



DATABASES

RUN TIME

ANALYSIS (MONGODB, NEO4J, MARIADB)



BY NAVEEN TIWARI
MAT ID: 522364

Objective:

In this report I am going to do a benchmark analysis of the runtime of three different databases: MongoDB, Neo4j, MariaDB. The databases contain information of the lost credit cards for that each database is divided into four tables of sizes: 25000, 50000, 75000 and 100000 rows. All the data have been imported from <https://www.onlinedatagenerator.com/>.

In four different csv files with sizes mention above. To get the runtime analysis I have used four queries that gives same output for the three-database system with increasing complexity.

The runtime analysis has been performed using Python. For MongoDB and MariaDB I have used docker containers and a normal Neo4j application because it was preinstalled.

GENERAL OVERVIEW OF THE DATA MODEL USED:

GUID	First Name	Last Name	Gender	Email Address	Phone Number	Credit Card Number	Credit Card Type	IBAN	City
a5baadc6-6d	Benny	Noach	Male	Benny_Noach5383@jiman.org	4-888-754-7720	6144-5442-8211-7010	American Express	d557n3Pa2038374613208604	San Antonio
f545fe83-d3f	Elle	Gray	Female	Elle_Gray1730@typill.biz	5-413-451-0220	2313-5354-6688-3611	Citibank	C575JMtC1480820784634856	Jersey City
44d492e1-72	Alba	Vernon	Female	Alba_Vernon7236@muall.tech	2-310-233-1238	3853-6481-4247-4031	American Express	ol08c1qp6163433855507657	Minneapolis
6df7916e-aa	Jacob	Baxter	Male	Jacob_Baxter6408@fuliss.net	8-284-038-8086	0216-0807-8460-5841	Chase	Qe80GMWJ5735555106255146	Detroit
0d0697b7-e8	Joy	Oldfield	Female	Joy_Oldfield3773@gompie.com	5-136-506-8035	3061-3808-4125-8001	MasterCard	bq48LSwc2800372206386251	Arlington
130e12c6-bb	Luke	Henderson	Male	Luke_Henderson3484@vetan.org	4-844-375-2686	0807-1282-8351-4103	UnionPay	V3375RhK7373345081513503	Escondido
6cbb24fc-e2f	Parker	Reid	Female	Parker_Reid9913@twipet.com	3-488-747-1646	3818-5701-5831-2670	Chase	i964zGe23574108552113637	Chicago
8ad539ed-01	Mara	Morris	Female	Mara_Morris6705@gembat.biz	0-325-672-4750	2216-4844-5358-1757	Maestro	Zx55tWsD7317363424156833	Moreno Valley
74379cf5-98	Samara	Giles	Female	Samara_Giles533@naiker.biz	1-821-004-4846	5526-2613-8231-2331	Bank of America	rk24wlB01534702103710702	Richmond
44fd9c5f-93f	Peter	Hall	Male	Peter_Hall9995@deavo.com	5-857-221-7827	0068-0200-1058-6848	American Express	nl20K3zd1188847716052016	Glendale
87434343-b2	Caitlyn	Santos	Female	Caitlyn_Santos8726@bulaffy.com	7-361-707-8183	7010-2471-5143-1485	Visa	VO84URih1307377572046510	San Bernardino
23021a58-d3	Joseph	Watt	Male	Joseph_Watt6314@muall.tech	7-355-808-4704	1172-1256-3280-4707	UnionPay	Bg74frJa6275762413825036	Long Beach
eda98f75-67	Nate	Dallas	Male	Nate_Dallas2144@sheye.org	7-124-210-5457	4070-4564-2314-0800	MasterCard	OW20eeBa5282174565382771	Portland
5d2334b4-f7	Faith	Hunter	Female	Faith_Hunter2280@grannar.com	1-173-137-0565	8262-0457-2361-8350	MasterCard	hE26VsuL2332857157445780	Santa Ana
08d4f433-62	Elijah	Partridge	Male	Elijah_Partridge9179@ovock.tech	4-354-437-5500	7025-3680-2505-5612	Maestro	bf22Ye0z3026628340771444	New York
44820c62-c8	Tania	Weatcroft	Female	Tania_Weatcroft9300@typill.biz	2-181-067-3537	0652-4211-0533-7806	Capital One	vG30aRBg7176661324785112	Bridgeport
