**BDS Assignment**

**Group 75**

**Nareshkumar P (2020FC04122)**

**Kommajyosula VNS Kanth (2020fc04120)**

**Dola Tejesh (2020fc04459)**

Contents

[1. Introduction 2](#_Toc91540930)

[2. MongoDB Analysis: 2](#_Toc91540931)

[3. Postgresql Analysis 13](#_Toc91540932)

[4. MongoDB vs Postgresql Analysis 26](#_Toc91540933)

[5. Visualization 26](#_Toc91540934)

[6. Conclusion 27](#_Toc91540935)

# Introduction

* 1. Objective:

Analyze and understand the huge volume of Restaurant’s dataset. Load the dataset into Sql and NoSql databases and choose the best database for this dataset by comparing the performances of the queries.

Evaluating the performances of the databases by performing the following operations on the Sql and NoSql databases.

1. Loading
2. Reading
3. Writing
4. Deleting
5. Aggregating
   1. NoSql/Sql:

For Nosql, we used MongoDB Nosql database for the performance analysis. It’s a document based Nosql databases and its load the data in key-value format. Its very fast in nature while loading and reading the data from the database.

For Sql, we used postgresql database for the performance analysis. It’s a relational database and it’s tough to load huge size data like images, video, audio and logs data in databases. It performs good for read/write/delete operations for the certain volume of data.

* 1. File used:

To validate the performances of the two dbs, we have used train\_full.csv file since it has 3GB of data. The big files are always better to evaluate the performance when compared to the small sized files.

* 1. System Configurations:

Processor: Intel I3 8th Gen

RAM: 8GB

Disk Space: 120 GB

# MongoDB Analysis:

* 1. Schema:

Before loading the data, need to create a schema in db. We have created a new schema named ‘bds’ using below command.

**Execution command:** use bds

* 1. Data Import:

Mongoimport command is used to import the file from local disk to mongodb database.

**Collection name:** train\_full

**Note:** Collection name is same as table name in sql databases

**Execution Command:**

C:\Program Files\MongoDB\Server\5.0\bin>mongoimport --host "localhost" --port 27017 -d bds --collection train\_full --type CSV --file "F:\BITS Assignments\BDS\_Assignment\train\_full.csv" --headerline

Start time: 2021-12-23T13:44:31.225+0530

End time: 2021-12-23T14:01:29.676+0530

Total runtime: **16 Mins 58 Secs**

No. of Records Loaded:

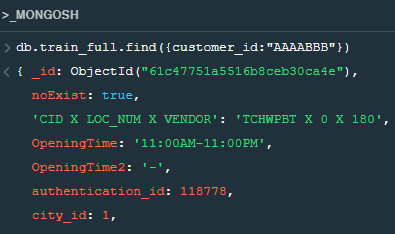
2021-12-23T14:01:29.676+0530 **5802400 document(s) imported successfully**. 0 document(s) failed to import.

* 1. Data Read:

Reading the particular record based on the particular customer\_id “TCHWPBT” in the train\_full collection.

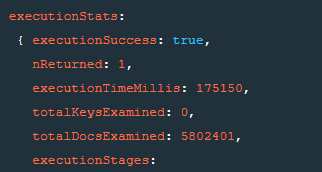
**Execution Command:**

db.train\_full.find({customer\_id:"AAAABBB"})



**Execution Time:**

2 Mins 55 Secs



**Performance Tuning by fetching record using key value(\_id):**

db.train\_full.find({\_id:ObjectId("61c47751a5516b8ceb30ca4e")})

Time taken: 82 milliseconds

executionStats:

{ executionSuccess: true,

nReturned: 1,

executionTimeMillis: 82,

* 1. Insert:

Inserting a new record/document in the collections using db.train\_full.insert() command.

We can use db.train\_full.insertOne, db.train\_full.insertMany, db.train.bulkWrite commands to insert one record, insert many records, loading bulk of records in a single execution respectively.

