Simple Techniques to Perform Join Operations in MongoDB

Master Generative AI: Your clear, step-by-step guide to

Analytics Vidhya

Home › Beginner

› Simple Techniques to Perform Join Operations in MongoDB

Aniket Mitra
04 Jul, 2023 • 8 min read

Introduction

Database people are very much familiar with **JOINS**. When we want to fetch data from multiple tables we often join the tables based on Primary Keys and Foreign Keys. Let's learn simple techniques to perform join operations in MongoDB in this article.

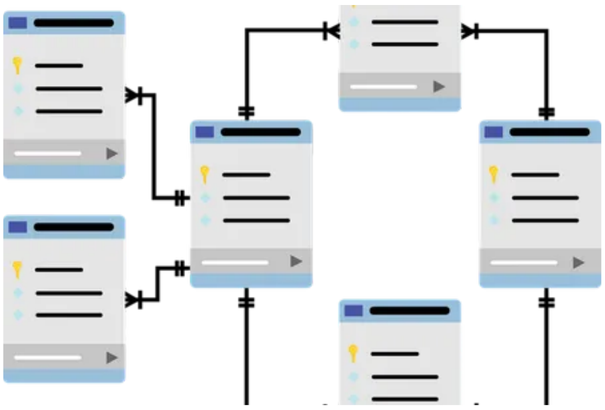


Image Source: Pixabay

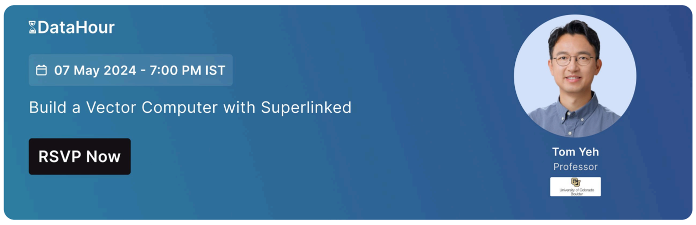
The above diagram is a pictorial representation of the relational database schema of any organization. Those blocks are tables storing particular kinds of data (students/professors/employees) and the lines & arrows represent the relation between the tables using common keys. We can perform joins across the tables based on the relations between them.

For eg: In an **Organization**, there are separate tables for storing data of employees, departments, projects, etc where data is stored in a *normalized* way. To fetch the **details of employees and in which department and** store students' and professors' data. To find out **which all professors teach a particular student**, we need to perform join across the tables.

Learning Objective

In this particular tutorial, we are going to see how we can perform different join operations*(Inner Join, Outer Join, Right Join & Left Join Operations)* in MongoDB.

This article was published as a part of the Data Science Blogathon.

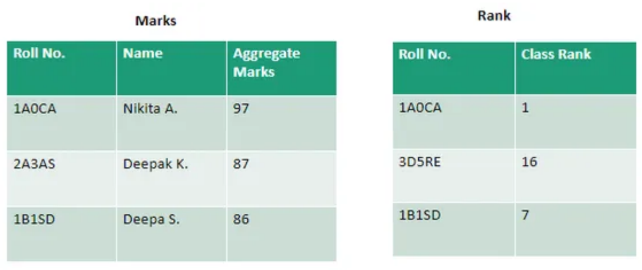

DataHour
07 May 2024 - 7:00 PM IST
Build a Vector Computer with Superlinked
RSVP Now
Tom Yeh
Professor

Understanding Different Types of Common Join Operations

A. SQL & Different Types of Joins

The majority of us have knowledge of SQL databases. There we often perform four main kinds of joins which we will discuss below.