30b706a2-23	Hank	Archer	Male	Hank_Archer526@gembat.biz	0-035-487-5357	5034-3818-5326-6581	American Express	IG06y6wo2063483374071426	Colorado Springs
1930a485-fe	Shelby	Bowen	Female	Shelby_Bowen6642@extex.org	4-738-240-7157	0835-0473-8182-1172	Maestro	TL03oENj0818657233331317	Detroit
ea4e0677-2c	Harvey	Pearce	Male	Harvey_Pearce2015@hourpy.biz	3-104-853-3718	8564-1400-7648-6781	Chase	6P502Fss6076804114503424	New Orleans
e2390cc0-aa	Henry	Kerr	Male	Henry_Kerr3111@muall.tech	0-374-548-0506	4322-8607-5330-1481	Wells Fargo	IA74XcOT0485130058711327	Seattle
e306be9f-82	Rebecca	Parker	Female	Rebecca_Parker6351@fuliss.net	5-244-512-1830	7102-3030-5410-2160	Visa	mO28bhUP673737714160312	Washington
90c1a829-c3	Angelica	Long	Female	Angelica_Long2572@gmail.com	7-114-610-3660	5281-2521-0652-0108	UnionPay	a988cnpj1215104807721681	Henderson
5bc762e6-24	Ron	Hope	Male	Ron_Hope850@bretoux.com	5-138-670-1734	0214-7502-8326-4606	Discover	Ou26lBee7060858866585301	Lancaster
725c91fd-16	Emmanuelle	Avery	Female	Emmanuelle_Avery8834@twipet.com	5-128-116-8260	0431-8727-6037-1760	Visa	9H107Wj76676063253473100	Anaheim
57f5027c-c8f	Alessandra	Moss	Female	Alessandra_Moss1151@deons.tech	4-058-158-0612	2030-6204-5411-6700	American Express	zg85lIYo3726527618054381	Portland
30dfaaa1-d1	Cameron	Quinn	Female	Cameron_Quinn8229@twace.org	1-452-125-6403	2477-0124-5803-7346	Discover	FA06mpFM8054414510430348	Berna
c809f650-27	Maxwell	Vass	Male	Maxwell_Vass7191@naiker.biz	6-475-486-6765	0060-1647-6303-6640	Capital One	Zd81PT575405435532323035	Innsbruck
7b8f1359-69	Sadie	Roman	Female	Sadie_Roman711@ovock.tech	8-732-364-7670	4242-2846-3317-7202	Bank of America	P037jDuJ0128836335110641	Seattle
3dad60fe-a4	Adeline	Gordon	Female	Adeline_Gordon5052@bungar.biz	2-455-741-8634	1617-4833-4671-7421	Wells Fargo	2B26jN5b2778164645784253	Anaheim
f5c29c25-b5f	Sharon	Edwards	Female	Sharon_Edwards7649@deons.tech	5-260-516-5423	1154-3034-2842-8547	Visa	IA41gtOo454782610361511	Portland
1197a7a0-2c	Chester	Wren	Male	Chester_Wren7285@joiniaa.com	1-358-472-5337	8348-4824-7500-8072	Capital One	Ej15tOpl1868838335767816	Tulsa
99f5b7a6-e7	Rocco	Briggs	Male	Rocco_Briggs7222@infotech44.tech	6-485-721-1177	3855-8106-5305-4133	Citibank	y247YB728710145704218850	Murfreesboro
786db4b9-77	Oliver	Sylvester	Male	Oliver_Sylvester4437@joiniaa.com	4-713-628-0851	8385-3836-3611-1764	Capital One	PZ24YmP008504260207662	Henderson



MongoDB is a [source-available cross-platform document-oriented database](#) program. Classified as a [NoSQL](#) database program, MongoDB uses [JSON](#)-like documents with optional [schemas](#). MongoDB is developed by [MongoDB Inc.](#) and licensed under the [Server Side Public License](#) (SSPL) which is deemed non-free by several distributions. The latter type of document extends the Json model to provide data types additional, ordered fields and to be efficient in encoding and decoding with several programming languages. A MongoDB instance can have zero or more databases, each of which acts as a top-tier container for everything else. A database can have zero or more 'collections'. A collection has a lot in common with Traditional 'tables'. The collections are made up of zero or more 'documents', the latter can be compared to the 'rows' (records) of a table. A document is in turn composed of one or more 'fields', similar to the concept of 'Columns'.