**Execution Command:**

db.train\_full.insert(

{

customer\_id: 'BBBBAAA',

gender: 'Male',

status\_x: 1,

verified\_x: 1,

created\_at\_x: '2021-12-07 19:16:23',

updated\_at\_x: '2021-12-07 19:16:23',

location\_number: 0,

location\_type: 'Work',

latitude\_x: -96.44,

longitude\_x: -67.2,

id: 180,

authentication\_id: 118778,

latitude\_y: -1.059,

longitude\_y: 0.05157,

vendor\_category\_en: 'Restaurants',

vendor\_category\_id: 2,

delivery\_charge: 0.7,

serving\_distance: 15,

is\_open: 0,

OpeningTime: '11:00AM-11:00PM',

OpeningTime2: '-',

prepration\_time: 15,

commission: 0,

is\_akeed\_delivering: 'Yes',

discount\_percentage: 0,

status\_y: 0,

verified\_y: 0,

rank: 11,

language: 'EN',

vendor\_rating: 3.8,

sunday\_from\_time1: '11:00:00',

sunday\_to\_time1: '23:00:00',

sunday\_from\_time2: '',

sunday\_to\_time2: '',

monday\_from\_time1: '11:00:00',

monday\_to\_time1: '23:00:00',

monday\_from\_time2: '',

monday\_to\_time2: '',

tuesday\_from\_time1: '11:00:00',

tuesday\_to\_time1: '23:00:00',

tuesday\_from\_time2: '',

tuesday\_to\_time2: '',

wednesday\_from\_time1: '11:00:00',

wednesday\_to\_time1: '23:00:00',

wednesday\_from\_time2: '',

wednesday\_to\_time2: '',

thursday\_from\_time1: '11:00:00',

thursday\_to\_time1: '23:45:00',

thursday\_from\_time2: '',

thursday\_to\_time2: '',

friday\_from\_time1: '11:00:00',

friday\_to\_time1: '23:45:00',

friday\_from\_time2: '',

friday\_to\_time2: '',

saturday\_from\_time1: '11:00:00',

saturday\_to\_time1: '23:00:00',

saturday\_from\_time2: '',

saturday\_to\_time2: '',

primary\_tags: '',

open\_close\_flags: 1,

vendor\_tag: '2,22,10,27,24,17,28,56',

vendor\_tag\_name: 'Arabic,Grills,Indian,Pasta,Salads,Seafood,Soups,Sweets',

one\_click\_vendor: 'Y',

country\_id: 1,

city\_id: 1,

created\_at\_y: '2021-12-23 15:30:39',

updated\_at\_y: '2021-12-23 11:47:36',

device\_type: 3,

display\_orders: 1,

location\_number\_obj: 0,

id\_obj: 180,

'CID X LOC\_NUM X VENDOR': 'TCHWPBT X 0 X 180',

target: 0 } )

**Log:**

{ acknowledged: true,

insertedIds: { '0': ObjectId("61c46f18c10b0db45537d243") } }

db.train\_full.insert(

{

customer\_id: 'BBBBAAA',

gender: 'Male',

status\_x: 1,

verified\_x: 1,

Time taken: Less than 1 second

* 1. Upsert: (Update and Insert)

As per the requirement, need to update a record if already exists otherwise insert a new record in a collection.

In the below example, looking for a customer\_id ('AAAABBB') exists or not. If exists then update the existing document else insert a new document in the collection.

We can use updateOne, updateMany commands to update the documents in the collections. $Set is used to update the document if it exists already.

