International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 04 | Apr 2022 www.irjet.net p-ISSN: 2395-0072

Democratization of NOSQL Document-Database over Relational Database Comparative Case Study: Cloud Kitchen

Kaushik Ganguly[1], Souvik Ghosh[2], Aditya Tripathi[3]

[1,2,3]*MTech Candidates, Data Science & Engineering, Birla Institute of Technology and Science, Pilani, India*

Abstract - *The online food delivery market revenue is expected to reach $140 Million+ worldwide. As a result, the new model of dining has already taken a paradigm shift to capture this extravagant market. Consumers are now shifting from dining to delivery/take-away model. Here takes birth the concept of Cloud Kitchen – orders placed, meals cooked, packed and then dispatched immediately to the respective delivery locations by the assigned fleet, E.g. Swiggy, Zomato, UberEats. Apart from Lower Real Estate costs, saving overhead costs, convenience, etc., one should also consider what it takes to setup a proper datastore infrastructure to cater to the need of these cloud kitchens. With respect to that need, we intend to perform a comparative analysis on database(s) (NOSQL Vs. SQL) one needs to choose for faster data access, order placement, bill generation, delivery notification settings, standard feedback loops and customer sentiment analysis for better service. We have done a detailed analysis based on expected/possible database CRUD operations that could get performed throughout one order cycle – right from placement till delivery and post-delivery customer feedbacks and have given our proposal of database choice based on that mode and complexity of operation.*

Key Words: **Infrastructure, Cloud Kitchen, Delivery, Feedback Loop, Database Operations, NOSQL VS SQL**

1. INTRODUCTION

Currently, India has got 3500+ cloud kitchens serving dishes from all over the world. With increase in demand, new cloud kitchens and SMB's – big or small are being opened almost every day resulting in intense competition to attract foodies. During June 2020 when the COVID-19 pandemic was at it's peak in India, with restaurants being shut for months due to the nation-wide lockdown imposed by India in late March, the demand for food delivery has surged and revenue share has jumped from around 18% pre-COVID-19 to almost 100% by May 2020.

This is the perfect time to go for the big break for existing food delivery business chains or entrepreneurs by innovating with Data and Data Store infrastructures to cater to this ever-increasing demand for authentic cuisines. Data driven decision making infrastructures with faster and smarter setups for data storage and utilization are going to be the new trend setters.

As the next step, maximum data companies and tech wizards would want to invest into the data platform that will help them to come up with a recommendation engine solution that will provide the cloud kitchens suggestions to the registered customers. For that purpose, the first step would be data acquisition, exploration and using it in remaining stages of the data processing.

From Data Architecture perspective, the crucial question we must ask is whether to use a SQL or NoSQL database for application. SQL has had a large lead over the non-relational alternatives for decades, but NoSQL is quickly closing the gap with popular databases such as MongoDB, Redis, and Cassandra. SQL still holds 60% with rising demand for systems such as PostgreSQL. They are taking help of this report further to decide upon the database that they can harvest to fulfill their requirements.

On this initial prototyping stage, we need to explore the various SQL/ NoSQL database options available for the storage and querying the customer data mainly based on performance.

2. PROPOSED APPROACH

The first step that we must perform is to explore the various, commonly used NoSQL/SQL databases such as

- MongoDB
- MSSQL/MySQL

Our simulation will involve loading the given data in the above-mentioned databases and do a performance comparison against the below mentioned database

CRUD operations. Based on this result, we will recommend the database option and then the vendor can deploy the application infrastructure in their data pipeline.

2.1 PERFORMANCE CONSIDERATIONS

The database operations that need to be considered are:

- **Write** - If a given record / key-value pair is not found in the database storage, then the pair is added to the storage. Otherwise, it updates the value for the given key in the storage. This operation therefore combines

"Create" and "Update" operations of the CRUD model.

- **Read** - Reads the value corresponding to a given record / key from storage. This is the same as the Read operation of the CRUD (Create, Read, Update, and Delete) model commonly used to describe database operations.
- **Delete** - This deletes the record (i.e. key-value pair) corresponding to a given key from the key-value pair storage. This is the same as the Delete operation of the CRUD model.
- **GroupBy and OrderBy** – This operation involves grouping of the records and ordering by certain criteria.