Neo4j is a [graph database management system](#) developed by Neo4j, Inc. Described by its developers as an [ACID](#)-compliant transactional database with native graph storage and processing, Neo4j is available in a [GPL3](#)-licensed [source-available](#) "community edition", with [online backup](#) and [high availability](#) extensions licensed under a closed-source commercial license. Neo also licenses Neo4j with these extensions under closed-source commercial terms. Neo4j is implemented in [Java](#) and accessible from software written in other languages using the [Cypher query language](#) through a transactional HTTP endpoint, or through the binary "[Bolt](#)" protocol The "4j" in Neo4j is a reference to its being built in Java, however is now largely viewed as an [anachronism](#). It is a Nosql database that is based on graph dbms that implements a database

management that is flexible in dealing with incredibly structured data. Neo4j graph DBMS exploit graphs for data storage, allowing the management of the management of closely related data structures, as in the case of social networks. Neo4j, not only allows us to store data, but also offers tools for the study of graphs, with the possibility of implementing very complex queries for the identification of nodes and arcs. More specifically, the graph that can be created is made up of nodes, arcs and property. Each node has properties, which represent our records and data from to memorize. The arcs, on the other hand, have a direction and represent the relationships between the nodes. Neo4j supports a declarative language called “Cypher”, a declarative language inspired by SQL. Cypher allows you to indicate what we want to select, insert, update or delete from our graphic data without a description of exactly how to do it.



MariaDB is a community-developed, commercially supported [fork](#) of the [MySQL relational database management system](#) (RDBMS), intended to remain [free and open-source software](#) under the [GNU General Public License](#). Development is led by some of the original developers of MySQL, who forked it due to concerns over its [acquisition](#) by [Oracle Corporation](#) in 2009.^[6]

MariaDB is intended to maintain high compatibility with MySQL, with library binary parity and exact matching with MySQL [APIs](#) and commands, allowing it in many cases to function as drop-in replacement for MySQL. However, new features are diverging.^[7] It includes new [storage engines](#) like [Aria](#), [ColumnStore](#), and [MyRocks](#).

Its lead developer/CTO is [Michael "Monty" Widenius](#), one of the founders of [MySQL AB](#) and the founder of Monty Program AB. On 16 January 2008, MySQL AB announced that it had agreed to be acquired by [Sun Microsystems](#) for approximately \$1 billion. The acquisition completed on 26 February 2008. Sun was then bought the following year

by [Oracle Corporation](#). MariaDB is named after Widenius' younger daughter, Maria. (MySQL is named after his other daughter, My

Implementation:

MongoDB:

For this database I have used docker. To make a mongodb container I have used the command:

```
docker run -d -p 27017:27017 --name dbmongo mongo:latest
```

As we have created a container so now we need to import the csv files to mongodb folder in order to use it for creating database for this we use the command:

```
docker cp /Users/naveen/Desktop/25.csv mongodb:/bin
```

now we will go inside the mongodb container using command:

```
docker exec -it dbmongo bash
```

Now we need to create four different collections from the csv files, for this we use the command:

```
mongoimport --db Lost_credit_card --collection db25 --type csv --file bin/25.csv --headerline
```

AS we have to create four collections inside the database `Lost_credit_card`, so we will change the name of collections and source csv file.

- We are done with MongoDB now we need to connect it with python.

Libraries imported:

```
1 import pandas
2 import pymongo
3 import neo4j
4 import csv
5 import mysql.connector
```

I have imported libraires pymongo, neo4j, MySQL for their respective databases. Along with it I have also used pandas and csv to read the csv data that we generated and to return the runtime of all the queries.