**Execution Command:**

db.train\_full.update({customer\_id: 'AAAABBB'}, {"$set": {customer\_id: 'AAAABBB',

gender: 'Male',

status\_x: 1,

verified\_x: 1,

created\_at\_x: '2021-12-17 19:16:23',

updated\_at\_x: '2021-12-17 19:16:23',

location\_number: 0,

location\_type: 'Work',

latitude\_x: -96.44,

longitude\_x: -67.2,

id: 180,

authentication\_id: 118778,

latitude\_y: -1.059,

longitude\_y: 0.05157,

vendor\_category\_en: 'Restaurants',

vendor\_category\_id: 2,

delivery\_charge: 0.7,

serving\_distance: 15,

is\_open: 0,

OpeningTime: '11:00AM-11:00PM',

OpeningTime2: '-',

prepration\_time: 15,

commission: 0,

is\_akeed\_delivering: 'Yes',

discount\_percentage: 0,

status\_y: 0,

verified\_y: 0,

rank: 11,

language: 'EN',

vendor\_rating: 3.8,

sunday\_from\_time1: '11:00:00',

sunday\_to\_time1: '23:00:00',

sunday\_from\_time2: '',

sunday\_to\_time2: '',

monday\_from\_time1: '11:00:00',

monday\_to\_time1: '23:00:00',

monday\_from\_time2: '',

monday\_to\_time2: '',

tuesday\_from\_time1: '11:00:00',

tuesday\_to\_time1: '23:00:00',

tuesday\_from\_time2: '',

tuesday\_to\_time2: '',

wednesday\_from\_time1: '11:00:00',

wednesday\_to\_time1: '23:00:00',

wednesday\_from\_time2: '',

wednesday\_to\_time2: '',

thursday\_from\_time1: '11:00:00',

thursday\_to\_time1: '23:45:00',

thursday\_from\_time2: '',

thursday\_to\_time2: '',

friday\_from\_time1: '11:00:00',

friday\_to\_time1: '23:45:00',

friday\_from\_time2: '',

friday\_to\_time2: '',

saturday\_from\_time1: '11:00:00',

saturday\_to\_time1: '23:00:00',

saturday\_from\_time2: '',

saturday\_to\_time2: '',

primary\_tags: '',

open\_close\_flags: 1,

vendor\_tag: '2,22,10,27,24,17,28,56',

vendor\_tag\_name: 'Arabic,Grills,Indian,Pasta,Salads,Seafood,Soups,Sweets',

one\_click\_vendor: 'Y',

country\_id: 1,

city\_id: 1,

created\_at\_y: '2021-12-24 15:30:39',

updated\_at\_y: '2021-12-24 11:47:36',

device\_type: 3,

display\_orders: 1,

location\_number\_obj: 0,

id\_obj: 180,

'CID X LOC\_NUM X VENDOR': 'TCHWPBT X 0 X 180',

target: 0 }}, {upsert: true})

**For Update:**

db.train\_full.update({customer\_id: 'AAAABBB'}, {"$set": {customer\_id: 'AAAABBB',

gender: 'Male',

status\_x: 1,

verified\_x: 1,

created\_at\_x: '2021-12-17 19:16:23',

updated\_at\_x: '2021-12-17 19:16:23',

Time taken: 67 seconds

**Commands used to find the time taken of the query:**

var before = new Date()

#query

var after = new Date()

execution\_mills = after – before

67991

67991 milliseconds.

**Log:**

{ acknowledged: true,

insertedId: null,

matchedCount: 1,

modifiedCount: 1,

upsertedCount: 0 }

**For new Insert:**

var before = new Date()

db.train\_full.update({customer\_id: 'BBBBBBB'}, {"$set": {customer\_id: 'AAAABBB',

gender: 'Male',

status\_x: 1,

verified\_x: 1,

created\_at\_x: '2021-12-17 19:16:23',

Time taken: 57 Seconds

57969

57969 milliseconds.

**Log:**

{ acknowledged: true,

insertedId: ObjectId("61c47751a5516b8ceb30ca4e"),

matchedCount: 0,

modifiedCount: 0,

upsertedCount: 1 }

* 1. Delete:

Delete a record/document in train\_full collections using db.train\_full.remove() command.

We can use db.train\_full.deleteOne, db.train\_full.deleteMany commands to delete one record, delete many records in a single execution respectively.

