MongoDB Coding Assessment:

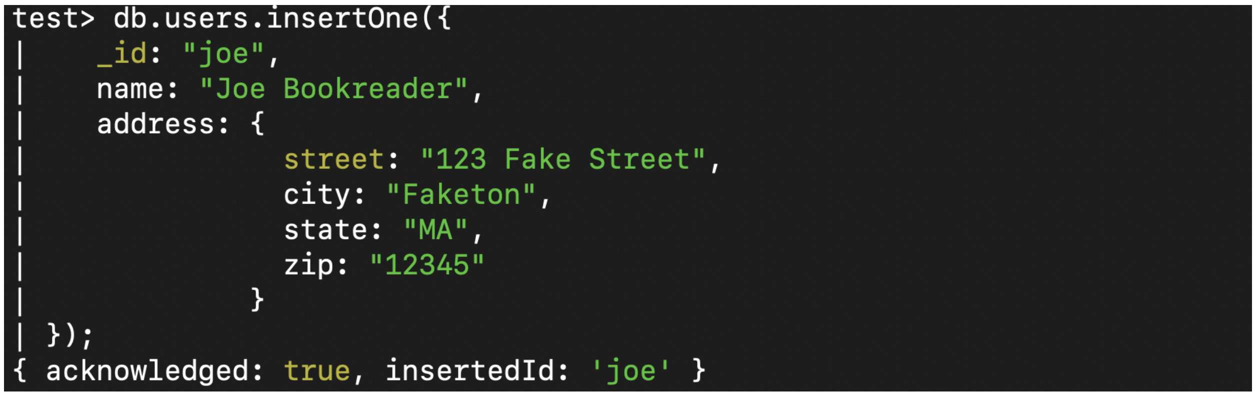
Documents Relationships

By Esaq A

In MongoDB, we can model relationships between data using embedding and referencing. There are 4 types of document relationships:

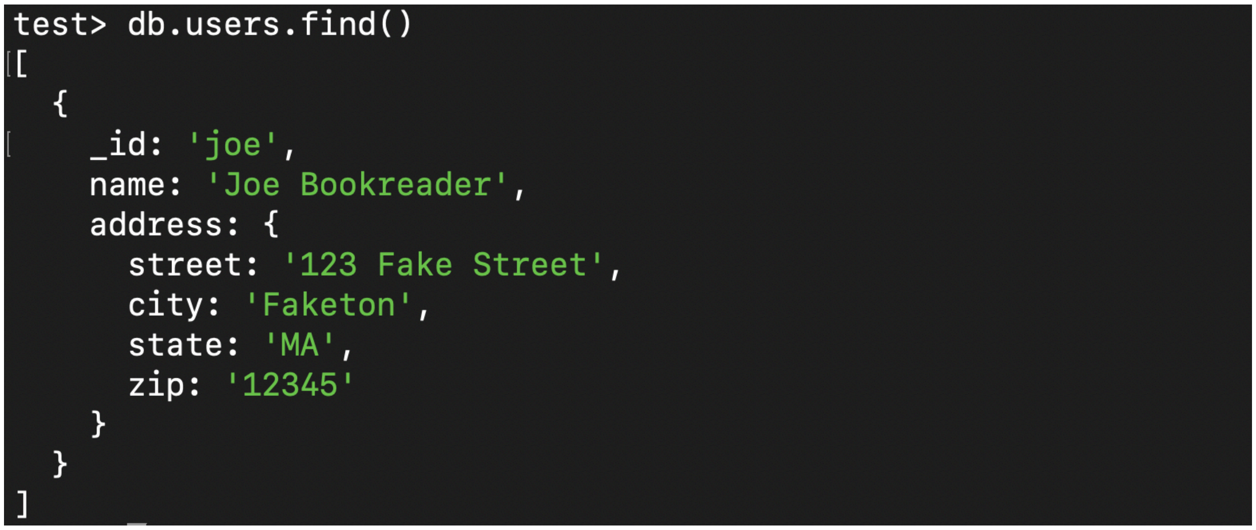
1 . One-to-One Relationships with Embedded Documents:
 - This relationship involves **embedding a single sub-document** within a parent document. It's ideal when the related data is accessed together in a single query and doesn't have its own separate identity.