Now we need to make connection with the mongoDB database.

```
20
21 connect = pymongo.MongoClient("mongodb://localhost:27017")
22 mydb = connect["Lost_credit_card"]
```

As we have already crated the collections, we just need to perform the queries. For this report we will run it for 31 times.

```
for j in ('db100', 'db75', 'db50', 'db25'):
    xm = []
    xm1 = []
    xm2 = []
    xm3 = []
    csv_db = mydb[j]
    for i in range(0,31):
```



```

mydoc = csv_db.find({}).explain()
q1 = mydoc['executionStats']['executionTimeMillis']
xm.append(q1)
mydoc = csv_db.find({"Credit Card Type": "UnionPay"}).explain()
q2 = mydoc['executionStats']['executionTimeMillis']
xm1.append(q2)
query3 = csv_db.find({'City': 'Arlington'})
query3_time = query3.explain()
q3 = query3_time['executionStats']['executionTimeMillis']
xm2.append(q3)
query4_codes = csv_db.find({'Credit Card Number': '3853-6481-4247-4031'})
query4_time = query4_codes.explain()
q4 = query4_time['executionStats']['executionTimeMillis']
xm3.append(q4)
m3.append(xm3)
m2.append(xm2)
m1.append(xm1)
m.append(xm)
connect.close()

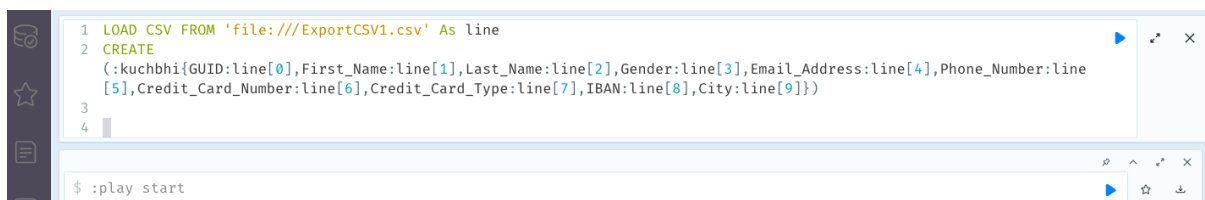
```

Now we take the run time of that query and append it in xm array which we can use later to export in csv file. Using this I have performed the query four times in four different processes.

NEO4J:

As I already had the application installed, so I just need to make node using the csv files data for that I have first created a database named “DBPT”.

In order to load data from csv files we need to use command:



```

1 LOAD CSV FROM 'file:///ExportCSV1.csv' As line
2 CREATE
  (:kuchbhi{GUID:line[0],First_Name:line[1],Last_Name:line[2],Gender:line[3],Email_Address:line[4],Phone_Number:line
  [5],Credit_Card_Number:line[6],Credit_Card_Type:line[7],IBAN:line[8],City:line[9]})
3
4

```

\$:play start

As we need to make four labels so we will change the name of the labels which is just after CREATE line, E.g., “kuchbhi”.

Now we will initiate the connection procedure:

```
51 conn = neo4j.GraphDatabase.driver("neo4j://localhost:7687", auth=('neo4j', '1234'))
52 db = conn.session()
53 for j in ('ft', 'kuchbhi', 'stf', 'tf'):
```

Once connected we will perform the queries for 31 times.

```
for j in ('ft', 'kuchbhi', 'stf', 'tf'):
    xn = []
    xn1 = []
    xn2 = []
    xn3 = []
    for i in range(0, 31):
        num = db.run("MATCH (n:"+j+") return n")
        time = num.consume()
        xn.append(time.result_available_after)
        num = db.run("MATCH (n:"+j+") where n.Credit_Card_Type='UnionPay' return n")
        summ = num.consume()
        xn1.append(int(summ.result_available_after))
        num = db.run("MATCH (c:"+j+") where c.City = 'Arlington' return c")
        summ = num.consume()
        xn2.append(int(summ.result_available_after))
        num = db.run("Match (u:"+j+") where u.Credit_Card_Number =3853-6481-4247-4031 return u")
        summ = num.consume()
        xn3.append(int(summ.result_available_after))
    n.append(xn)
    n1.append(xn1)
    n2.append(xn2)
    n3.append(xn3)
db.close()
```

Where xn stores the run time.