1. Inner Join: Only rows with common keys from both tables will be there in the resultant table.



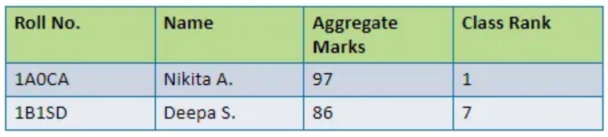
Marks

Roll No.	Name	Aggregate Marks
1A0CA	Nikita A.	97
2A3AS	Deepak K.	87
1B1SD	Deepa S.	86

Rank

Roll No.	Class Rank
1A0CA	1
3D5RE	16
1B1SD	7

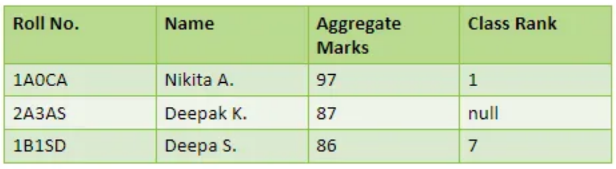
Two Tables of School Dataset – Marks & Rank



Roll No.	Name	Aggregate Marks	Class Rank
1A0CA	Nikita A.	97	1
1B1SD	Deepa S.	86	7

Inner Join

2. Left Outer Join: All rows of left tables(matching + non-matching keys) will be on the resultant table. Thus in the resultant table, there be only rows of matching keys from the right table; all rows where keys not matching, you can't eliminate it.

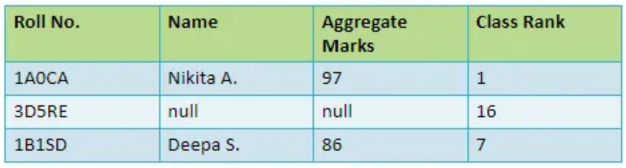


Roll No.	Name	Aggregate Marks	Class Rank
1A0CA	Nikita A.	97	1
2A3AS	Deepak K.	87	null
1B1SD	Deepa S.	86	7

Left Outer Join

After performing the left join, we have all the columns from the left table. Since **Class Rank** for **Deepak K.** is not present in the right table so fill it with null. As discussed only those records of the right table which match the **Roll No** with the left table are present in the results. For that reason tuple **(3D5RE,16)** from the right table is not in the result.

3. Right Outer Join: Opposite of left outer join. Here all rows of the right table will be there in the resultant table and only rows with matching keys will be there from the left table.

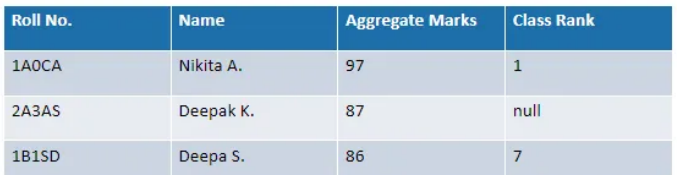


Roll No.	Name	Aggregate Marks	Class Rank
1A0CA	Nikita A.	97	1
3D5RE	null	null	16
1B1SD	Deepa S.	86	7

Right Outer Join

As expected all records/tuples from the right table are present in the result, but the record **(2A3AS, Deepak K., 87)** from the left table is absent.

4. Full Outer Join: All the rows from both tables (matching+non-matching keys) will be present in the resultant table.



Roll No.	Name	Aggregate Marks	Class Rank
1A0CA	Nikita A.	97	1
2A3AS	Deepak K.	87	null
1B1SD	Deepa S.	86	7

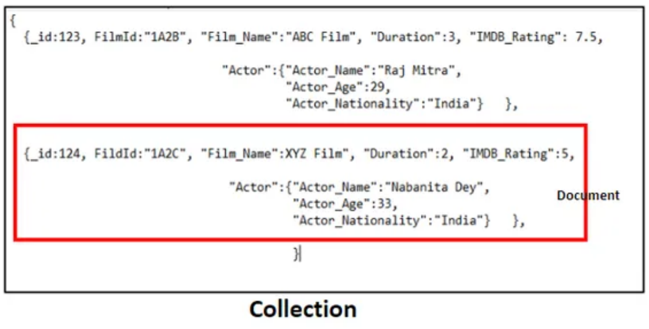
our result. The places where values are not present are fill it with **null**.

B. Brief Introduction to MongoDB

MongoDB is a document-based NoSQL Database. NoSQL databases are better for storing large-scale, non-relational, unstructured and frequently changing data. The below two blogs have comparison and operations in MongoDB.

- Introduction to MongoDB
- CRUD Operations in MongoDB

MongoDB database consists of one or more **Collections**. Collections can be considered equivalent to tables in SQL Databases. Each collection consists of one or more **documents**. So documents can be thought of as rows or tuples of a table in SQL databases. Data is stored in ***BSON(Binary JSON format)*** inside MongoDB.