Example: Embedding user and address details



```
test> db.users.insertOne({
|     _id: "joe",
|     name: "Joe Bookreader",
|     address: {
|                 street: "123 Fake Street",
|                 city: "Faketon",
|                 state: "MA",
|                 zip: "12345"
|               }
| });
{ acknowledged: true, insertedId: 'joe' }
```

Instead of having two separate collections of users and addresses, we can embed addresses into users collection.



```
test> db.users.find()
[
  {
    _id: 'joe',
    name: 'Joe Bookreader',
    address: {
      street: '123 Fake Street',
      city: 'Faketon',
      state: 'MA',
      zip: '12345'
    }
  }
]
```

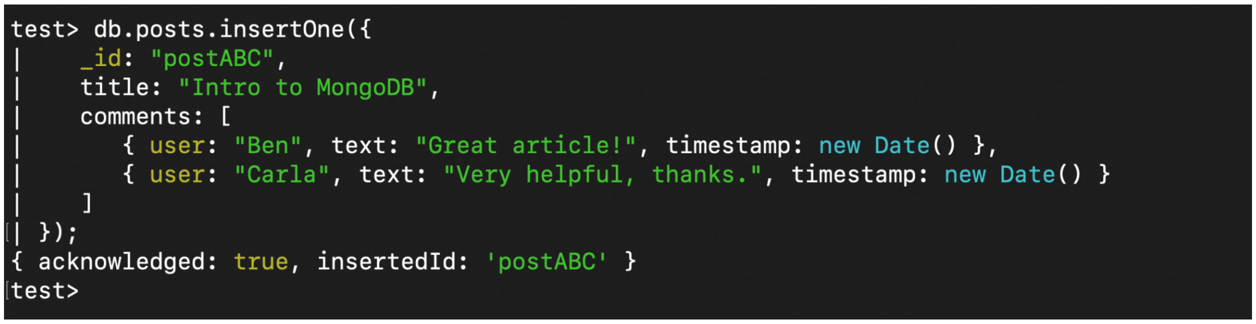
We can get all the details in a single query.

2 . One-to-Many Relationships with Embedded Documents:

- Here, a parent document contains an **array of embedded sub-documents**. This works best when the "many" side is limited in size and doesn't need to be accessed independently.

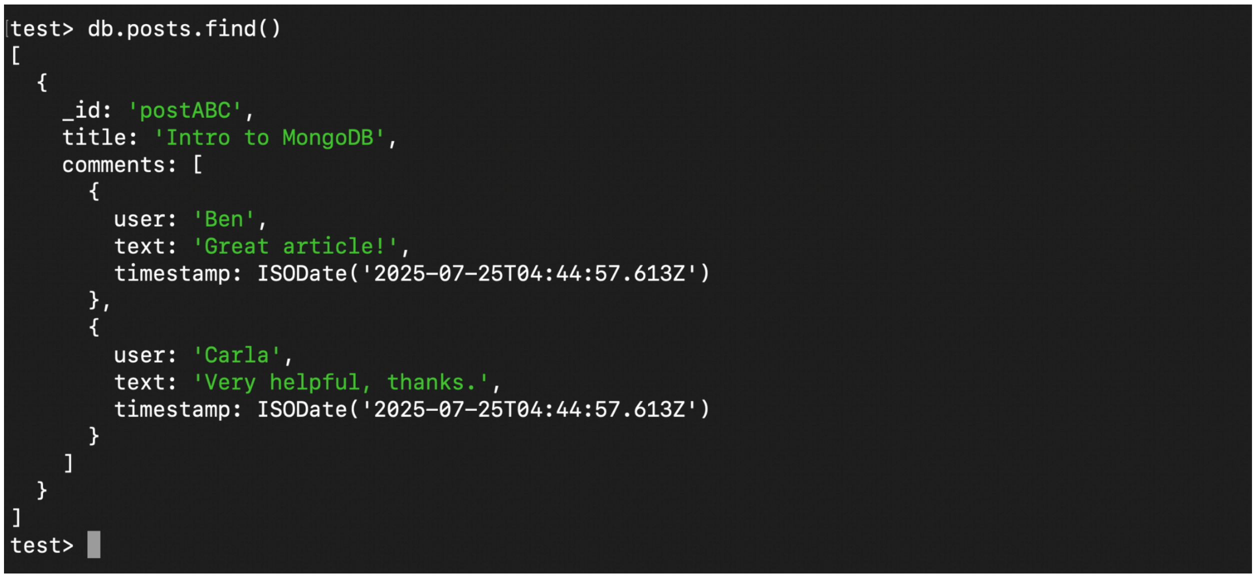
Example: A blog post and its comments.

