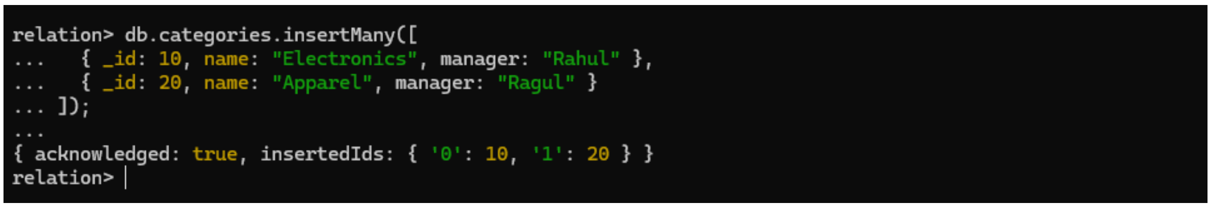
Referenced Approach

Products collection



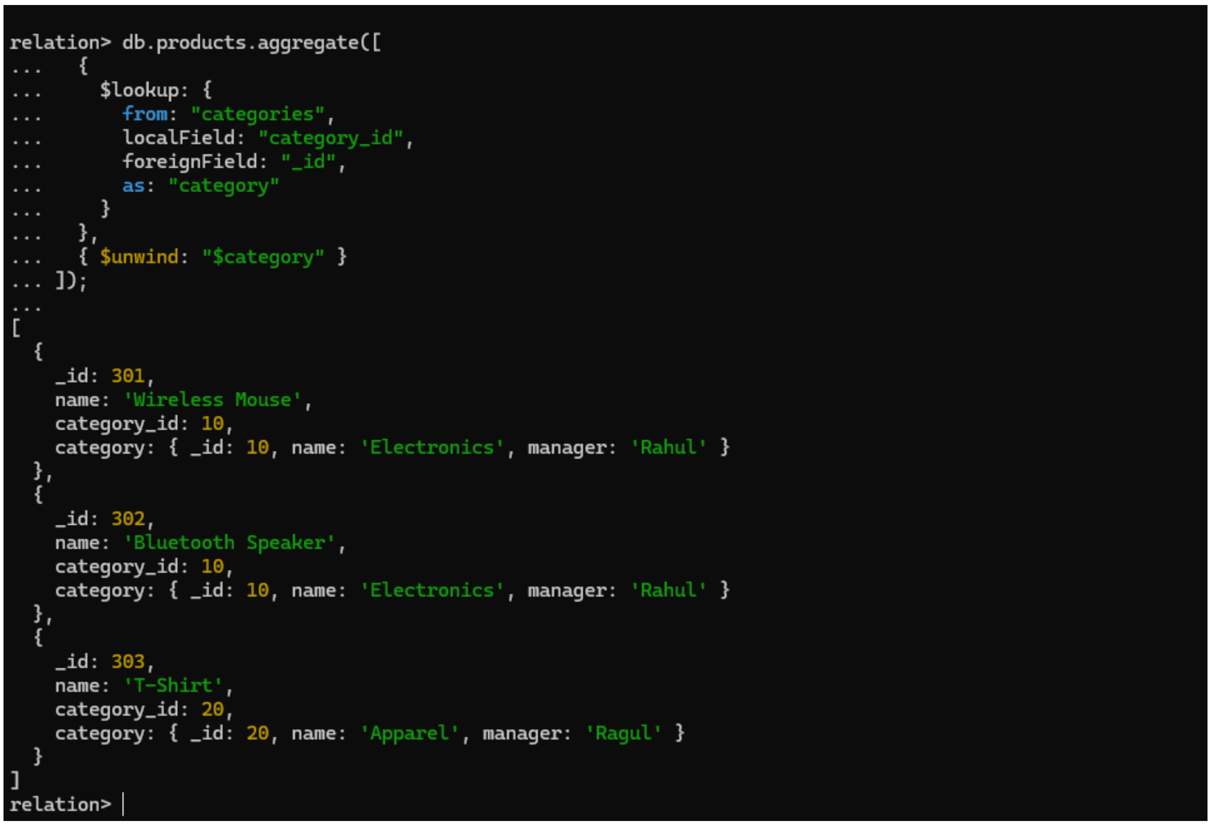
```
relation> db.products.insertMany([
...    { _id: 301, name: "Wireless Mouse", category_id: 10 },
...    { _id: 302, name: "Bluetooth Speaker", category_id: 10 },
...    { _id: 303, name: "T-Shirt", category_id: 20 }
... ]);
...
{ acknowledged: true, insertedIds: { '0': 301, '1': 302, '2': 303 } }
relation> |
```