**Execution Command:**

db.train\_full.remove({ customer\_id: 'AAABBBB' })

var before = new Date()

db.train\_full.remove({ customer\_id: 'AAABBBB' })

var after = new Date()

execution\_mills = after - before

131729

Time taken: 131729 milli seconds => 2 Mins 11 Secs

**Log:**

{ acknowledged: true, deletedCount: 1 }

**Performance Tuning:**

Using DeleteOne command delete a record.

var before = new Date()

db.train\_full.deleteOne({ customer\_id: 'AAABBBB' })

var after = new Date()

execution\_mills = after - before

63901

Time taken: 63 Secs

‘DeleteOne’/’DeleteMany’ command performs better than ‘remove’ function.

* 1. Aggregate: (Group by/ Order by)

Calculating the record count of each customer id by using db.train\_full.aggregate() command and sorting the result set by record count in descending order.

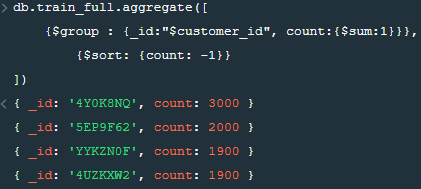
**Execution Command:**

db.train\_full.aggregate([

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

{$sort: {count: -1}}

])

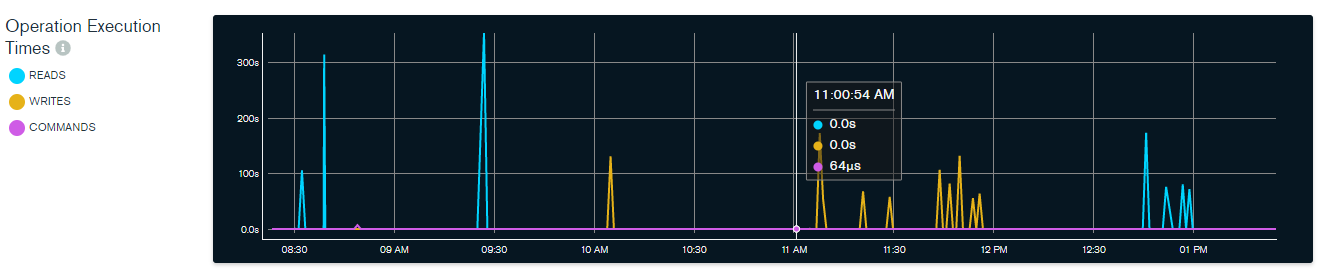


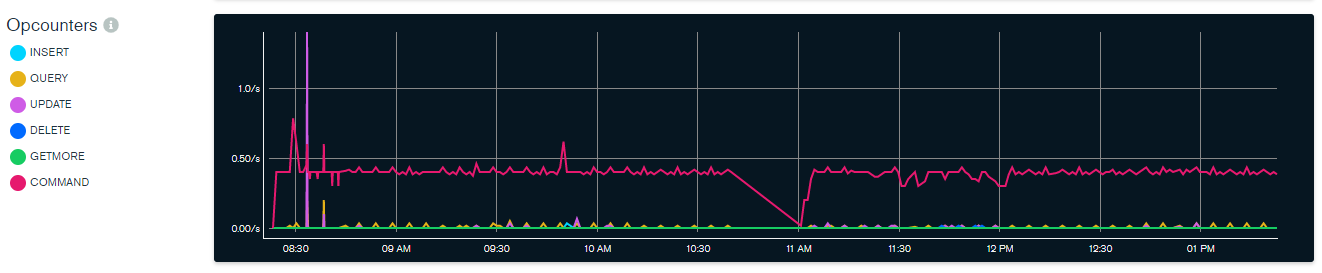
**Time taken:**

81495

81495 Milliseconds => 81 Seconds

* 1. MongoDB Operation Execution Graphs:





# Postgresql Analysis

* 1. Schema:

Used common ‘public’ schema to create new table.

