**How to Perform the SQL Join Equivalent in MongoDB?**

* MongoDB, being a NoSQL database, is quite different from SQL databases. MongoDB is schema-less and document-oriented.
* It means it stores data in flexible BSON documents instead of predefined tables.
* As a result, the concept of joins in MongoDB doesn't directly translate from SQL.
* In MongoDB, achieving joins as SQL joins is possible through aggregation pipelines and the use of the $lookup operator.
* Aggregation pipelines allow us to process data records and transform them using a series of stages.

**Syntax for joins:**

db.modelName.aggregate ([  
 {  
 $lookup: {  
 from: "Collection to Join",  
 localField: "Field from the input documents",  
 foreignField: "Field from the documents of the 'from' collection",  
 as: "Pick a field-name as output"  
 }  
 }  
]);

**Explanation:** This MongoDB aggregation syntax utilizes the $lookup stage to perform a join between two collections.

* **from:** This represents the foreign collection, which we want to join to our current collection.
* **localField:** LocalField specifies the field in the input documents to perform the join operation.
* **foreignField:** The foreignField specifies the field from the foreign collection to match with the localField of the input documents.
* **as:** It denotes the name of the field that will contain the join array data in the output documents.

**1) Left Join:**

* A left join in MongoDB combines matching documents from the primary ("left") collection with documents from the secondary ("right") collection.
* It returns all documents from the primary collection along with related documents from the secondary collection.

db.books.aggregate ([  
 {  
 $lookup: {  
 from: "authors",  
 localField: "authorId",  
 foreignField: "\_id",  
 as: "author"  
 }  
 }  
]);

**Explanation:**

* In the above Query, We have use aggregation pipeline $lookup stage to perform a left join between the books collection and the authors collection based on matching values between the authorId field in the books collection and the \_id field in the authors collection.
* The result is an aggregation output where each document from the books collection includes an additional field named author containing an array of matching documents from the authors collection.

**2) Right Join:**

* In a right join MongoDB reverses the typical roles of the primary and secondary collections.
* It is similar to the left join but we are reversing the input documents like in left-join we combine the authors collection to the books collection but in right-join we combine the books collection into the authors collection.

db.authors.aggregate ([  
 {  
 $lookup: {  
 from: "books",  
 localField: "bookId",  
 foreignField: "\_id",  
 as: "book"  
 }  
 }  
]);

**Explanation:**

* In the above Query, It matches documents from the authors collection where the bookId field corresponds to the \_id field in the books collection. The result is an array named book containing related book documents for each author entry.
* This output array represents authors along with their associated books. Each object in the array contains an authorId, name, and their bookId. If an author has a book, it appears in the book array within the object, showing the bookId, name, and the authorId. For authors without any associated books, the book array is empty ([]), as seen with "Peter Thiel" and "Kristina Chodorow" in the provided example.

**3) Inner Join:**

* In MongoDB, an inner join operation combines matching documents from two collections based on specified conditions and resulting in a set of documents that intersect.
* This process involves matching documents from the primary collection with those from the secondary collection using a common field. The result includes only the documents where a match is found in both collections.

db.books.aggregate ([  
 { $lookup: {  
 from: "authors",  
 localField: "authorId",  
 foreignField: "\_id",  
 as: "author"  
 }

}

]);

db.books.aggregate ([  
 {  
 $lookup: {  
 from: "authors",  
 localField: "authorId",  
 foreignField: "\_id",  
 as: "author"  
 }  
 },  
 {  
 $match: {  
 "author": { $ne: [] }  
 }  
 }  
]);

**Explanation:**

* In the above Query, It basically brings together information from the books and authors collections by finding matches based on the authorId field in the books collection and the \_id field in the authors collection.
* This way, you get details about books along with their respective authors. The output only includes books with valid authors. If there's a book without a matching author, it simply won't show up in the result.
* In this case, there are 4 documents in the books collection, but some of them does not have a corresponding author in the authors collection. As a result, only 3 documents are returned in the output.

**4) Full Join:**

* In MongoDB, a full join is a database operation that combines data from two collections using a related field and bringing together all documents from both collections, regardless of matching related fields.
* If there's no match for a document in one collection with another, both documents will still appear in the output.

db.books.aggregate([  
 {  
 $lookup: {  
 from: "authors",  
 localField: "authorId",  
 foreignField: "\_id",  
 as: "author"  
 }  
 },  
 {  
 $unionWith: {  
 coll: "authors",  
 pipeline: [  
 {  
 $lookup: {  
 from: "books",  
 localField: "bookId",  
 foreignField: "\_id",  
 as: "book"  
 }  
 }  
 ]  
 }  
 }  
]);

**Explanation:**

**1)** **$lookup**:

* Joins the books collection with the authors collection by matching authorId in books with \_id in authors. This creates a new field called author containing matched documents.
* Results in documents from books enriched with corresponding author details in an array under the field author.

**2)** **$unionWith**:

* Combines the books collection with the authors collection after performing another $lookup to join authors with books based on bookId and \_id.
* Adds data from the authors collection to the pipeline.
* For each document in the authors collection, it performs a $lookup with the books collection, matching bookId in authors to \_id in books.
* The matched books are added as an array under the field book.
* In the output, there are a total of 8 documents, 4 from the books collection and another 4 from the authors collections.

**Note:**

1. This query is useful when you need a comprehensive view of both books and authors collections, showing relationships in both directions:

* Books with their authors.
* Authors with their books.

1. However, this query can become complex and resource-intensive for large datasets, so ensure indexes on authorId in books and \_id in authors (and vice versa).