**DBMS Practical No: 10**

**# Write the MongoDB Queries using aggregation and**

**indexing withsuitable example using MongoDB.**

**Step 1: Connect to MongoDB**

First, make sure you have MongoDB installed and running. You can connect to MongoDB using the mongo shell or a MongoDB driver in your preferred programming language.

**Step 2: Create a Sample Collection**

Assuming you have a MongoDB database, create a sample collection called sales. You can do this with the following command in the mongo shell:

**javascript**

**db.createCollection("sales")**

**Step 3: Insert Sample Data**

Insert some sample data into the sales collection. Here's an example of how to insert a few documents:

**javascript**

**db.sales.insertMany([**

**{ date: ISODate("2023-01-15"), product: "A", quantity: 10, revenue: 500 },**

**{ date: ISODate("2023-01-20"), product: "B", quantity: 5, revenue: 300 },**

**{ date: ISODate("2023-01-25"), product: "A", quantity: 7, revenue: 350 },**

**{ date: ISODate("2023-02-05"), product: "C", quantity: 3, revenue: 150 },**

**]);**

These sample documents represent sales transactions with date, product, quantity, and revenue information.

**Step 4: Perform Aggregation**

Now, let's perform aggregation to find the total revenue for each product. This is a step-by-step explanation of the aggregation query:

**javascript**

**db.sales.aggregate([**

**{**

**$group: {**

**\_id: "$product",**

**totalRevenue: { $sum: "$revenue" }**

**}**

**}**

**]);**

$group is a stage that groups the documents in the collection based on a specific field, in this case, the product field.

totalRevenue is a new field we're creating within each group using the $sum operator to calculate the sum of the revenue field.

The result will be a list of products and their total revenues.

Run this aggregation query in the mongo shell, and you'll get the aggregated results.

**Step 5: Create an Index**

To improve query performance, let's create an index on the date field. Here's how you can do it:

**javascript**

**db.sales.createIndex({ date: 1 })**

createIndex is used to create an ascending index (1) on the date field.

This index will help improve the performance of queries that filter or sort by the date field.

**Step 6: Query Using the Index**

Now, let's query the data using the created index. This query retrieves sales transactions for a specific date range (from January 1, 2023, to January 31, 2023):

**javascript**

**db.sales.find({ date: { $gte: ISODate("2023-01-01"), $lt: ISODate("2023-02-01") });**

This query uses the $gte and $lt operators to specify the date range.

Thanks to the index on the date field, MongoDB will efficiently retrieve the relevant documents.