NoSQL Database

Table of Contents

[About NoSQL Database 3](#_Toc152691754)

[Installation process 4](#_Toc152691755)

[Prerequisites 4](#_Toc152691756)

[Installing MongoDB 7.0 Community Edition 5](#_Toc152691757)

[Data Modelling 6](#_Toc152691758)

[Data Migration 6](#_Toc152691759)

[Product Listing and Search 7](#_Toc152691760)

[Scalability 7](#_Toc152691761)

[Data consistency and integrity 8](#_Toc152691762)

[NoSQL Database selection 8](#_Toc152691763)

[Scalability and performance testing 8](#_Toc152691764)

[Advantage of you specific NOSQL database over RDMS approach 9](#_Toc152691765)

[References 11](#_Toc152691766)

[Appendix 12](#_Toc152691767)

# About NoSQL Database

NoSQL databases, as the name suggests, are databases that do not rely on the traditional SQL (Structured Query Language) for querying and managing data. These databases are designed to handle large volumes of unstructured or semi-structured data and offer flexible schema designs. NoSQL databases are often used in scenarios where traditional relational databases may face limitations in terms of scalability, flexibility, or performance.

MongoDB, specifically, is a popular NoSQL database that falls under the category of document-oriented databases. Here are some key characteristics and reasons why MongoDB is chosen in various scenarios:

1. Document-Oriented Model:

MongoDB stores data in BSON (Binary JSON) documents. Each document is a JSON-like object that can contain key-value pairs, arrays, and nested documents. This document-oriented approach is flexible and allows for a dynamic schema, making it suitable for handling diverse data types.

1. Scalability:

MongoDB is designed to scale horizontally, allowing for the distribution of data across multiple servers. This enables MongoDB to handle large amounts of data and high traffic volumes efficiently.

1. Flexibility in Schema Design:

Unlike traditional relational databases with rigid schemas, MongoDB allows for dynamic schema design. This means that fields in a document can vary, making it easy to evolve the data model as requirements change over time.

1. Rich Query Language:

MongoDB provides a powerful query language that supports a wide range of queries, including field-level queries, range queries, and geospatial queries. This makes it suitable for various types of applications.

1. Indexing and Aggregation Framework:

MongoDB supports indexing to improve query performance. Additionally, it has a flexible and expressive Aggregation Framework that allows for complex data transformations and analytics.

1. Community Support and Ecosystem:

MongoDB has a large and active community, providing support, tutorials, and resources. It also has a rich ecosystem of tools and libraries that integrate with various programming languages and frameworks.

1. Document-Level Transactions:

MongoDB supports ACID-compliant transactions at the document level. This ensures data consistency and integrity, making it suitable for applications with transactional requirements.

1. Open Source and Cross-Platform:

MongoDB is open-source, which means it is free to use and has a vibrant community of contributors. It is also cross-platform, supporting various operating systems.

# Installation process

Install MongoDB Community Edition on macOS systems from MongoDB archives.

### Prerequisites

Ensure your system meets each of the following prerequisites. You only need to perform each prerequisite step once on your system. If you have already performed the prerequisite steps as part of an earlier MongoDB installation using Homebrew, you can skip to the installation process

#### **Install Xcode Command-Line Tools**

Homebrew requires the Xcode command-line tools from Apple's Xcode.

* Install the Xcode command-line tools by running the following command in your macOS Terminal:

xcode-select –install

macOS does not include the Homebrew brew package by default.

* Install brew using the official from brew home page.

/bin/bash -c "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh>)"

### Installing MongoDB 7.0 Community Edition

Follow these steps to install MongoDB Community Edition using Homebrew's brew package manager. Be sure that you have followed the installation prerequisites above before proceeding.

1. Using the above link you can download the official Homebrew formula for MongoDB and the Database Tools, by running the following command in your macOS Terminal:

brew tap mongodb/brew

1. If you have already done this for a previous installation of MongoDB, you can skip this step.
2. To update Homebrew and all existing formulae:

brew update

1. To install MongoDB, run the following command in your macOS Terminal application:

brew install [mongodb-community@7.0](mailto:mongodb-community@7.0)

1. Next start the Mongo DB community server.

brew services start mongodb-community

1. To stop the sever we can use.

brew services Stop mongodb-community

# Data Modelling

For customer and seller interactions, item listings, and orders, a flexible and scalable data model is crucial. Consider the following:

Entities:

Customers and Sellers: Capture relevant information such as username, email, and account details.

Items: Include details like item name, description, price, and seller information.