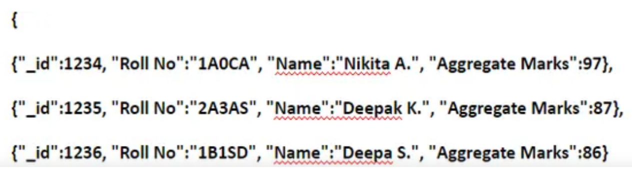
```
{
 {_id:123, FilmId:"1A2B", "Film_Name":"ABC Film", "Duration":3, "IMDB_Rating": 7.5,

                    "Actor":{"Actor_Name":"Raj Mitra",
                             "Actor_Age":29,
                             "Actor_Nationality":"India"}   },

 {_id:124, FildId:"1A2C", "Film_Name":XYZ Film", "Duration":2, "IMDB_Rating":5,

                     "Actor":{"Actor_Name":"Nabanita Dey",
                              "Actor_Age":33,
                              "Actor_Nationality":"India"}   },
                              }
```

MongoDB Data Storage Format

Join Operation in MongoDB

Now let's see how different join operations will perform on MongoDB collections.

Convert the two tables **Marks** and **Rank** to Collections with each tuple within them as the documents of the respective collections. Store the collections in a database named **School** inside MongoDB.



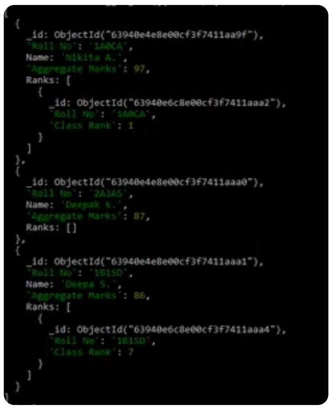
```
{
{"_id":1234, "Roll No":"1A0CA", "Name":"Nikita A.", "Aggregate Marks":97},
{"_id":1235, "Roll No":"2A3AS", "Name":"Deepak K.", "Aggregate Marks":87},
{"_id":1236, "Roll No":"1B1SD", "Name":"Deepa S.", "Aggregate Marks":86}
{"_id":1111, "Roll No":"1A0CA", "Class Rank":1},

{_"id":1112, "Roll No":"3D5RE", "Class Rank":16},

{"_id":1123, "Roll No":"1B1SD", "Class Rank":7}

}
```

Ranks

Left Outer Join

Code:

```
db.Marks.aggregate([{$lookup:{from:"Rank",localField:"Roll
foreignField:"Roll No",as:"Ranks"}}])
```



```
{
  _id: ObjectId("63940e4e8e00cf3f7411aa9f"),
  'Roll No': '1A0CA',
  Name: 'Nikita A.',
  'Aggregate Marks': 97,
  Ranks: [
    {
      _id: ObjectId("63940e6c8e00cf3f7411aaa2"),
      'Roll No': '1A0CA',
      'Class Rank': 1
    }
  ]
},
{
  _id: ObjectId("63940e4e8e00cf3f7411aaa0"),
  'Roll No': '2A3AS',
  Name: 'Deepak K.',
  'Aggregate Marks': 87,
  Ranks: []
},
{
  _id: ObjectId("63940e4e8e00cf3f7411aaa1"),
  'Roll No': '1B1SD',
  Name: 'Deepa S.',
  'Aggregate Marks': 86,
  Ranks: [
    {
      _id: ObjectId("63940e6c8e00cf3f7411aaa4"),
      'Roll No': '1B1SD',
      'Class Rank': 7
    }
  ]
}
]
```

Left Outer Join

[Enlarge Output]

As we can see the rank details of the respective student are appended to their document. For **Deepak**, there are no rank details in the **Rank** table so obviously, his **Ranks** field is an empty list.

Now let us understand the parameters used:

- Here **Marks** is our left table.
- **$lookup** is the aggregate function/operator for performing join across two collections.
- Inside lookup, **from** denotes the collection with which we want to perform join i.e, our right table(collection). In our case Rank is our right collection.
- **localField** denotes the key from the left collection which will be matched with the key from the right collection to perform join. If a matching key is found in

case in our example).