2.2 EXPECTED OUTCOME

At the end of this simulation exercise, we will generate a report clearly giving the recommendation of database choice as per the Cloud-Kitchen business model.

The outcome will involve the following:

- The exact queries used for each of the operation with respect to each database.
- Various performance parameters considered (with reasons) while executing the queries and their analysis.
- Visualizations based on the performance parameters used for the queries and their interpretation.
- Tabular comparison summary of the prototyping exercise.
- Finally - The Recommendation

3. METHODOLOGY

3.1 TECHNOLOGY STACK USED

- Mongo DB 4.4 (NO-SQL DB)
  - Mongo DB Atlas
  - Mongo DB Compass 1.29.5
  - Mongo DB Command-Line Shell
- MSSQL (SQL DB)
  - MSSQL on SQLite

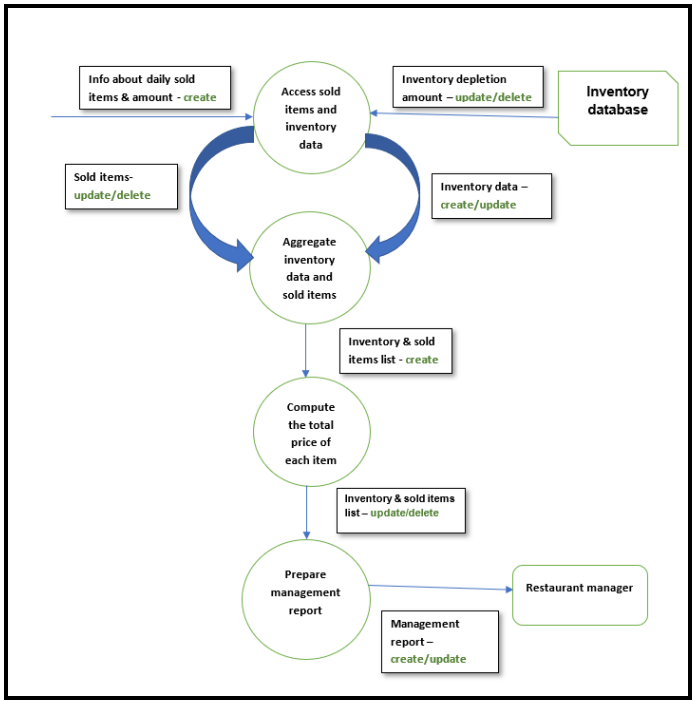
3.2 ENVIRONMENT USED

- Cloud Provider - AWS, Region - N. Virginia (us-east-1)
- Cluster Tier - M10 (2 GB RAM, 10 GB Storage) 1,000 IOPS, Encrypted, Auto-expand Storage
- Version - MongoDB 4.4
- MS SQL Server Express – 2019

3.3 DATASOURCE AND LOADING

We have considered Restaurant Recommendation customer data from Kaggle(https://www.kaggle.com) for this study. Data loaded from the csv file using MongoDB compass GUI and connected the same compass application with MongoDB cloud Atlas.

3.4 Workflow Operations - Cloud Kitchen



4. EXPERIMENTAL RESULT

1. <u>Write Operation: MongoDB query statement:</u>

If a mentioned record (customer_id: 'TCHWP7BT') was not found in the DB, then the item is added to the DB with the update command as below. This is create operation within the CRUD model.

```
Atlas atlas-5nkwcu-shard-0 [primary] customerDB>
db.customer_data.update({ customer_id: 'TCHWP7BT'},{$set:{
gender: 'Female', dob: '1988', status: '121', verified: '1',
language: 'EN', created_at: '2018-05-07 23:57:21', updated_at:
'2018-05-07 23:57:21' }},{upsert:true})
```

Next again when we run the below update query for the same record (customer_id: 'TCHWP7BT'), as the record has already existed it got updated this time, The gender value got modified from Female to Male. This is an Update operation in the CRUD model.

```
Atlas atlas-5nkwcu-shard-0 [primary] customerDB>
db.customer_data.update({customer_id: 'TCHWP7BT'},{$set:{
gender: 'Male', dob: '1988', status: '121', verified: '1',
language: 'EN', created_at: '2018-05-07 23:57:21',
updated_at: '2018-05-07 23:57:21' }},{upsert:true})
```