Orders: Include information about the items ordered, quantity, total price, and customer details.

Relationships:

Establish relationships between customers and orders, sellers and items, and items and orders. This could involve using unique identifiers to link entities.

Denormalization:

While NoSQL databases allow for flexible schema designs, consider denormalization to improve query performance. Embed relevant information within documents to reduce the need for joins.

A screenshot of a computer

Description automatically generated

# Data Migration

Data migration from MySQL to MongoDB using a JSON file is a common approach, and it allows you to transfer your data while preserving its structure. Here are the general steps involved in this process:

1. Export Data from MySQL to JSON:

Use MySQL Query:

Write a SQL query to select the data you want to migrate from MySQL. Use the SELECT INTO OUTFILE statement to export the result set to a JSON file.

SELECT \* FROM your\_table INTO OUTFILE '/path/to/exported\_data.json'

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n';

For your\_table we can place Customers, Seller, Orders etc.

Transform JSON File (if needed):

Inspect JSON Structure:

Examine the exported JSON file to understand its structure and make note of any adjustments needed for MongoDB compatibility.

Transform JSON (if necessary):

Depending on the structure, you might need to modify the JSON file. MongoDB prefers a flexible schema, but you may need to adjust certain aspects.

Import Data into MongoDB:

Use mongoimport Tool:

MongoDB provides the mongoimport tool to import data from various formats, including JSON.

Or we can directly import the data by downloading using interface and upload that data through interface.

# Product Listing and Search

Efficient product listing and search functionalities are essential for a seamless user experience:

Indexing:

Utilize indexing in your NoSQL database (e.g., MongoDB) to accelerate search queries. Index key fields like item name, seller, and category.

Full-Text Search:

If supported by your chosen NoSQL database, implement full-text search for more advanced and flexible search capabilities.

Caching:

Implement a caching layer, especially for frequently accessed or searched items, to enhance response times and reduce the load on the database.

# Scalability

Sharding:

Distribute data across multiple nodes using sharding to achieve horizontal scalability. This ensures that the database can handle growing amounts of data.

Horizontal Scaling:

Plan for horizontal scaling by adding more servers to the system. NoSQL databases like MongoDB are designed to scale horizontally.

# Data consistency and integrity

Consistency Model:

We Choose a consistency model that aligns with your application's requirements. NoSQL databases often provide options for eventual consistency or strong consistency.

Validation:

Implemented proper validation at the application level to ensure data integrity before interacting with the NoSQL database.

Atomic Operations:

Leverage transactions or atomic operations provided by the NoSQL database to maintain data consistency during complex operations.

# NoSQL Database selection

We Consider MongoDB as a potential NoSQL database, but evaluate other options based on specific requirements:

MongoDB:

MongoDB is a widely used NoSQL database with features such as horizontal scalability, flexible schema design, and robust query capabilities.

Alternative Databases:

We can also explore other NoSQL databases such as Cassandra, Couchbase, or DynamoDB based on specific requirements such as performance, scalability, and ease of use.

# Scalability and performance testing

In this phase of our project, we're focusing on making sure our system can handle growth and performs well under different situations.

Testing Your System:

We're going to put our system to the test to see how it handles various scenarios, especially when the number of users and data increases. This helps us identify any points where our system might slow down or struggle as more people use it.

# Advantage of you specific NOSQL database over RDMS approach

Query 1: Flexible Schema

Advantage: NoSQL databases allow for a flexible schema, accommodating varying structures for different documents.

# Insert a document with a new field (RDMS might require altering the table)

db['customer'].insert\_one({

'name': 'Alice',

'email': 'alice@email.com',

'phone': 9876543210,

'address': '123 Main Street',

'city': 'Cityville',

'country': 'Countryland',

'loyalty\_points': 100

})

Query 2: Embedded Documents

Advantage: NoSQL databases support embedded documents, reducing the need for complex joins in queries.

# Insert an order with embedded product details

db['orders'].insert\_one({

'custID': 'CID0000002',

'products': [

{'prodID': 'PROD000002', 'quantity': 2},

{'prodID': 'PROD000003', 'quantity': 1}

],

'date': datetime(2023, 2, 5),

'status': 'Pending'

})

Query 3: Scalability

Advantage: NoSQL databases excel in horizontal scalability, distributing data across multiple nodes easily.

# Sharding the 'orders' collection for horizontal scalability

db['orders'].create\_index('custID')

db['orders'].create\_index('date')

Query 4: Aggregation Pipeline

Advantage: MongoDB's Aggregation Pipeline allows for powerful and flexible data transformations.

