**Group No:** BDS Group 124

**Group members:**

* Ashwini BR - 2020fc04180@wilp.bits-pilani.ac.in
* Ashutosh Kumar - 2020fc04341@wilp.bits-pilani.ac.in
* Rahul Paul – 2020fc04350@wilp.bits-pilani.ac.in

**MongoDB vs MySQL Comparison**

Given below are the device specifications and software versions installed for testing. The testing is done on a single database instance.

**Database Versions:**

|  |  |
| --- | --- |
| **Database** | **Version** |
| MySQL | 8.0.27 |
| MongoDB | 5.0.5 |

**Device Specifications of host:Graphical user interface

Description automatically generated with low confidence**

**Data Importation and Exploratory Analysis:**

**MySQL:**

The following is steps were followed to import data into MySQL:

1. Load train\_full.csv file into a pandas dataframe
2. Perform EDA to understand the data
3. Create a Database name ‘BITS’ in MySQL workbench
4. Use the ‘to\_sql’ function of the pandas module to create table and load data into BITS database in MySQL

import pandas as pd

df = pd.read\_csv('train\_full.csv')

df.to\_sql(con="mysql://root:root1234@localhost:3306/BITS", name='RestaurantData', if\_exists='replace')

**Python libraries used:** Pymysql, mysql-connector-python, sqlalchemy, mysqlclient

**MongoDB:**

The following steps were followed to import data into MongoDB:

1. Login to MongoDB Compass
2. Create database ‘BITS’
3. Create collection ‘RestaurantData’
4. Import train\_full.csv data into the newly created collection using ‘Add data’🡪’Import file’

The following understanding was gathered by querying the imported data (train\_full.csv)

* Total number of records: **5802400**
* Unique customers in dataset: **34523**
* Unique vendors in dataset: **100**

**Expected assignment Outcomes:**

1. **The schema / structure / other representation used in each of the databases:**

**MySQL:**

MySQL Table structure after importing from Pandas dataframe:

Text

Description automatically generated

1. | restaurantdata | CREATE TABLE `restaurantdata` (
2. `index` bigint DEFAULT NULL,
3. `customer\_id` text,
4. `gender` text,
5. `status\_x` bigint DEFAULT NULL,
6. `verified\_x` bigint DEFAULT NULL,
7. `created\_at\_x` text,
8. `updated\_at\_x` text,
9. `location\_number` bigint DEFAULT NULL,
10. `location\_type` text,
11. `latitude\_x` double DEFAULT NULL,
12. `longitude\_x` double DEFAULT NULL,
13. `id` bigint DEFAULT NULL,
14. `authentication\_id` double DEFAULT NULL,
15. `latitude\_y` double DEFAULT NULL,
16. `longitude\_y` double DEFAULT NULL,
17. `vendor\_category\_en` text,
18. `vendor\_category\_id` double DEFAULT NULL,
19. `delivery\_charge` double DEFAULT NULL,
20. `serving\_distance` double DEFAULT NULL,
21. `is\_open` double DEFAULT NULL,
22. `OpeningTime` text,
23. `OpeningTime2` text,
24. `prepration\_time` bigint DEFAULT NULL,
25. `commission` double DEFAULT NULL,
26. `is\_akeed\_delivering` text,
27. `discount\_percentage` double DEFAULT NULL,
28. `status\_y` double DEFAULT NULL,
29. `verified\_y` bigint DEFAULT NULL,
30. `rank` bigint DEFAULT NULL,
31. `language` text,
32. `vendor\_rating` double DEFAULT NULL,
33. `sunday\_from\_time1` text,
34. `sunday\_to\_time1` text,
35. `sunday\_from\_time2` text,
36. `sunday\_to\_time2` text,
37. `monday\_from\_time1` text,
38. `monday\_to\_time1` text,
39. `monday\_from\_time2` text,
40. `monday\_to\_time2` text,
41. `tuesday\_from\_time1` text,
42. `tuesday\_to\_time1` text,
43. `tuesday\_from\_time2` text,
44. `tuesday\_to\_time2` text,
45. `wednesday\_from\_time1` text,
46. `wednesday\_to\_time1` text,
47. `wednesday\_from\_time2` text,
48. `wednesday\_to\_time2` text,
49. `thursday\_from\_time1` text,
50. `thursday\_to\_time1` text,
51. `thursday\_from\_time2` text,
52. `thursday\_to\_time2` text,
53. `friday\_from\_time1` text,
54. `friday\_to\_time1` text,
55. `friday\_from\_time2` text,
56. `friday\_to\_time2` text,
57. `saturday\_from\_time1` text,
58. `saturday\_to\_time1` text,
59. `saturday\_from\_time2` text,
60. `saturday\_to\_time2` text,
61. `primary\_tags` text,
62. `open\_close\_flags` double DEFAULT NULL,
63. `vendor\_tag` text,
64. `vendor\_tag\_name` text,
65. `one\_click\_vendor` text,
66. `country\_id` double DEFAULT NULL,
67. `city\_id` double DEFAULT NULL,
68. `created\_at\_y` text,
69. `updated\_at\_y` text,
70. `device\_type` bigint DEFAULT NULL,
71. `display\_orders` bigint DEFAULT NULL,
72. `location\_number\_obj` bigint DEFAULT NULL,
73. `id\_obj` bigint DEFAULT NULL,
74. `CID X LOC\_NUM X VENDOR` text,
75. `target` bigint DEFAULT NULL,
76. KEY `ix\_RestaurantData\_index` (`index`)
77. ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci |

With data exploration, it was found that column `CID X LOC\_NUM X VENDOR` has all unique keys. Thus, an unique constraint was created using the below query on this column for CRUD operations.

Alter table restaurantdata add constraint unqcol unique(`CID X LOC\_NUM X VENDOR`(50));

Graphical user interface, application, Word

Description automatically generated

**MongoDB:**

Graphical user interface, text, application, email

Description automatically generated

MongoDB Table structure:

1. db.RestaurantData.findOne()
2. { \_id: ObjectId("61c6d05073eab20d9f16f50b"),
3. customer\_id: 'TCHWPBT',
4. gender: 'Male',
5. status\_x: '1',
6. verified\_x: '1',
7. created\_at\_x: '2018-02-07 19:16:23',
8. updated\_at\_x: '2018-02-07 19:16:23',
9. location\_number: '0',
10. location\_type: 'Work',
11. latitude\_x: '-96.44',
12. longitude\_x: '-67.2',
13. id: '4',
14. authentication\_id: '118597.0',
15. latitude\_y: '-0.5884',
16. longitude\_y: '0.7544',
17. vendor\_category\_en: 'Restaurants',
18. vendor\_category\_id: '2.0',
19. delivery\_charge: '0.0',
20. serving\_distance: '6.0',
21. is\_open: '1.0',
22. OpeningTime: '11:00AM-11:30PM',
23. OpeningTime2: '-',
24. prepration\_time: '15',
25. commission: '0.0',
26. is\_akeed\_delivering: 'Yes',
27. discount\_percentage: '0.0',
28. status\_y: '1.0',
29. verified\_y: '1',
30. rank: '11',
31. language: 'EN',
32. vendor\_rating: '4.4',
33. sunday\_from\_time1: '00:00:00',
34. sunday\_to\_time1: '00:30:00',
35. sunday\_from\_time2: '08:00:00',
36. sunday\_to\_time2: '23:59:00',
37. monday\_from\_time1: '00:00:00',
38. monday\_to\_time1: '00:30:00',
39. monday\_from\_time2: '08:00:00',
40. monday\_to\_time2: '23:59:00',
41. tuesday\_from\_time1: '00:00:00',
42. tuesday\_to\_time1: '00:30:00',
43. tuesday\_from\_time2: '08:00:00',
44. tuesday\_to\_time2: '23:59:00',
45. wednesday\_from\_time1: '00:00:00',
46. wednesday\_to\_time1: '00:30:00',
47. wednesday\_from\_time2: '08:00:00',
48. wednesday\_to\_time2: '23:59:00',
49. thursday\_from\_time1: '00:00:00',
50. thursday\_to\_time1: '00:30:00',
51. thursday\_from\_time2: '08:00:00',
52. thursday\_to\_time2: '23:59:00',
53. friday\_from\_time1: '00:00:00',
54. friday\_to\_time1: '00:30:00',
55. friday\_from\_time2: '10:00:00',
56. friday\_to\_time2: '23:59:00',
57. saturday\_from\_time1: '00:00:00',
58. saturday\_to\_time1: '00:30:00',
59. saturday\_from\_time2: '10:00:00',
60. saturday\_to\_time2: '23:59:00',
61. primary\_tags: '{"primary\_tags":"4"}',
62. open\_close\_flags: '1.0',
63. vendor\_tag: '2,4,5,8,91,22,12,24,16,23',
64. vendor\_tag\_name: 'Arabic,Breakfast,Burgers,Desserts,Free Delivery,Grills,Lebanese,Salads,Sandwiches,Shawarma',
65. one\_click\_vendor: 'Y',
66. country\_id: '1.0',
67. city\_id: '1.0',
68. created\_at\_y: '2018-01-30 14:42:04',
69. updated\_at\_y: '2020-04-07 15:12:43',
70. device\_type: '3',
71. display\_orders: '1',
72. location\_number\_obj: '0',
73. id\_obj: '4',
74. 'CID X LOC\_NUM X VENDOR': 'TCHWPBT X 0 X 4',
75. target: '0' }
76. **Exact Queries used:**

Use cases used:

1. New customers are mapped to existing vendors (Insert occurs)
2. Adding/updating menu items for existing vendors for all its customers (Update occurs)
3. Delete new customer mapping records inserted in 1.
4. Group number of vendors based on preparation time taken and order by preparation time in descending order.