Write Operation: MSSQL equivalent statement:

The csv data has been uploaded to the table train_full with MSSQL DB. As the mentioned customer id was not available in the table, a new record has been created with the below update query statement.

```
if EXISTS (SELECT * FROM train_full WHERE customer_id LIKE
'TCHWP7B3T') BEGIN UPDATE train_full set status_x = 10 ,
updated_at_x = SYSDATETIME(), updated_at_y =
SYSDATETIME() where customer_id LIKE  'TCHWP7B3T'
END
ELSE
BEGIN
INSERT INTO train_full
(customer_id,gender,updated_at_x,updated_at_y)
VALUES('TCHWP7B3T','FEMALE',SYSDATETIME(),SYSDATETI
ME())
END;
```

Query to find the customer record:

```
SELECT * FROM train_full WHERE customer_id LIKE
'TCHWP7B3T'
```

While again we run the same query statement again it just updates the record as that record already existed.

```
if EXISTS (SELECT * FROM train_full WHERE customer_id LIKE
'TCHWP7B3T') BEGIN UPDATE train_full set status_x = 10 ,
updated_at_x = SYSDATETIME(), updated_at_y =
SYSDATETIME() where customer_id LIKE  'TCHWP7B3T'
END
ELSE
BEGIN
INSERT INTO train_full
(customer_id,gender,updated_at_x,updated_at_y)
VALUES('TCHWP7B3T','FEMALE',SYSDATETIME(),SYSD
ATETIME())
END;
```

```
SELECT * FROM train_full WHERE customer_id LIKE
'TCHWP7B3T'
```

2. Read Operation: MongoDB query statement:

We can easily read any record using the find command using any unique attribute value as customer_id is a unique attribute for this data.

```
Atlas atlas-5nkwcu-shard-0 [primary] customerDB>
db.customer_data.find({customer_id:'TCHWPBT'}).pretty()
```

Read Operation: MSSQL equivalent statement:

Where in MSMSQL we can run the below command to the same job.

```
SELECT * FROM train_full WHERE customer_id LIKE
'TCHWPBT'
```

3. Delete Operation: MongoDB query statement:

To delete any particular record, we can use remove command in MongoDB as below.

```
Atlas atlas-5nkwcu-shard-0 [primary] customerDB>
db.customer_data.remove({_id:
ObjectId("61bdbc13d515988b3aba6ab3")}
```

Also, we can use *deleteOne* command for deleting any particular record as below:

```
Atlas atlas-5nkwcu-shard-0 [primary] customerDB>
db.customer_data.deleteOne({_id:
ObjectId("61bdbc13d515988b3aba6ab7")})
```

We can use *deleteMany* command to delete multiple records same time as below. We have deleted 300 records for the customer id ''TCHWPBT'.

```
Atlas atlas-5nkwcu-shard-0 [primary] customerDB>
db.customer_data.deleteMany({customer_id:'TCHWPBT'})
```

Delete Operation: MSSQL equivalent statement:

We can run the delete command to do the same job.

```
DELETE FROM train_full WHERE customer_id like
'TCHWP7B3T';
```

```
SELECT * FROM train_full WHERE customer_id LIKE
'TCHWP7B3T';
```

4. OrderBy Operation: MongoDB query statement:

In MongoDB, *OrderBy* operation can be done using sort() command. Items are sorted/OrderBy created_at filed value ascending order as below.

```
Atlas atlas-5nkwcu-shard-0 [primary] customerDB>
db.customer_data.find().sort({created_at:1}).pretty()
```

To sort/OrderBY with descending value for created_at_x field we can use -1.

```
Atlas atlas-5nkwcu-shard-0 [primary] customerDB>
db.customer_data.find().sort({created_at_x:-1}).pretty()
```

OrderBy Operation MSSQL equivalent statement:
Same operation can be done easily using below select query statement in MSSQL

```
SELECT * FROM train_full ORDER BY created_at_x;
```