Before loading the data, need to create a new table in db. We have created a new table named ‘train\_full’ with all the columns mentioned in the train\_full.csv file.

**Schema**: Public

**Table\_Name**: train\_full

**Table DDL:**

CREATE TABLE PUBLIC.train\_full (

customer\_id VARCHAR,

gender VARCHAR,

status\_x NUMERIC,

verified\_x NUMERIC,

created\_at\_x TIMESTAMP,

updated\_at\_x TIMESTAMP,

location\_number NUMERIC,

location\_type VARCHAR,

latitude\_x FLOAT8,

longitude\_x FLOAT8,

id NUMERIC,

authentication\_id NUMERIC ,

latitude\_y FLOAT8,

longitude\_y FLOAT8,

vendor\_category\_en VARCHAR,

vendor\_category\_id NUMERIC,

delivery\_charge FLOAT8,

serving\_distance NUMERIC,

is\_open NUMERIC,

OpeningTime VARCHAR,

OpeningTime2 VARCHAR,

prepration\_time NUMERIC,

commission NUMERIC,

is\_akeed\_delivering VARCHAR,

discount\_percentage NUMERIC,

status\_y NUMERIC,

verified\_y NUMERIC ,

rank NUMERIC,

language VARCHAR,

vendor\_rating FLOAT8,

sunday\_from\_time1 TIME,

sunday\_to\_time1 TIME,

sunday\_from\_time2 TIME,

sunday\_to\_time2 TIME,

monday\_from\_time1 TIME,

monday\_to\_time1 TIME ,

monday\_from\_time2 TIME,

monday\_to\_time2 TIME,

tuesday\_from\_time1 TIME,

tuesday\_to\_time1 TIME,

tuesday\_from\_time2 TIME,

tuesday\_to\_time2 TIME,

wednesday\_from\_time1 TIME,

wednesday\_to\_time1 TIME,

wednesday\_from\_time2 TIME,

wednesday\_to\_time2 TIME,

thursday\_from\_time1 TIME,

thursday\_to\_time1 TIME,

thursday\_from\_time2 TIME,

thursday\_to\_time2 TIME,

friday\_from\_time1 TIME,

friday\_to\_time1 TIME,

friday\_from\_time2 TIME,

friday\_to\_time2 TIME,

saturday\_from\_time1 TIME,

saturday\_to\_time1 TIME,

saturday\_from\_time2 TIME,

saturday\_to\_time2 TIME,

primary\_tags VARCHAR,

open\_close\_flags NUMERIC,

vendor\_tag VARCHAR,

vendor\_tag\_name VARCHAR,

one\_click\_vendor VARCHAR,

country\_id NUMERIC,

city\_id NUMERIC,

created\_at\_y TIMESTAMP,

updated\_at\_y TIMESTAMP,

device\_type NUMERIC ,

display\_orders NUMERIC,

location\_number\_obj NUMERIC,

id\_obj NUMERIC,

cid\_x\_loc\_num\_x\_vendor VARCHAR PRIMARY KEY,

target NUMERIC

);

* 1. Data Import:

Used copy command to import the data of train\_full.csv file into public.train\_full table.

**Execution Command:**

copy public.train\_full FROM 'F:\BITS Assignments\BDS\_Assignment\train\_full.csv' with csv header;



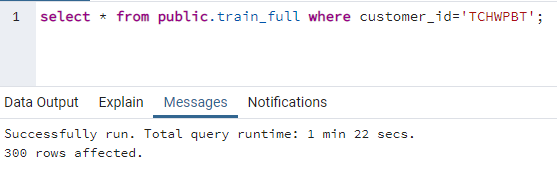
**Time Taken:** 19 Min 27 Seconds

* 1. Read:

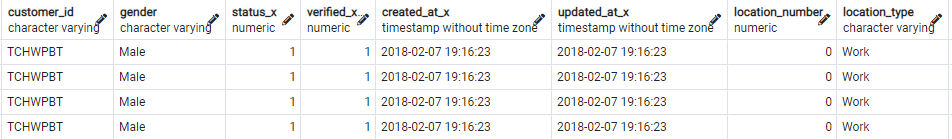
To perform read operation, used select query to retrieve a customer\_id=’TCHWPBT’

**Query:**

select \* from public.train\_full where customer\_id='TCHWPBT';



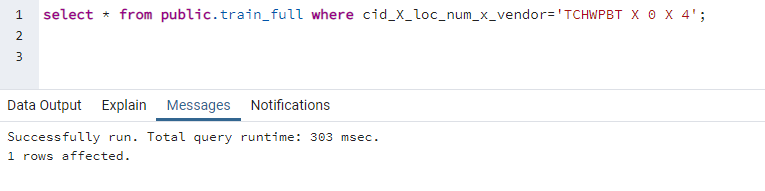
**Query Result:**



**Time Taken:** 1 Min 22 Secs

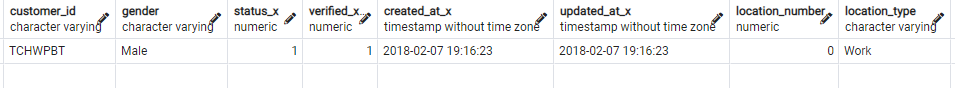
**Performance Tuning:**

Selecting the data based on primary key or Index columns, it will improve the performance and provide results in less than a second.



**Time taken:** 303 milliseconds

**Query Result:**

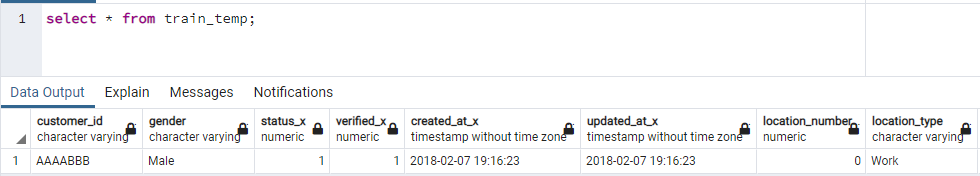


Querying the data using key/indexed columns showing better performance than the querying non-index columns

* 1. Upsert: (Update and Insert)

As per the requirement, need to update a record if already exists otherwise insert a new record in a collection.

Created a temp table (train\_temp) with new customer\_id(‘AAAABBB’) with same set of columns as train\_full table. Preparing the query by checking records in train\_temp exists in train\_full table or not, If exists then update else insert in train\_full table.



In the below query, looking for a customer\_id ('AAAABBB') exists or not. If exists then update the existing record else insert a new record in the train\_full table.

In oracle db, Merge query helps to execute the upsert logic but postgresql not supporting the merge query hence wrote the below script to implement the upsert logic.

**Query:**

Begin work;

UPDATE public.train\_full

SET city\_id = train\_temp.city\_id,

country\_id=train\_temp.country\_id,

target=train\_temp.target,

updated\_at\_x=train\_temp.updated\_at\_x,

updated\_at\_y=train\_temp.updated\_at\_y

from public.train\_temp train\_temp

WHERE train\_full.customer\_id = train\_temp.customer\_id;