Categories collection



```
relation> db.categories.insertMany([
...    { _id: 10, name: "Electronics", manager: "Rahul" },
...    { _id: 20, name: "Apparel", manager: "Ragul" }
... ]);
...
{ acknowledged: true, insertedIds: { '0': 10, '1': 20 } }
relation> |
```

Query with $lookup

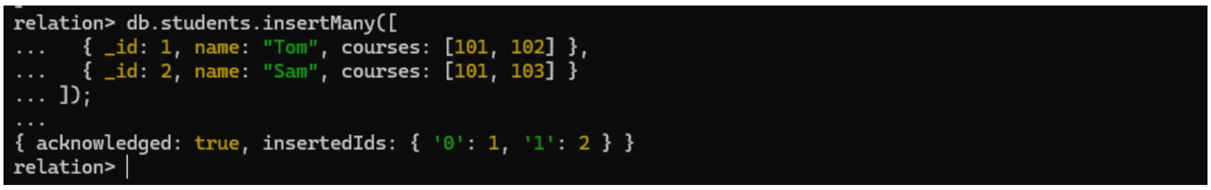


```
relation> db.products.aggregate([
...    {
...      $lookup: {
...        from: "categories",
...        localField: "category_id",
...        foreignField: "_id",
...        as: "category"
...      }
...    },
...    { $unwind: "$category" }
... ]);
...
[
  {
    _id: 301,
    name: 'Wireless Mouse',
    category_id: 10,
    category: { _id: 10, name: 'Electronics', manager: 'Rahul' }
  },
  {
    _id: 302,
    name: 'Bluetooth Speaker',
    category_id: 10,
    category: { _id: 10, name: 'Electronics', manager: 'Rahul' }
  },
  {
    _id: 303,
    name: 'T-Shirt',
    category_id: 20,
    category: { _id: 20, name: 'Apparel', manager: 'Ragul' }
  }
]
relation> |
```