MARIADB:

First we need to make a container:

```
docker run --name mariadbproject -e MYSQL_ROOT_PASSWORD=1234 -p 3306:3306
-d docker.io/library/mariadb:10.3
```

Now we will import the csv file to mariadb folder:

In every step we change the name of csv file along with table name.

```
load data local infile 'bin/ExportCSV1.csv' into table db100 fields terminated by ',' ignore 1 lines;
```


Now we go inside the container using the credentials:

```
docker exec -it mariadbproject bash
```

```
mariadb -u root -p
```

```
enter password:
```

Now we will create the database “Lost_credit_card” using command :

```
Create database Lost_credit_card;
```

Now we need to create four tables to import the data of the four csv files :

```
create table db100 (GUID int,First_Name varchar(150),Last_Name varchar(50),Gender varchar(100) ,  
Email_Address varchar(500), Phone_Number int,Credit_Card_Number int,Credit_Card_Type  
varchar(100),IBAN int,City varchar(50),primary key(GUID));
```

We will just change the name of tables like db100, db75 etc, rest will remain same.

As we are finished with mariadb now we will connect to python:

```
77  
78 conn = mysql.connector.connect(user='root',password='1234',port=3306)  
79 curr = conn.cursor()  
80 num_4_sql = 0
```

```

85     curr.execute("use Lost_credit_card;")
86     for j in ('db100', 'db75', 'db50', 'db25'):
87         curr.execute("set profiling = 1;")
88         for i in range(0,31):
89             curr.execute("select * from "+j)
90             curr.fetchall()
91             curr.execute("show profiles;")
92             cm = curr.fetchall()
93             if num_4_sql < 14:
94                 xma.append(cm[i][1])
95             elif num_4_sql == 14:
96                 xma.append(cm[14][1])
97             else:
98                 xma.append(cm[14][1])
99             num_4_sql += 1

```

Now we perform the query 31 times:

```

for i in range(0,31):
    curr.execute("select * from "+j)
    curr.fetchall()
    curr.execute("show profiles;")
    cm = curr.fetchall()
    if num_4_sql < 14:
        xma.append(cm[i][1])
    elif num_4_sql == 14:
        xma.append(cm[14][1])
    else:
        xma.append(cm[14][1])
    num_4_sql += 1
for i in range(0,31):
    curr.execute("select * from "+j+" where Credit_Card_Type = 'UnionPay'")
    curr.fetchall()
    curr.execute("show profiles;")
    cm = curr.fetchall()

    xma1.append(cm[14][1])
    num_4_sql += 1

```

cm gives us the time it took for the query. This way we can run queries for different data set.

Now we have runned all the queries for different data sets, now we can export it to csv file to analyze. Where m is for mongodb, n for neo4j, ma for mariadb.

```

157
158 with open('Final runtime.csv','w',newline='') as file:
159     mywrite = csv.writer(file, delimiter=',')
160     mywrite.writerow(m)
161     mywrite.writerow(m1)
162     mywrite.writerow(m2)
163     mywrite.writerow(m3)
164     mywrite.writerow(n)
165     mywrite.writerow(n1)
166     mywrite.writerow(n2)
167     mywrite.writerow(n3)
168     mywrite.writerow(ma)
169     mywrite.writerow(ma1)
170     mywrite.writerow(ma2)
171     mywrite.writerow(ma3)

```

DATABASE PERFORMANCE:

First Query:

MongoDB:

```

mydoc = csv_db.find({}).explain()
q1 = mydoc['executionStats']['executionTimeMillis']
xm.append(q1)