5. GroupBY Operation: MongoDB query statement

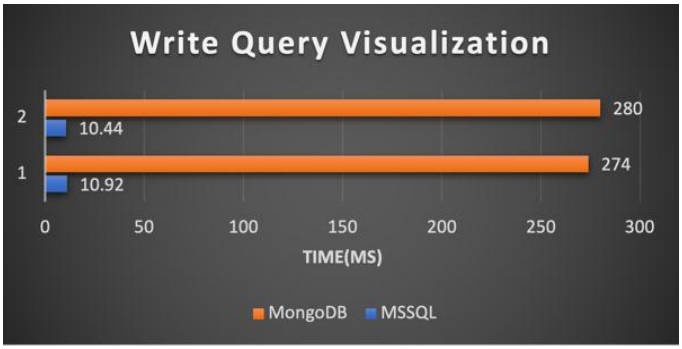
GroupBy operation in MongoDB can be done using aggregate function as below. Below queries were used to find average preparation time for vendor tag and group by vendor_tag_name and order by preparation time.

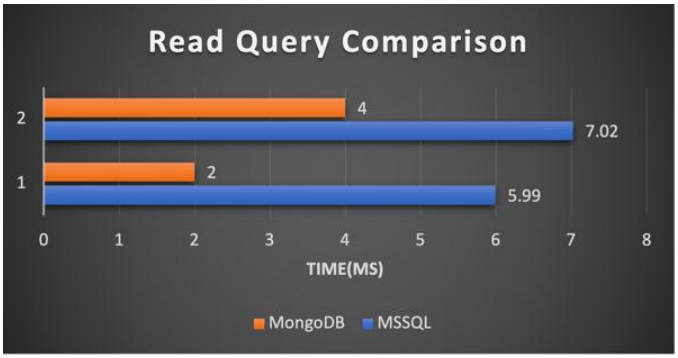
```
Atlas atlas-5nkwcu-shard-0 [primary] customerDB>
db.train_full.aggregate([{$group:{_id:"$vendor_tag_name",avg
PrepTime:{$avg:"$prepration_time"}}}])
```

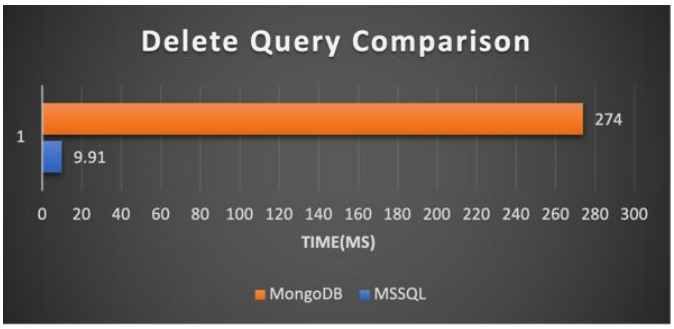
GroupBY Operation: MSSQL equivalent statement:

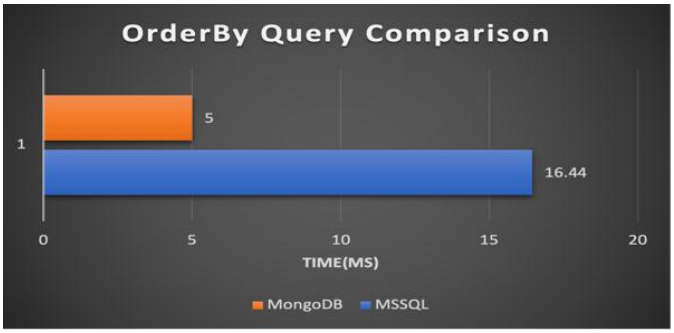
```
SELECT vendor_tag_name, AVG (prepration_time) AS
Prepration_Time FROM train_full GROUP BY vendor_tag_name
```