# Calculate total expenditure per customer using the aggregation pipeline

total\_expenditure\_per\_customer = db['orders'].aggregate([

{"$group": {"\_id": "$custID", "total\_expenditure": {"$sum": {"$multiply": ["$sell\_price", "$quantity"]}}}}

])

Query 5: Indexing

Advantage: MongoDB provides efficient indexing strategies for improved query performance.

# Create an index on the 'name' field in the 'customer' collection

db['customer'].create\_index('name')

# Query using the index for faster retrieval

result = db['customer'].find({'name': 'Bob'}).explain()['executionStats']

Query 6: NoJoins

Advantage: NoSQL databases often avoid complex joins, improving query performance.

# Retrieve customer details with orders using the $lookup stage

customer\_orders = db['customer'].aggregate([

{"$lookup": {

"from": "orders",

"localField": "custID",

"foreignField": "custID",

"as": "orders"

}},

{"$project": {

"name": 1,

"order\_count": {"$size": "$orders"}

}}

])

# Print everything

print("Total Expenditure per Customer:")

for record in total\_expenditure\_per\_customer:

print(record)

print("\nIndexing Result:")

print(result)

print("\nCustomer Orders with Order Count:")

for record in customer\_orders:

print(record)

**Output**

Total Expenditure per Customer:

{'\_id': 'CID0000001', 'total\_expenditure': 1800}

{'\_id': 'CID0000002', 'total\_expenditure': 0}

Indexing Result:

{'executionSuccess': True, 'nReturned': 0, 'executionTimeMillis': 8, 'totalKeysExamined': 0, 'totalDocsExamined': 0, 'executionStages': {'stage': 'nlj', 'planNodeId': 2, 'nReturned': 0, 'executionTimeMillisEstimate': 0, 'opens': 1, 'closes': 1, 'saveState': 0, 'restoreState': 0, 'isEOF': 1, 'totalDocsExamined': 0, 'totalKeysExamined': 0, 'collectionScans': 0, 'collectionSeeks': 0, 'indexScans': 0, 'indexSeeks': 1, 'indexesUsed': ['name\_1'], 'innerOpens': 0, 'innerCloses': 0, 'outerProjects': [], 'outerCorrelated': [4, 7, 8, 9, 10], 'outerStage': {'stage': 'cfilter', 'planNodeId': 1, 'nReturned': 0, 'executionTimeMillisEstimate': 0, 'opens': 1, 'closes': 1, 'saveState': 0, 'restoreState': 0, 'isEOF': 1, 'numTested': 1, 'filter': '(exists(s5) && exists(s6)) ', 'inputStage': {'stage': 'ixseek', 'planNodeId': 1, 'nReturned': 0, 'executionTimeMillisEstimate': 0, 'opens': 1, 'closes': 1, 'saveState': 0, 'restoreState': 0, 'isEOF': 1, 'indexName': 'name\_1', 'keysExamined': 0, 'seeks': 1, 'numReads': 1, 'indexKeySlot': 9, 'recordIdSlot': 4, 'snapshotIdSlot': 7, 'indexIdentSlot': 8, 'outputSlots': [], 'indexKeysToInclude': '00000000000000000000000000000000', 'seekKeyLow': 's5 ', 'seekKeyHigh': 's6 '}}, 'innerStage': {'stage': 'limit', 'planNodeId': 2, 'nReturned': 0, 'executionTimeMillisEstimate': 0, 'opens': 0, 'closes': 0, 'saveState': 0, 'restoreState': 0, 'isEOF': 0, 'limit': 1, 'inputStage': {'stage': 'seek', 'planNodeId': 2, 'nReturned': 0, 'executionTimeMillisEstimate': 0, 'opens': 0, 'closes': 0, 'saveState': 0, 'restoreState': 0, 'isEOF': 0, 'numReads': 0, 'recordSlot': 11, 'recordIdSlot': 12, 'seekKeySlot': 4, 'snapshotIdSlot': 7, 'indexIdentSlot': 8, 'indexKeySlot': 9, 'indexKeyPatternSlot': 10, 'fields': [], 'outputSlots': []}}}, 'allPlansExecution': []}

Customer Orders with Order Count:

{'\_id': 'CID0000001', 'name': 'John Doe', 'order\_count': 0}

{'\_id': ObjectId('656faada8c18eb3b30071547'), 'name': 'Alice', 'order\_count': 0}

{'\_id': ObjectId('656fab0a06e00b820153bd89'), 'name': 'Alice', 'order\_count': 0}

{'\_id': ObjectId('656fab2075960fdbbb15b6c7'), 'name': 'Alice', 'order\_count': 0}

{'\_id': ObjectId('656fab3e710a01f2855a31c4'), 'name': 'Alice', 'order\_count': 0}

These queries showcase some of the advantages of using MongoDB as a NoSQL database, emphasizing flexibility, scalability, and performance in various scenarios. Keep in mind that the appropriateness of a NoSQL or RDMS approach depends on the specific requirements and nature of your application.