4. Many-to-Many Relationship

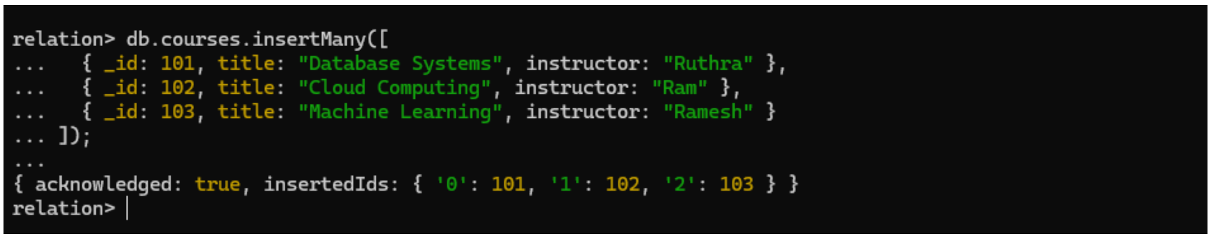
Referenced Approach

Students collection



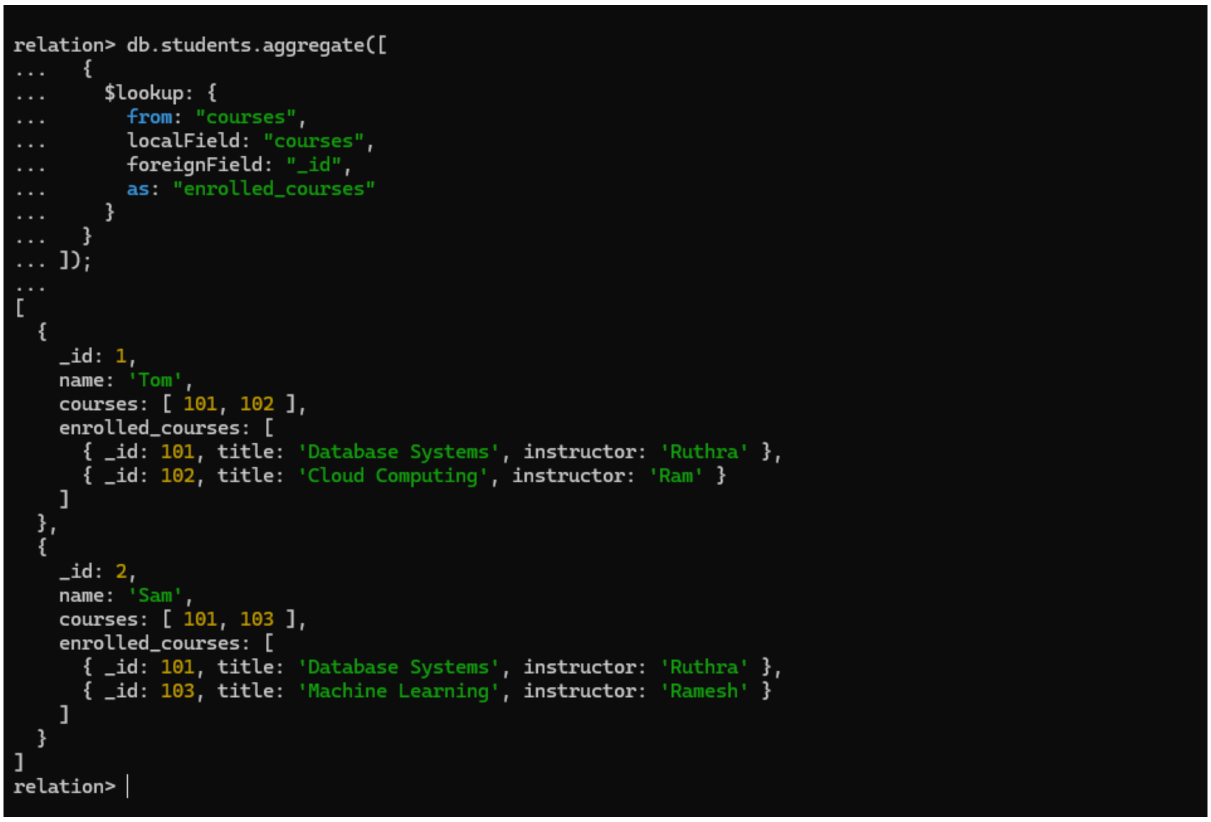
```
relation> db.students.insertMany([
...   { _id: 1, name: "Tom", courses: [101, 102] },
...   { _id: 2, name: "Sam", courses: [101, 103] }
... ]);
...
{ acknowledged: true, insertedIds: { '0': 1, '1': 2 } }
relation> |
```

Courses collection



```
relation> db.courses.insertMany([
...   { _id: 101, title: "Database Systems", instructor: "Ruthra" },
...   { _id: 102, title: "Cloud Computing", instructor: "Ram" },
...   { _id: 103, title: "Machine Learning", instructor: "Ramesh" }
... ]);
...
{ acknowledged: true, insertedIds: { '0': 101, '1': 102, '2': 103 } }
relation> |
```