- **foreignField** is the key from the right collection.
- **as** denotes the name of the new field which will form in the resulting table/collection due to joining where the details from the right table(collection) will be stored.
- In our case, add the new field **Ranks** to the resulting table(collection) which contains the details of the ranks of corresponding students.

Now one thing to remember, in MongoDB **$lookup** can only perform left joins and there is no specific syntax available for other types of joins. So we need to derive the other joins by using different tricks and operations

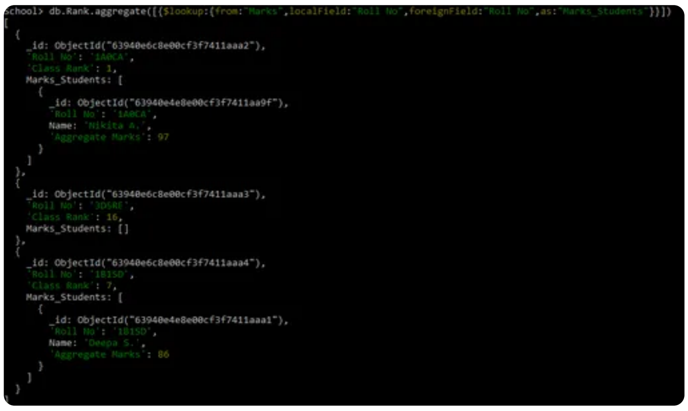
Right Outer Join

Now right join is just opposite to the left join where apart from matching records, non-matching records of the right collection/table should also be there in the resulting collection/table.

Now a simple way to do that is just to alter the position of two collections; then our right collection becomes the left and vice-versa. So now, the join will contain all the rows(matching+non-matching) of our right table.

Code:

```
db.Rank.aggregate([{$lookup:{from:"Marks", localField:"Rol
foreignField:"Roll No",  as:"Marks_Students"}}])
```



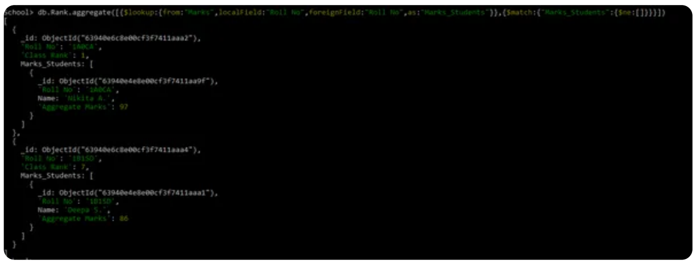
```
school> db.Rank.aggregate([{$lookup:{from:"Marks",localField:"Roll No",foreignField:"Roll No",as:"Marks_Students"}}])
[
  {
    _id: ObjectId("63940e6c8e00cf3f7411aaa2"),
    'Roll No': '1A0CA',
    'Class Rank': 1,
    Marks_Students: [
      {
        _id: ObjectId("63940e4e8e00cf3f7411aa9f"),
        'Roll No': '1A0CA',
        Name: 'Nikita A.',
        'Aggregate Marks': 97
      }
    ]
  },
  {
    _id: ObjectId("63940e6c8e00cf3f7411aaa3"),
    'Roll No': '3D5RE',
    'Class Rank': 16,
    Marks_Students: []
  },
  {
    _id: ObjectId("63940e6c8e00cf3f7411aaa4"),
    'Roll No': '1B1SD',
    'Class Rank': 7,
    Marks_Students: [
      {
        _id: ObjectId("63940e4e8e00cf3f7411aaa1"),
        'Roll No': '1B1SD',
        Name: 'Deepa S.',
        'Aggregate Marks': 86
      }
    ]
  }
```

[Enlarge Output]

trick!!! We will do a left join and then remove all those records where the **as** field is empty. So we will only be left with the records where the keys are present in both tables(collections).