INSERT INTO train\_full (customer\_id ,

gender ,

status\_x ,

verified\_x ,

created\_at\_x ,

updated\_at\_x ,

location\_number ,

location\_type ,

latitude\_x ,

longitude\_x ,

id ,

authentication\_id ,

latitude\_y ,

longitude\_y ,

vendor\_category\_en ,

vendor\_category\_id ,

delivery\_charge ,

serving\_distance ,

is\_open ,

OpeningTime ,

OpeningTime2 ,

prepration\_time ,

commission ,

is\_akeed\_delivering ,

discount\_percentage ,

status\_y ,

verified\_y ,

rank ,

language ,

vendor\_rating ,

sunday\_from\_time1 ,

sunday\_to\_time1 ,

sunday\_from\_time2 ,

sunday\_to\_time2 ,

monday\_from\_time1 ,

monday\_to\_time1 ,

monday\_from\_time2 ,

monday\_to\_time2 ,

tuesday\_from\_time1 ,

tuesday\_to\_time1 ,

tuesday\_from\_time2 ,

tuesday\_to\_time2 ,

wednesday\_from\_time1 ,

wednesday\_to\_time1 ,

wednesday\_from\_time2 ,

wednesday\_to\_time2 ,

thursday\_from\_time1 ,

thursday\_to\_time1 ,

thursday\_from\_time2 ,

thursday\_to\_time2 ,

friday\_from\_time1 ,

friday\_to\_time1 ,

friday\_from\_time2 ,

friday\_to\_time2 ,

saturday\_from\_time1 ,

saturday\_to\_time1 ,

saturday\_from\_time2 ,

saturday\_to\_time2 ,

primary\_tags ,

open\_close\_flags ,

vendor\_tag ,

vendor\_tag\_name ,

one\_click\_vendor ,

country\_id ,

city\_id ,

created\_at\_y ,

updated\_at\_y ,

device\_type ,

display\_orders ,

location\_number\_obj ,

id\_obj ,

cid\_x\_loc\_num\_x\_vendor,

target )