# References

1. <https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/#std-label-install-mdb-community-macos>
2. <https://brew.sh/#install>
3. <https://www.mongodb.com/docs/manual/reference/program/mongod/#mongodb-binary-bin.mongod>
4. <https://redis.com/blog/nosql-data-modeling/>
5. <https://www.educba.com/nosql-data-models/>

# Appendix

A screenshot of a document

Description automatically generated

Mangodb with my database as mydb1

A screenshot of a computer

Description automatically generated

Customers data

A screenshot of a computer

Description automatically generated

Orders data

A screenshot of a computer

Description automatically generated

Product data

A screenshot of a computer

Description automatically generated

Sellers data

# from pymongo import MongoClient

# # Requires the PyMongo package.

# # https://api.mongodb.com/python/current

# client = MongoClient('mongodb://localhost:27017/')

# result = client['mydb']['customers'].aggregate([])

Connection establishment string

# Insert documents into the customer collection

db['customer'].insert\_many([

{

'\_id': 'CID0000001',

'name': 'John Doe',

'email': 'johndoe@email.com',

'phone': 1234567890,

'address': {

'area': 'Main Street',

'locality': 'Downtown',

'city': 'New York',

'state': 'New York',

'country': 'USA',

'zipcode': 10001

},

'password': '123'

},

# ... (other customer documents)

])

# Insert documents into the seller collection

db['seller'].insert\_many([

{

'\_id': 'SELL000001',

'name': 'Electronics Store',

'email': 'electronics@store.com',

'phone': 1234567890,

'address': {

'area': 'Downtown',

'locality': 'City Center',

'city': 'New York',

'state': 'New York',

'country': 'USA',

'zipcode': 10001

},

'password': 'sellerpass',

'status': 'active'

},

# ... (other seller documents)

# Insert documents into the product collection

db['product'].insert\_many([

{

'\_id': 'PROD000001',

'name': 'Mobile Phone X',

'quantity': 100,

'category': 'Electronics',

'cost\_price': 500,

'sell\_price': 600,

'description': 'Latest smartphone model',

'sellID': 'SELL000001',

'transaction\_date': datetime(2023, 1, 10)

},

# ... (other product documents)

])

# Insert documents into the orders collection

db['orders'].insert\_many([

{

'\_id': 'ORD000001',

'custID': 'CID0000001',

'prodID': 'PROD000001',

'quantity': 3,

'date': datetime(2023, 1, 15),

'cost\_price': 500,

'sell\_price': 600,

'status': 'Completed'

},

# ... (other order documents)

])

Business Quires

# MongoDB Query 1: Count of registered vendors

result1 = db['seller'].count\_documents({})

# MongoDB Query 2: Count of active and inactive sellers

result2 = db['seller'].aggregate([

{"$group": {"\_id": "$status", "count": {"$sum": 1}}}

])

# Print results

print("Query 1: Count of registered vendors =", result1)

print("Query 2: Count of active and inactive sellers =", list(result2))

# Question 1: Count of registered vendors

result1 = db['seller'].count\_documents({})

# Question 2: Count of active and inactive sellers

result2 = db['seller'].aggregate([

{"$group": {"\_id": "$status", "count": {"$sum": 1}}}

])

# Question 3: Count of total customers

result3 = db['customer'].count\_documents({})

# Question 4: Number of listings for the top seller

result4 = db['product'].aggregate([

{"$group": {"\_id": "$sellID", "count": {"$sum": 1}}},

{"$sort": {"count": -1}},

{"$limit": 1}

])

# Question 5: Number of orders for the customer with the most orders

result5 = db['orders'].aggregate([

{"$group": {"\_id": "$custID", "count": {"$sum": 1}}},

{"$sort": {"count": -1}},

{"$limit": 1}

])

# Question 6: Monthly revenue for top 5 sellers between Jan 1 and Mar 31, 2023

result6 = db['product'].aggregate([

{"$match": {"transaction\_date": {"$gte": datetime(2023, 1, 1), "$lte": datetime(2023, 3, 31)}}},