Code:

```
db.Rank.aggregate([{$lookup:{from:"Marks", localField:"Rol
as: "Marks_Students"}}, {$match:{"Marks_Students":{$ne:[]}
```



```
school> db.Rank.aggregate([{$lookup:{from:"Marks",localField:"Roll No",foreignField:"Roll No",as:"Marks_Students"}},{$match:{"Marks_Students":{$ne:[]}}}])
[
  {
    _id: ObjectId("63940e6c8e00cf3f7411aaa2"),
    'Roll No': '1A0CA',
    'Class Rank': 1,
    Marks_Students: [
      {
        _id: ObjectId("63940e4e8e00cf3f7411aa9f"),
        'Roll No': '1A0CA',
        Name: 'Nikita A.',
        'Aggregate Marks': 97
      }
    ]
  },
  {
    _id: ObjectId("63940e6c8e00cf3f7411aaa4"),
    'Roll No': '1B15D',
    'Class Rank': 7,
    Marks_Students: [
      {
        _id: ObjectId("63940e4e8e00cf3f7411aaa1"),
        'Roll No': '1B15D',
        Name: 'Deepa S.',
        'Aggregate Marks': 86
      }
    ]
  }
```

Inner Join Result

[Enlarge Output]

As we can see in the above result we only have the records for which the keys from both collections matched. Here ***{$match:{“Marks_Students”:{$ne:[]}}}*** dictates to match only those records where **Marks_Students** field is not [](*empty list*)

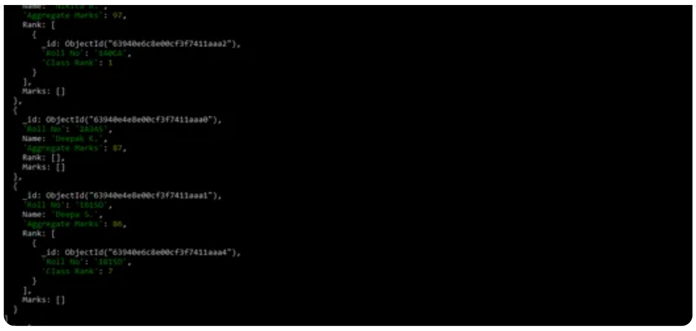
Full Outer Join

Full outer join is a little complicated, I designed it by a combination of 3 operations. So if it looks confusing on the first go, I request you to give it a read multiple times for better understanding.

Step 1: We will do a left join of **Marks**(left collection) and **Rank**(right collection) and add an empty field named Marks to all the resultant records and send/output the result in a new collection called **J2**.

Code:

```
db.Marks.aggregate([{$lookup:{from:"Rank",localField:"Roll
{$addFields:{Marks:[]}},{$out:"J2"}])
```



```
    'Aggregate Marks': 97,
    Rank: [
      {
        _id: ObjectId("63940e6c8e00cf3f7411aaa2"),
        'Roll No': '1A0CA',
        'Class Rank': 1
      }
    ],
    Marks: []
  },
  {
    _id: ObjectId("63940e4e8e00cf3f7411aaa0"),
    'Roll No': '2A3AS',
    Name: 'Deepak K.',
    'Aggregate Marks': 87,
    Rank: [],
    Marks: []
  },
  {
    _id: ObjectId("63940e4e8e00cf3f7411aaa1"),
    'Roll No': '1B1SD',
    Name: 'Deepa S.',
    'Aggregate Marks': 86,
    Rank: [
      {
        _id: ObjectId("63940e6c8e00cf3f7411aaa4"),
        'Roll No': '1B1SD',
        'Class Rank': 7
      }
    ],
    Marks: []
  }
```

Result of Step 1

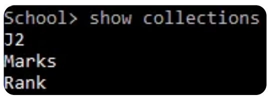
[Enlarge Output]

So our new collection looks like the screenshot above.

- **{$addFields:{Marks:[]}}** -> add extra field Marks to all records.
- **{$out:"J2"}]** -> outputs/sends the result to a new collection **J2**.