(SELECT train\_temp.customer\_id ,

train\_temp.gender ,

train\_temp.status\_x ,

train\_temp.verified\_x ,

train\_temp.created\_at\_x ,

train\_temp.updated\_at\_x ,

train\_temp.location\_number ,

train\_temp.location\_type ,

train\_temp.latitude\_x ,

train\_temp.longitude\_x ,

train\_temp.id ,

train\_temp.authentication\_id ,

train\_temp.latitude\_y ,

train\_temp.longitude\_y ,

train\_temp.vendor\_category\_en ,

train\_temp.vendor\_category\_id ,

train\_temp.delivery\_charge ,

train\_temp.serving\_distance ,

train\_temp.is\_open ,

train\_temp.OpeningTime ,

train\_temp.OpeningTime2 ,

train\_temp.prepration\_time ,

train\_temp.commission ,

train\_temp.is\_akeed\_delivering ,

train\_temp.discount\_percentage ,

train\_temp.status\_y ,

train\_temp.verified\_y ,

train\_temp.rank ,

train\_temp.language ,

train\_temp.vendor\_rating ,

train\_temp.sunday\_from\_time1 ,

train\_temp.sunday\_to\_time1 ,

train\_temp.sunday\_from\_time2 ,

train\_temp.sunday\_to\_time2 ,

train\_temp.monday\_from\_time1 ,

train\_temp.monday\_to\_time1 ,

train\_temp.monday\_from\_time2 ,

train\_temp.monday\_to\_time2 ,

train\_temp.tuesday\_from\_time1 ,

train\_temp.tuesday\_to\_time1 ,

train\_temp.tuesday\_from\_time2 ,

train\_temp.tuesday\_to\_time2 ,

train\_temp.wednesday\_from\_time1 ,

train\_temp.wednesday\_to\_time1 ,

train\_temp.wednesday\_from\_time2 ,

train\_temp.wednesday\_to\_time2 ,

train\_temp.thursday\_from\_time1 ,

train\_temp.thursday\_to\_time1 ,

train\_temp.thursday\_from\_time2 ,

train\_temp.thursday\_to\_time2 ,

train\_temp.friday\_from\_time1 ,

train\_temp.friday\_to\_time1 ,

train\_temp.friday\_from\_time2 ,

train\_temp.friday\_to\_time2 ,

train\_temp.saturday\_from\_time1 ,

train\_temp.saturday\_to\_time1 ,

train\_temp.saturday\_from\_time2 ,

train\_temp.saturday\_to\_time2 ,

train\_temp.primary\_tags ,

train\_temp.open\_close\_flags ,

train\_temp.vendor\_tag ,

train\_temp.vendor\_tag\_name ,

train\_temp.one\_click\_vendor ,

train\_temp.country\_id ,

train\_temp.city\_id ,

train\_temp.created\_at\_y ,

train\_temp.updated\_at\_y ,

train\_temp.device\_type ,

train\_temp.display\_orders ,

train\_temp.location\_number\_obj ,

train\_temp.id\_obj ,

train\_temp.cid\_x\_loc\_num\_x\_vendor,

train\_temp.target

FROM train\_temp

LEFT OUTER JOIN train\_full ON (train\_full.customer\_id = train\_temp.customer\_id)

WHERE train\_full.customer\_id IS NULL);

commit;

end;



**Time taken:** 40 seconds

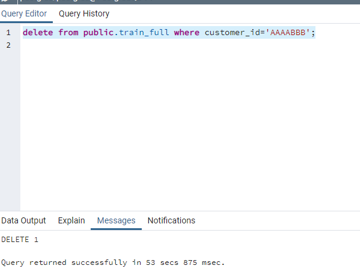
* 1. Delete:

Delete a record which has customer\_id=’AAABBBB’ in train\_full table using delete query.

**Query:**

delete from public.train\_full where customer\_id='AAAABBB';

commit;



**Time taken:** 53 seconds

**Performance Tuning:**

Delete records using index or key columns will improve the performance of deletion.

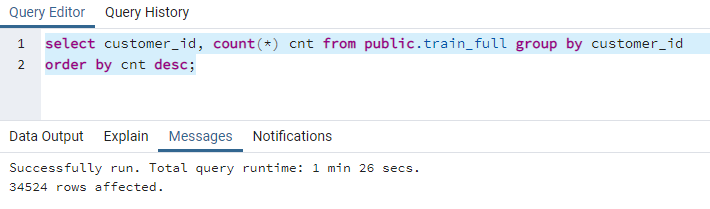
* 1. Aggregate: (Group by/ Order by)

Calculating the record count of each customer id by using ‘group by’ and sorting the result set by record count in descending order using ‘order by’.

**Query**:

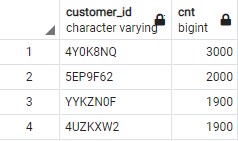
select customer\_id, count(\*) cnt from public.train\_full group by customer\_id

order by cnt desc;



**Time taken:** 1 Min 26 Secs

**Query Result:**



# MongoDB vs Postgresql Analysis

Tabulating the results.

|  |  |  |
| --- | --- | --- |
| **Operations** | **Runtime (in Seconds)** | |
|  | **MongoDB** | **Postgresql** |
| Import | 1018 | 1167 |
| Read | 175 | 142 |
| Upsert | 67 | 40 |
| Delete | 131 | 53 |
| Aggregate | 81 | 86 |

\*\*Green color indicates the good performance than the other.

# Visualization

# Conclusion

Postgresql performs better in Upsert, read and delete for the given dataset whereas Mongodb performs better than Postgresql in Import and aggregation operations.

For OLTP, Both Mongodb and Postgresql performs better since both are showing good performance in loading and reading records.

For OLAP, Postgresql slightly showing less performance than the Mongodb while doing aggregation. I assume Postgresql performance may degrade if it’s used huge volume of dataset whereas Mongodb can perform better in huge dataset.

The downside of Postgresql compared to Mongodb is that it relies on relational data models that are unfriendly to the data structures developers work with in code, and that must be defined in advance, slowing progress whenever requirements change. Mongodb supports a rapid, iterative cycle of development so well because of the way that a document database turns data into code under the control of developers.

Mongodb and Postgresql both are performing better but as per the need we need to choose which one suits for our dataset. If we need to maintain only unstructured data in our database then we can go for Mongodb. **Since our given dataset is relational data and it requires OLAP model to aggregate data for the analysis and insights, we are suggesting Postgresql is the best option for the given dataset.**