Query students with their courses

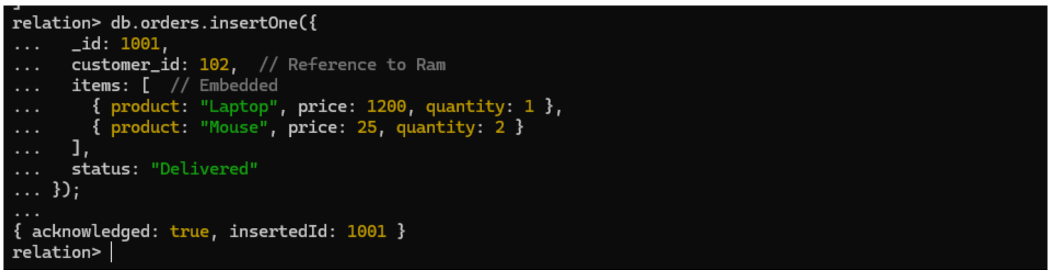


```
relation> db.students.aggregate([
...   {
...     $lookup: {
...       from: "courses",
...       localField: "courses",
...       foreignField: "_id",
...       as: "enrolled_courses"
...     }
...   }
... ]);
...
[
  {
    _id: 1,
    name: 'Tom',
    courses: [ 101, 102 ],
    enrolled_courses: [
      { _id: 101, title: 'Database Systems', instructor: 'Ruthra' },
      { _id: 102, title: 'Cloud Computing', instructor: 'Ram' }
    ]
  },
  {
    _id: 2,
    name: 'Sam',
    courses: [ 101, 103 ],
    enrolled_courses: [
      { _id: 101, title: 'Database Systems', instructor: 'Ruthra' },
      { _id: 103, title: 'Machine Learning', instructor: 'Ramesh' }
    ]
  }
]
relation> |
```

5. Advanced Techniques

Hybrid Approach (Embedding + Referencing)

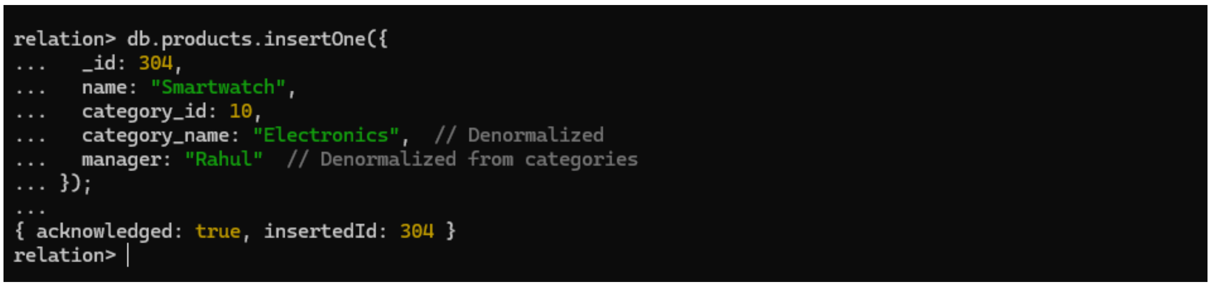
Example: Order with embedded items + referenced user



```
relation> db.orders.insertOne({
...    _id: 1001,
...    customer_id: 102,  // Reference to Ram
...    items: [  // Embedded
...      { product: "Laptop", price: 1200, quantity: 1 },
...      { product: "Mouse", price: 25, quantity: 2 }
...    ],
...    status: "Delivered"
... });
...
{ acknowledged: true, insertedId: 1001 }
relation>
```

Denormalization Example

Storing category name in product (avoiding frequent lookups)



```
relation> db.products.insertOne({
...    _id: 304,
...    name: "Smartwatch",
...    category_id: 10,
...    category_name: "Electronics",  // Denormalized
...    manager: "Rahul"  // Denormalized from categories
... });
...
{ acknowledged: true, insertedId: 304 }
relation>
```

6. Best Practices Summary

Scenario	Recommended Approach	Example
One-to-One	Embedding	Ruthra ↔ Profile
One-to-Few	Embedding	Ramesh's blog comments
One-to-Many	Referencing	Sam's product reviews
Many-to-Many	Referencing	Tom ↔ Courses
Frequent Reads	Embedding	User preferences
Frequent Updates	Referencing	Order history

Final Notes

Always index reference fields (category_id, user_id, etc.)
Test with real-world data volumes before finalizing schema
Combine embedding & referencing when needed (hybrid approach)