{"$group": {"\_id": {"month": {"$dateToString": {"format": "%B", "date": "$transaction\_date"}}, "sellID": "$sellID"},

"revenue": {"$sum": {"$multiply": ["$sell\_price", "$quantity"]}}}},

{"$sort": {"\_id.month": 1, "revenue": -1}},

{"$limit": 5}

])

# Question 7: Total revenue for top 5 customers in the current year

result7 = db['orders'].aggregate([

{"$match": {"date": {"$gte": datetime(datetime.now().year, 1, 1)}}},

{"$group": {"\_id": "$custID", "total\_revenue": {"$sum": {"$multiply": ["$sell\_price", "$quantity"]}}}},

{"$sort": {"total\_revenue": -1}},

{"$limit": 5}

])

# Question 8: Total revenue for the current month and the same month in the previous year

current\_month\_revenue = db['orders'].aggregate([

{"$match": {"date": {"$gte": datetime(datetime.now().year, datetime.now().month, 1)}}},

{"$group": {"\_id": None, "current\_month\_revenue": {"$sum": {"$multiply": ["$sell\_price", "$quantity"]}}}}

])

previous\_year\_month\_revenue = db['orders'].aggregate([

{"$match": {"date": {"$gte": datetime(datetime.now().year - 1, datetime.now().month, 1) if datetime.now().month != 1 else datetime(datetime.now().year - 1, 12, 1),

"$lt": datetime(datetime.now().year - 1, datetime.now().month, 1)}}},

{"$group": {"\_id": None, "previous\_year\_month\_revenue": {"$sum": {"$multiply": ["$sell\_price", "$quantity"]}}}}

])

# Question 9: Maximum value of (sell\_price \* quantity) in orders

result9 = db['orders'].aggregate([

{"$group": {"\_id": None, "max\_value": {"$max": {"$multiply": ["$sell\_price", "$quantity"]}}}}

])

# Question 10: Total orders for customer 'CID0000003'

result10 = db['orders'].count\_documents({"custID": "CID0000003"})

# Question 11: Total spend for customer 'CID0000003' in the current year

result11 = db['orders'].aggregate([

{"$match": {"custID": "CID0000003", "date": {"$gte": datetime(datetime.now().year, 1, 1)}}},

{"$group": {"\_id": None, "total\_spend": {"$sum": {"$multiply": ["$sell\_price", "$quantity"]}}}}

])

# Question 13: Database size in MB

try:

result13 = db.command({"dbStats": 1, "scale": 1024 \* 1024})

except OperationFailure:

# The command may not be available, especially on older MongoDB versions

result13 = {"mydb1": "N/A"}

# Additional Query: Retrieve all documents from the 'customer' collection

all\_customers = list(db['customer'].find())

# Additional Query: Retrieve customer order details

customer\_orders = list(db['customer'].aggregate([

{"$lookup": {

"from": "orders",

"localField": "custID",

"foreignField": "custID",

"as": "orders"

}},

{"$unwind": "$orders"},

{"$project": {

"name": 1,

"order\_date": "$orders.date",

"expenditure": {"$multiply": ["$orders.sell\_price", "$orders.quantity"]}

}},

{"$sort": {"name": 1, "order\_date": 1}}

]))

# Print results

print("Question 1: Count of registered vendors =", result1)

print("Question 2: Count of active and inactive sellers =", list(result2))

print("Question 3: Count of total customers =", result3)

print("Question 4: Number of listings for the top seller =", list(result4))

print("Question 5: Number of orders for the customer with the most orders =", list(result5))

print("Question 6: Monthly revenue for top 5 sellers between Jan 1 and Mar 31, 2023 =", list(result6))

print("Question 7: Total revenue for top 5 customers in the current year =", list(result7))

print("Question 8: Total revenue for the current month and the same month in the previous year =")

print("Current Month Revenue:", list(current\_month\_revenue), "Previous Year Month Revenue:", list(previous\_year\_month\_revenue))

print("Question 9: Maximum value of (sell\_price \* quantity) in orders =", list(result9))

print("Question 10: Total orders for customer 'CID0000003' =", result10)

print("Question 11: Total spend for customer 'CID0000003' in the current year =", list(result11))

print("Question 13: Database size in MB =", result13)

print("Additional Query: Retrieve all documents from the 'customer' collection =", all\_customers)

# Close the MongoDB connection

client.close()

output

A screen shot of a computer screen

Description automatically generated