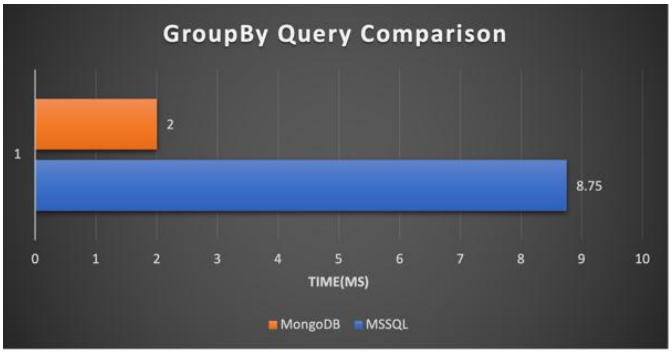
4.1 QUERY TIME VISUALIZATION – MONGODB VS MSSQL











4.2. DATA MODEL COMPARISON – MONGODB VS MSSQL

- **RDBMS Data Model:** Relational Database Management Systems (RDBMSs) have been around for ages MySQL is the most popular among them Data stored in tables Schema-based, i.e., structured tables Each row (data item) in a table has a primary key that is unique within that

table Queried using SQL (Structured Query Language) Supports joins.

- **Key-value/NoSQL Data Model:** NoSQL = "Not Only SQL" Necessary API operations: get(key) and put(key, value) Tables "Column families" in Cassandra, "Table" in HBase, "Collection" in MongoDB Like RDBMS tables, but ... May be unstructured: May not have schemas Some columns may be missing from some rows don't always support joins or have foreign keys can have index tables, just like RDBMS.

4.3 PERFORMANCE PARAMETERS COMPARISON

MongoDB	MSSQL
1. Server Status: This parameter is used in MongoDB performance monitoring to check the server status. This helps us check the details of the connections running on the database server on that instance/shard. ***db.serverStatus()***	**1. CPU and Memory Usage:** MIN_MEMORY_PERCENT and MAX_MEMORY_PERCENT parameters signify the amount of memory assigned per resource pool.
2. Locks: This is used along with the server-status command to check all the lock status in the DB. *db.***serverStatus().connection**	**2. Disc I/O Operation:** Used to control the physical I/O operation per resource pool.
3. Connection: To check the number of connections from the DB server which are currently running. ***top*** **This is also used to check the memory and CPU utilization.*	**3. Disc queue length:** Average number of read and write requests per Disc.
4. Create Index: To create index on a specific field for optimizing query performance. ***db.customer_data.createIndex({"id":1},{unique:true})***	**4. Memory Pages/Sec:** Helps to identify the faults causing system-wide delays.
5. Field Name: This is the name of the field which is used to create the index to improve the performance of the query. ***db.customer_data.createIndex({"id":1},{unique:true})***	**5. Indexing:** Used for faster data retrieval.
6. Name of the collection: Collection name based on which we must create the index.	**6. Page Split/Sec:** While creating indexes if the page becomes full, page split occurs.
7 Find & Sort Method: These are used to retrieve the document from the collection.	
8. Monitor Replication State: Check whether replication is working properly, and the server nodes are synced with each other.	

4.4 RDBMS (MYSQL) Vs Key-value Stores (MongoDB) w.r.t Cloud Kitchen Data Model

Query	MYSQL/MSSQL	MongoDB
Create Schema	In MySQL, we can create schema or database using CREATE SCHEMA customerDB	The below command would create a new schema (if it's not already created) else switch to the existing schema use customerDB
Create Table	***CREATE TABLE customer(id INT PRIMARY KEY, age INT, name VARCHAR(100), city VARCHAR(100));***	No specific create script/query required
Primary Key	Explicitly mentioned (as part of the create)	By default, assigned by the server - which is named as `_id`. For a user-defined key - we can create Unique Index, and use the same when data is accessed
Creating Index	MySQL by default creates an index on the PRIMARY KEY defined in	***db.customer_data.createIndex({"id":1},{unique:true})***