So now as obvious, our database **School** contains 3 collections-

Marks, Rank, J2



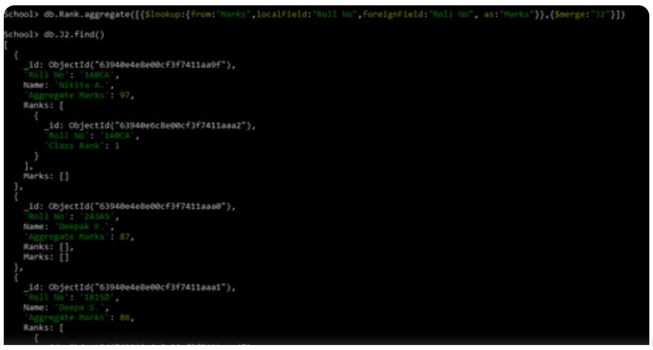
```
School> show collections
J2
Marks
Rank
```

Collections

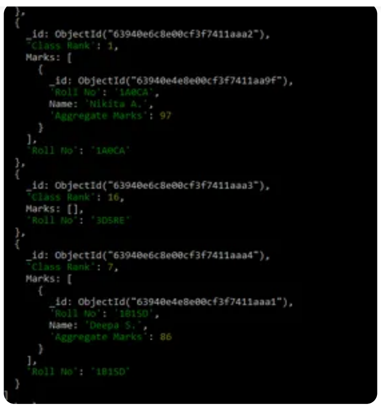
Step 2: We will perform the right join(as discussed before considering **Rank** as the left collection)
of **Marks** and **Rank** and append the result to
the **J2** collection.

Code:

```
db.Rank.aggregate([{$lookup:{from:"Marks",localField:"Roll
 as:"Marks"}},{$merge:"J2"}])
```



```
School> db.Rank.aggregate([{$lookup:{from:"Marks",localField:"Roll No",foreignField:"Roll No", as:"Marks"}},{$merge:"J2"}])

School> db.J2.find()
[
  {
    _id: ObjectId("63940e4e8e00cf3f7411aa9f"),
    'Roll No': '1A0CA',
    Name: 'Nikita A.',
    'Aggregate Marks': 97,
    Ranks: [
      {
        _id: ObjectId("63940e6c8e00cf3f7411aaa2"),
        'Roll No': '1A0CA',
        'Class Rank': 1
      }
    ],
    Marks: []
  },
  {
    _id: ObjectId("63940e4e8e00cf3f7411aaa0"),
    'Roll No': '2A3AS',
    Name: 'Deepak K.',
    'Aggregate Marks': 87,
    Ranks: [],
    Marks: []
  },
  {
    _id: ObjectId("63940e4e8e00cf3f7411aaa1"),
    'Roll No': '1B1SD',
    Name: 'Deepa S.',
    'Aggregate Marks': 86,
    Ranks: [
      {
```



```
},
{
  _id: ObjectId("63940e6c8e00cf3f7411aaa2"),
  'Class Rank': 1,
  Marks: [
    {
      _id: ObjectId("63940e4e8e00cf3f7411aa9f"),
      'Roll No': '1A0CA',
      Name: 'Nikita A.',
      'Aggregate Marks': 97
    }
  ],
  'Roll No': '1A0CA'
},
{
  _id: ObjectId("63940e6c8e00cf3f7411aaa3"),
  'Class Rank': 16,
  Marks: [],
  'Roll No': '3D5RE'
},
{
  _id: ObjectId("63940e6c8e00cf3f7411aaa4"),
  'Class Rank': 7,
  Marks: [
    {
      _id: ObjectId("63940e4e8e00cf3f7411aaa1"),
      'Roll No': '1815D',
      Name: 'Deepa S.',
      'Aggregate Marks': 86
    }
  ],
  'Roll No': '1815D'
}
]
```

Result of Step 2 Contd...

[Enlarge Output]

Notice how the system appends the new output to the bottom of the old output of step 1.

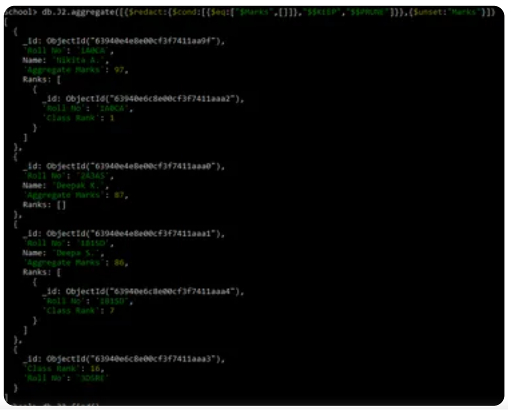
Step 3:

We will only keep/retain the records in our aggregate result where the Match field is [](*empty*) and discard the rest. In this way, remove the duplicates and we will only have in our results all the distinct fields from both collections/tables. *(You may have noticed in the output of Step 2 there are duplicates e.g: There are two records of Nikita)*

Atlast we will remove the empty **Marks** field from the aggregation result since it's an empty field and no use displaying it. Its purpose was to remove the duplicates.