- Insert Data:



```
test> db.posts.insertOne({
|     _id: "postABC",
|     title: "Intro to MongoDB",
|     comments: [
|        { user: "Ben", text: "Great article!", timestamp: new Date() },
|        { user: "Carla", text: "Very helpful, thanks.", timestamp: new Date() }
|     ]
| });
{ acknowledged: true, insertedId: 'postABC' }
test>
```

- Find Data:



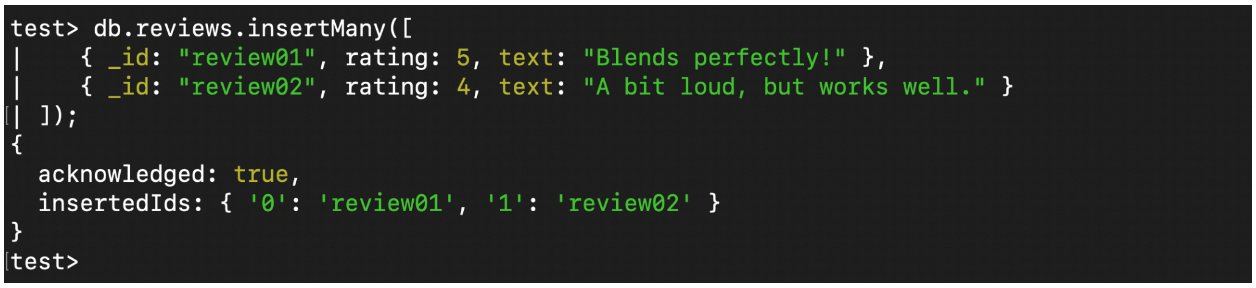
```
test> db.posts.find()
[
  {
    _id: 'postABC',
    title: 'Intro to MongoDB',
    comments: [
      {
        user: 'Ben',
        text: 'Great article!',
        timestamp: ISODate('2025-07-25T04:44:57.613Z')
      },
      {
        user: 'Carla',
        text: 'Very helpful, thanks.',
        timestamp: ISODate('2025-07-25T04:44:57.613Z')
      }
    ]
  }
]
test>
```

3 . One-to-Many Relationships with Document References:

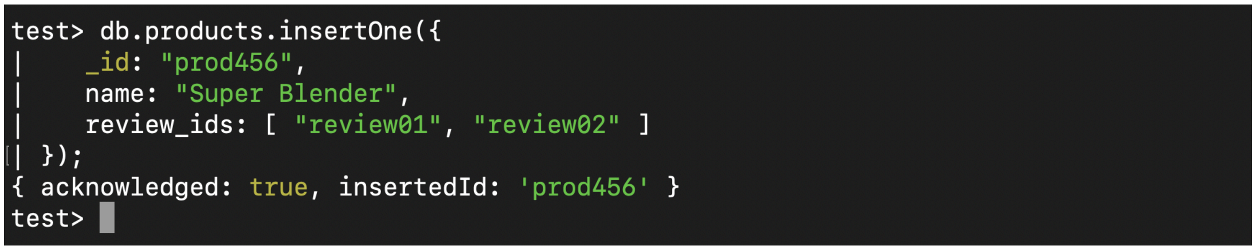
- This pattern stores an **array of _id**s in the parent document, which reference documents in another collection. Use this when the "many" side is large, grows indefinitely, or needs to be queried separately.

Example: A product and its numerous reviews.