	the table creation script. For additional Indexes, CREATE INDEX command can be used. ***CREATE INDEX index_name on table_name(column_name(s));***	
Inserting Record	If not exist it will create a new record else, it will update an existing one. ***if EXISTS (SELECT * FROM train_full WHERE customer_id LIKE 'TCHWP7B3T') BEGIN UPDATE train_full set status_x = 10 , updated_at_x = SYSDATETIME(), updated_at_y = SYSDATETIME() where customer_id LIKE 'TCHWP7B3T' END ELSE BEGIN INSERT INTO train_full (customer_id, gender,updated_at_x,updated_at_y) VALUES('TCHWP7B3T','FEMALE', SYSDATETIME(), SYSDATETIME()) END;***	If not exist it will create a new record else, it will update an existing one. ***db.customer_data.update({customer_id: 'TCHWP7BT'},{$set:{ gender: 'Female', dob: '1988', status: '121', verified: '1', language: 'EN', created_at: '2018-05-07 23:57:21', updated_at: '2018-05-07 23:57:21' }},{upsert:true})***
Query Record	***SELECT * FROM train_full WHERE customer_id LIKE 'TCHWPBT';***	***db.customer_data.find({customer_id:'TCHWPBT'}).pretty()***
Explain Query	***EXPLAIN ANALYZE SELECT * FROM train_full where customer_id LIKE 'TCHWPBT';***	***db.customer_data.find({customer_id:'TCHWPBT'}).explain()***
List Indexes	***SHOW INDEX FROM train_full;***	***db.customer_data.getIndexes()***
Delete Record	***DELETE FROM train_full WHERE customer_id like 'TCHWP7B3T';***	***db..remove({_id: ObjectId("61bdbc13d515988b3aba6ab3")}*** ***db.customerData.deleteOne({_id: ObjectId("61bb8a2d4853e8333ee41aba")})***
Delete Table	***DROP TABLE train_full***	***db.customer_data.drop()***

5. CONCLUSION/RECOMMENDATION

In our restaurant use case, depending on how complex the ordering application is and the architecture of the application, it seems:

- The benefits for row and column and table and foreign key table of MSSQL is great for some of the functionality, other parts will really benefit from the ability to store items as documents in MongoDB and some other parts as key-value pairs.
- So, for this use case, the structured information such as users, restaurants can be stored in MySQL/MSSQL and information related to menu, orders and others can be stored in NoSQL (MongoDB).

Hence, we recommend the firm to deploy both databases into their data pipeline.

Please Note: We did not test the databases for more complex operations. The database performance rankings we noted may not hold when it comes to complex operations.

REFERENCES

[1] "BIG DATA AND ANALYTICS" – Seema Acharya, Subhashini Chellappan

[2] "A performance comparison of SQL and NoSQL databases"

https://www.researchgate.net/publication/261079289

[3] "The 6 Cloud Kitchen Business Models and How They Work"-Niharika Maggo https://limetray.com/blog/cloud-kitchen-business-model/

[4] "Serving Food From the Cloud" - Naveen Sharda

https://www.toptal.com/finance/growth-strategy/cloud-kitchen

[5] "All-in-One Food Order Management System for Restaurants, Takeaways and Cloud Kitchens"

https://delivety.com/?gclid=Cj0KCQjwjN-SBhCkARIsACsrBz7uvt30CMdswkgj3hDvX9UR4gaDgIBqtwAOXZHDpFul1yA3pw6oQl0aAiedEALw_wcB

[6] "Cloud kitchens to dominate India's food-tech industry" - https://kr-asia.com/cloud-kitchens-to-dominate-indias-food-tech-industry

[7] "Why cloud kitchens need to scale up smart, not fast" - RAGHAV JOSHI

https://www.forbesindia.com/article/new-year-special-2022/why-cloud-kitchens-need-to-scale-up-smart-not-fast/72951/1

[8] "Everything You Need to Know About Cloud Kitchens"

https://www.oracle.com/in/industries/food-beverage/cloud-kitchens/

[9] "With Food Delivery Startups Backing Cloud Kitchens, Do Restaurants Stand A Chance?"- Kopal Cheema https://inc42.com/features/with-food-delivery-apps-backing-cloud-kitchens-do-restaurants-stand-a-chance/

[10] "Monitoring MongoDB performance metrics (WiredTiger)" - Jean-Mathieu Saponaro

https://www.datadoghq.com/blog/monitoring-mongodb-performance-metrics-wiredtiger/

[11] "MongoDB Performance Tuning: Everything You Need to Know"

https://stackify.com/mongodb-performance-tuning/