**MySQL:**

**Write Operation:**

INSERT INTO … ON DUPLICATE KEY UPDATE command is used to perform UPSERT query.

When run, the below query check if the unique key already exists. If yes, it performs an UPDATE to existing record if not INSERT a new record.

The unique key is the `CID X LOC\_NUM X VENDOR` and is used to update/insert ID and vendor\_tag\_name details.

INSERT INTO restaurantdata (`CID X LOC\_NUM X VENDOR`,ID, vendor\_tag\_name)

VALUES ('AAA X 0 X 1','1', 'Breakfast,Cakes,Crepes') ON DUPLICATE KEY UPDATE

ID='1', vendor\_tag\_name='Pani Puri';

**Read Operation:**

select `CID X LOC\_NUM X VENDOR`,ID, vendor\_tag\_name from restaurantdata where id=1;

**Delete Operation:**

delete from restaurantdata where `CID X LOC\_NUM X VENDOR`='AAA X 0 X 1';

**Groupby and Orderby Operation:**

select prepration\_time, count(distinct id) as "No of Vendors" from restaurantdata group by prepration\_time order by 2 desc;

**MongoDB :**

**Write Operation:**

const query = { id : '1' };

const update = { $set: { vendor\_tag\_name: "Deli Llama"}};

const options = {upsert: true};

db.RestaurantData.updateOne(query, update, options);

**Read Operation:**

db.RestaurantData.find({id:'1'})

**Delete Operation:**

db.RestaurantData.deleteOne({id:'1'})

**Groupby and Orderby Operation:**

db.RestaurantData.aggregate([

{"$group":{

"\_id":"$prepration\_time"

}},

{"$group": {

"\_id": "$\_id.prepration\_time",

"Count":{"$sum": 1}}},

{"$sort": { "Count": -1}}

])

1. **Performance parameters**

Based on the top 3 performance parameters published [here](https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/) in this report, we have considered the top 1 parameter which is “Query Response Time” for Query performance evaluation.

In addition, we acknowledge that the real-time query performance is affected by delays caused by locks and consistency mode configured. However, evaluation on these parameters have not been possible in the test setup.

1. **Visualization on performance parameters**

Write comparison:

Read Comparison:

Delete Comparison:

GroupBy & OrderBy Comparison:

1. **Tabular summary on performance parameters:**

|  |  |  |
| --- | --- | --- |
|  | **ExecutionTime in ms** | |
| **Database Operation** | *MySQL* | *MongoDB* |
| Write | 15 | 1 |
| Read | 11650 | 2 |
| Delete | 16 | 1 |
| Groupby/Orderby | 13750 | 26936 |

Based on the above performance parameters, it is evident that MongoDB(Non-SQL DB) gives better performance for “Read, Write and Delete” operations while MySQL(SQL DB) is better for “group by” and “order by”operations.

1. **Recommendation:**

NoSQL databases like MongoDB is suggested for use cases involving high read, write and delete operations as it clearly gives very high performance in these operations compared to relational database like MySQL. MongoDB is seen to have approximately 15x better performance in this regard.

For use cased involving high aggregate and sorting operations like group by, distinct and order by, relational database like MySQL is seen to perform relatively better than NoSQL database like MongoDB. This might be more suitable for Data Warehousing environment with OLAP transactions.

MongoDB is best suited if we consider data acquisition but there is small trade-off as MySQL works better in aggregating data so we might take little hit if Data Science team requirement is to work on pre-aggregated data.

**Overall, considering the performance parameters, current data volume and future scope of growth, we would recommend MongoDB.**

1. **Other Factors & Observation:**
2. This exercise has been performed on single DB instance we might get different result when tested on clustered or distributed systems.
3. MySQL database being an relational SQL database, mandatorily needs an unique key defined for the CRUD operations, which is an overhead. This overhead is eliminated in the NoSQL database.
4. There are equal number of records for each of the vendors. Thus data is not biased and the test queries run on any vendor id would have resulted in the same execution time.