- Insert Data:

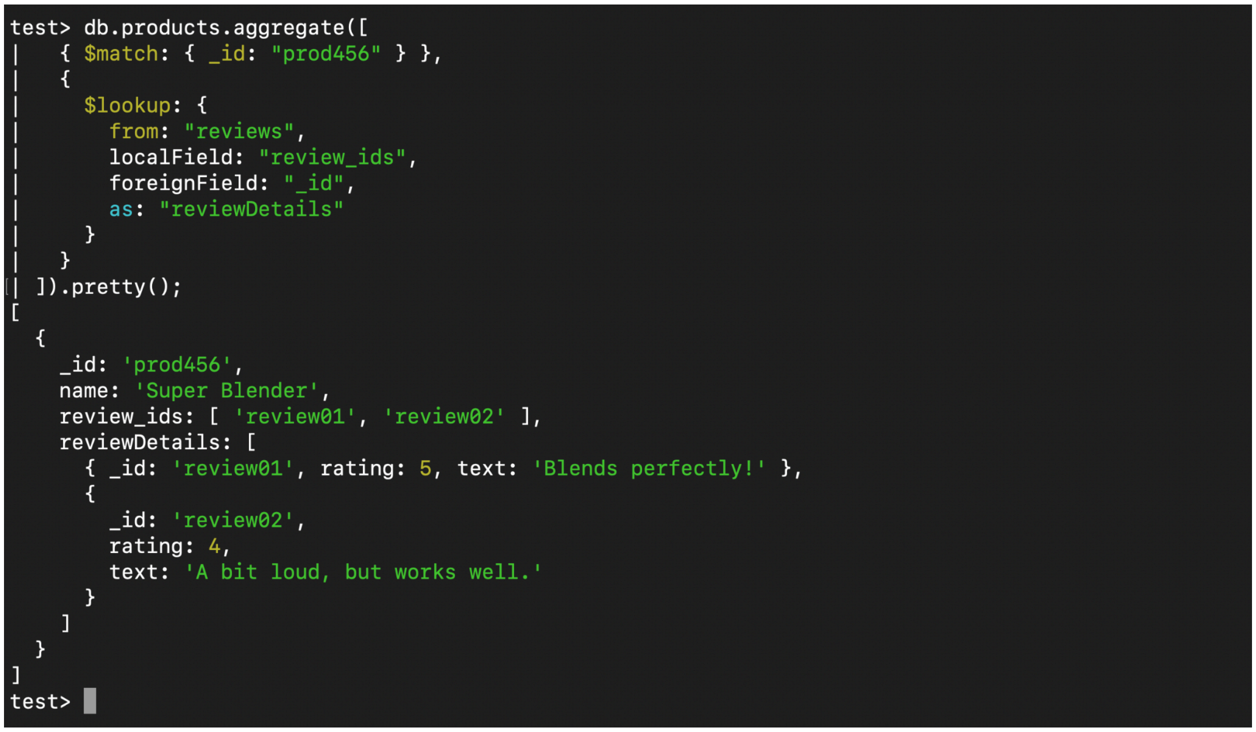


```
test> db.reviews.insertMany([
|    { _id: "review01", rating: 5, text: "Blends perfectly!" },
|    { _id: "review02", rating: 4, text: "A bit loud, but works well." }
| ]);
{
  acknowledged: true,
  insertedIds: { '0': 'review01', '1': 'review02' }
}
test>
```



```
test> db.products.insertOne({
|    _id: "prod456",
|    name: "Super Blender",
|    review_ids: [ "review01", "review02" ]
| });
{ acknowledged: true, insertedId: 'prod456' }
test>
```

- Find Data:

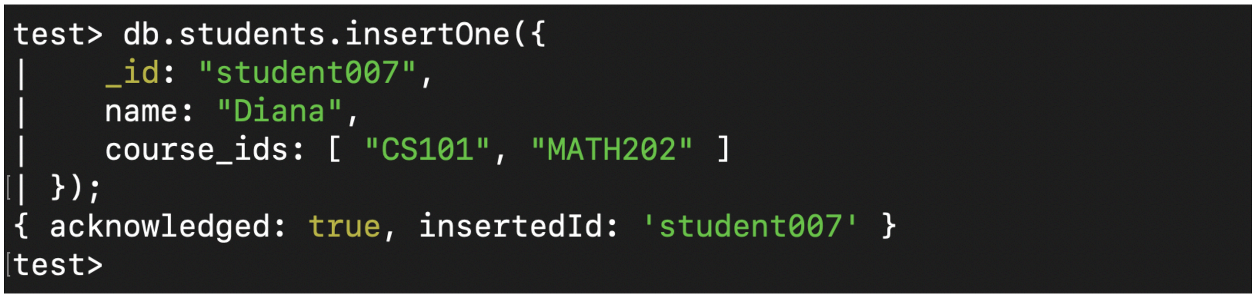

```
test> db.products.aggregate([
|   { $match: { _id: "prod456" } },
|   {
|     $lookup: {
|       from: "reviews",
|       localField: "review_ids",
|       foreignField: "_id",
|       as: "reviewDetails"
|     }
|   }
[| ]).pretty();
[
  {
    _id: 'prod456',
    name: 'Super Blender',
    review_ids: [ 'review01', 'review02' ],
    reviewDetails: [
      { _id: 'review01', rating: 5, text: 'Blends perfectly!' },
      {
        _id: 'review02',
        rating: 4,
        text: 'A bit loud, but works well.'
      }
    ]
  }
]
test> 
```

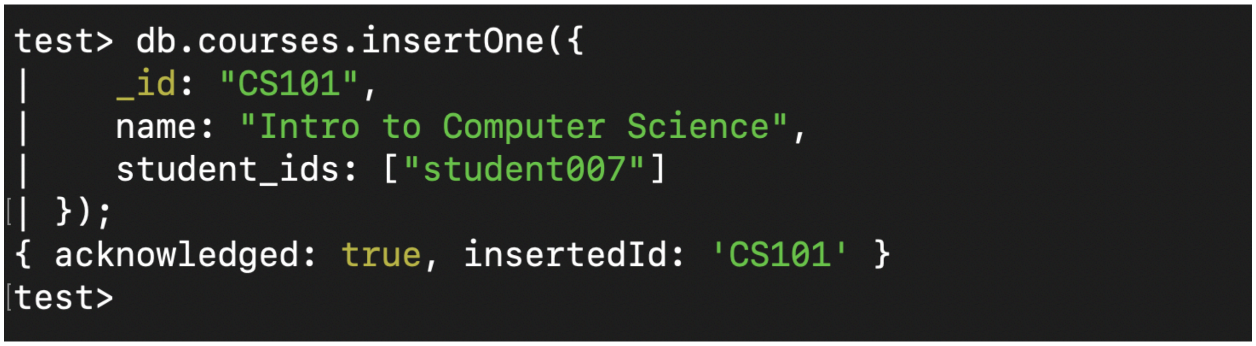
4 . Many-to-Many Relationships with Embedded Documents:

This involves **storing an array of references (_ids) in documents on both sides** of the relationship. This is the most flexible approach for many-to-many scenarios.
Example: Students and courses, where a student can take many courses, and a course can have many students.

- Insert Data:

Insert students and courses with arrays of references

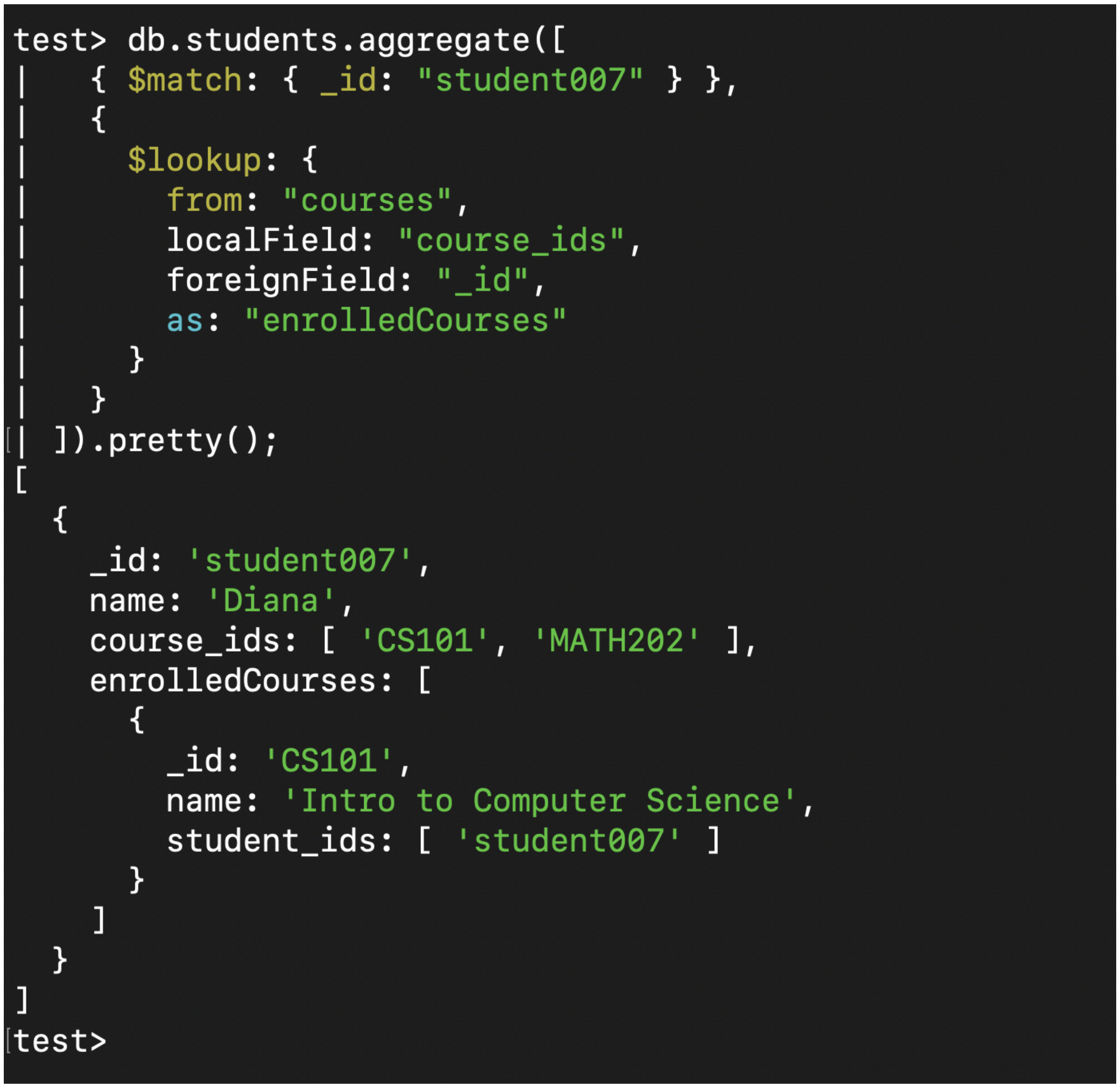

```
test> db.students.insertOne({
|     _id: "student007",
|     name: "Diana",
|     course_ids: [ "CS101", "MATH202" ]
[| });
{ acknowledged: true, insertedId: 'student007' }
[test>
```



```
test> db.courses.insertOne({
|     _id: "CS101",
|     name: "Intro to Computer Science",
|     student_ids: ["student007"]
[| });
{ acknowledged: true, insertedId: 'CS101' }
[test>
```