Code:

```
db.J2.aggregate([{$redact:{$cond:[{$eq:["$Marks",[]]},"$KE
{$unset:”Marks”}])
```



```
school> db.J2.aggregate([{$redact:{$cond:[{$eq:["$Marks",[]]},"$$KEEP","$$PRUNE"]}},{$unset:"Marks"}])
[
  {
    _id: ObjectId("63940e4e8e00cf3f7411aa9f"),
    'Roll No': '1A0CA',
    Name: 'Nikita A.',
    'Aggregate Marks': 97,
    Ranks: [
      {
        _id: ObjectId("63940e6c8e00cf3f7411aaa2"),
        'Roll No': '1A0CA',
        'Class Rank': 1
      }
    ]
  },
  {
    _id: ObjectId("63940e4e8e00cf3f7411aaa0"),
    'Roll No': '2A3AS',
    Name: 'Deepak K.',
    'Aggregate Marks': 87,
    Ranks: []
  },
  {
    _id: ObjectId("63940e4e8e00cf3f7411aaa1"),
    'Roll No': '1815D',
    Name: 'Deepa S.',
    'Aggregate Marks': 86,
    Ranks: [
      {
        _id: ObjectId("63940e6c8e00cf3f7411aaa4"),
        'Roll No': '1815D',
        'Class Rank': 7
      }
    ]
  },
  {
    _id: ObjectId("63940e6c8e00cf3f7411aaa3"),
    'Class Rank': 16,
    'Roll No': '3D5RE'
  }
]
```

So finally we have our desired output. We have all the records which matched in both the tables(collections) as well as other records present in either of the tables(collections). (**Deepak's** in **Marks**, and roll no. **3D5RE** in **Rank**).

Conclusion

So we successfully derived different kinds of joins from left join in **MongoDB**. So just to recap, there is only direct syntax available in MongoDB for performing left joins. Other types of joins should be derived by applying different types of operations and techniques on the left join. For eg: removing collections with empty **as** field in case of an inner join, etc.

While deriving these joins we realized:

- Good knowledge of aggregation queries is necessary.
- Careful observation of the intermediate results is needed to decide on the next step.
- There may be other(even better ways) to derive the joins.

If you know better ways you are welcome to share in the comments.

Key Takeaways

- Only the left join operation has direct syntax in MongoDB.
- Derive the right join by altering the position of the collections in the syntax.
- Derive the Inner join by first performing left join and then removing empty as field.
- Outer joins can be performed by a series of simple, clever operations.

References

- MongoDB Documentation on Lookup

Simplified by Samuel Salmon

The media shown in this article is not owned by Analytics Vidhya and is used at the Author's discretion.

blogathon collections join joins left join MongoDB Rank tables

Aniket Mitra
04 Jul 2023

Beginner Database MongoDB NoSQL SQL

Frequently Asked Questions

$lookup

A. Yes, we can only perform the left join of collections using $lookup directly. Perform other join operations using a combination of different operators as shown in this article.

Q2. Can I take a different approach to perform the joins?

Q3. How can I write efficient aggregation queries in MongoDB?

Q4. Can proper indexing influence the performance of the join queries?

Q5. How can we show our Data Manipulations and Exploration works in a proper and presentable format to job recruiters to demonstrate our proficiency in MongoDB?

Write for us

- ✓ Get Expert Feedback
- ✓ Build Your Brand & Audience
- ✓ Cash In on Your Knowledge
- ✓ Join a Thriving Community
- ✓ Level Up Your Data Science Game

Rahul Shah
27

Sion Chakrat
16

Simple Techniques to Perform Join Operations in MongoDB