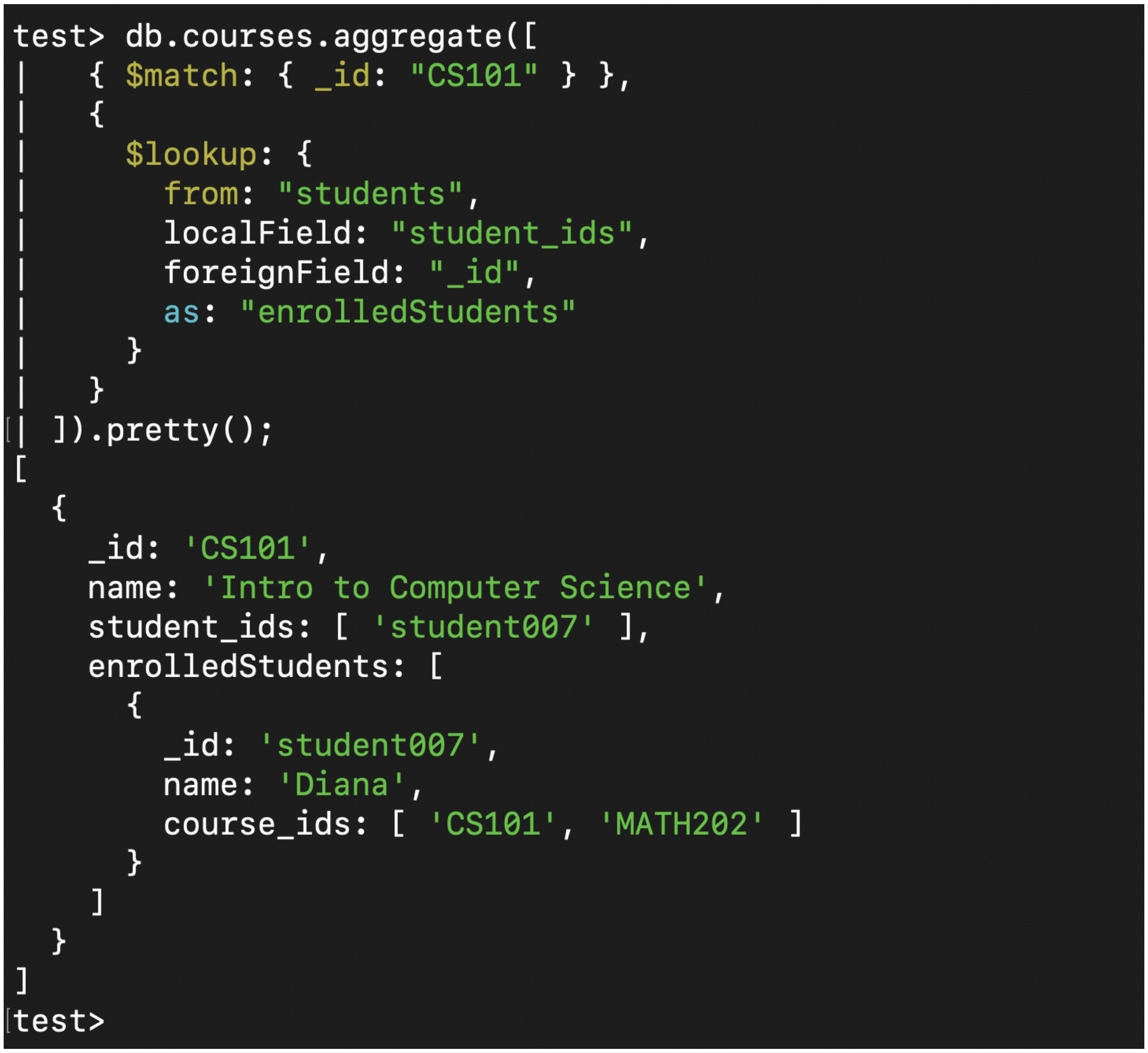
- Find Data:

- Find a student and their enrolled courses



```
test> db.students.aggregate([
|    { $match: { _id: "student007" } },
|    {
|      $lookup: {
|        from: "courses",
|        localField: "course_ids",
|        foreignField: "_id",
|        as: "enrolledCourses"
|      }
|    }
[| ]).pretty();
[
  {
    _id: 'student007',
    name: 'Diana',
    course_ids: [ 'CS101', 'MATH202' ],
    enrolledCourses: [
      {
        _id: 'CS101',
        name: 'Intro to Computer Science',
        student_ids: [ 'student007' ]
      }
    ]
  }
]
[test>
```

- Find a course and its enrolled students



```
test> db.courses.aggregate([
|   { $match: { _id: "CS101" } },
|   {
|     $lookup: {
|       from: "students",
|       localField: "student_ids",
|       foreignField: "_id",
|       as: "enrolledStudents"
|     }
|   }
| ]).pretty();
[
  {
    _id: 'CS101',
    name: 'Intro to Computer Science',
    student_ids: [ 'student007' ],
    enrolledStudents: [
      {
        _id: 'student007',
        name: 'Diana',
        course_ids: [ 'CS101', 'MATH202' ]
      }
    ]
  }
]
test>
```