```

Neo4j:

```

num = db.run("MATCH (n:"+j+") return n")
time = num.consume()

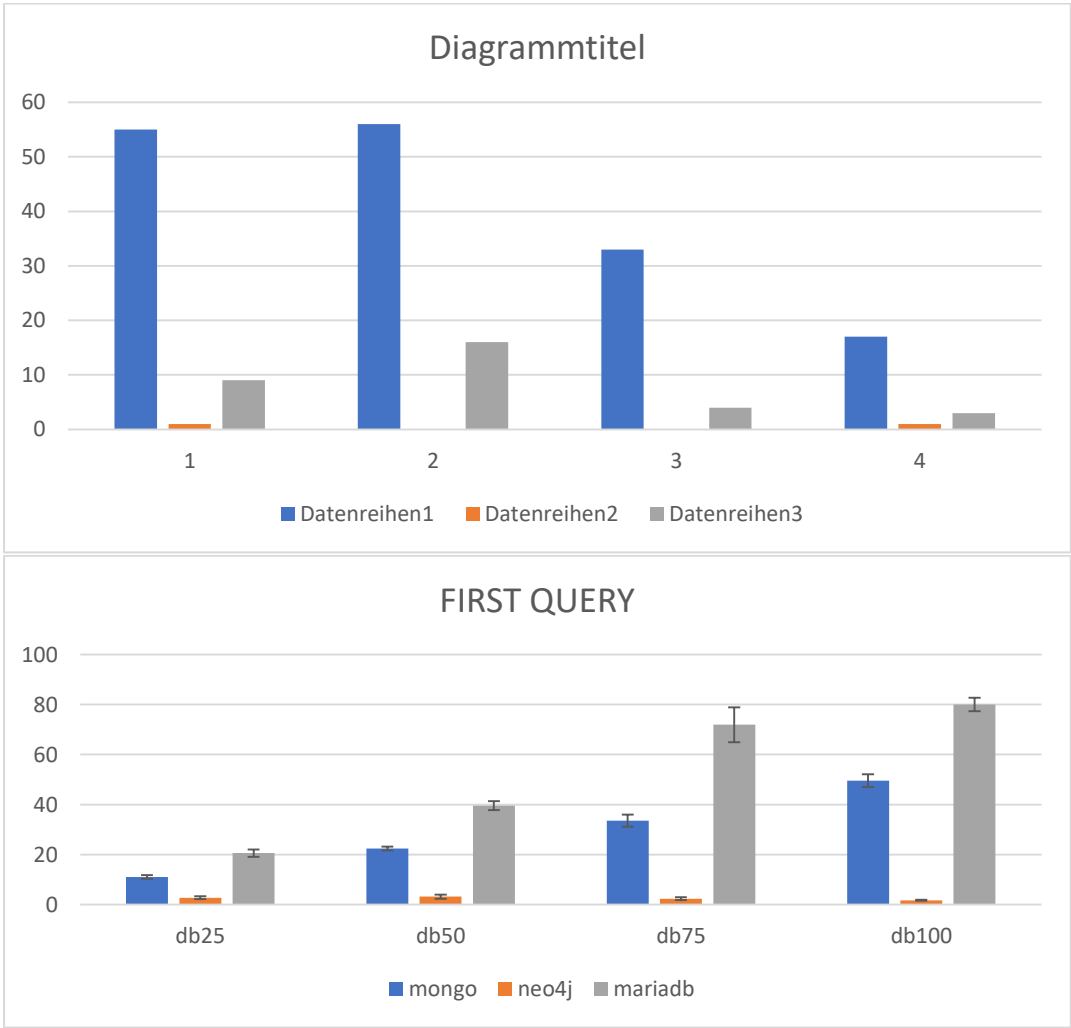
```

MariaDB:

```
for i in range(0,31):
    curr.execute("select * from "+j)
    curr.fetchall()
    curr.execute("show profiles;")
```

Performance:

	Mean			Standard Deviation			Confidence at 95%		
	MongoDB	Neo4j	MariaDB	MongoDB	Neo4j	MariaDB	MongoDB	Neo4j	MariaDB
db25	44,133333	0,5666667	24,290323	3,2836294	0,5587685	2,4212026	1,1750101	0,1999491	0,8664003
db50	40,9	0,7666667	18,677419	1,8138357	0,4229526	2,7292653	0,6490608	0,1513489	0,9766371
db75	23,866667	0,4	14,193548	1,8926759	0,4898979	2,6923761	0,6772729	0,1753045	0,9634367
db100	11,433333	0,6	6,6774194	0,8825468	0,4898979	1,6428295	0,3158095	0,1753045	0,5878682



Second Query:

MongoDB:

```
mydoc = csv_db.find({"Credit Card Type":"UnionPay"}).explain()
q2 = mydoc['executionStats']['executionTimeMillis']
```

Neo4j:

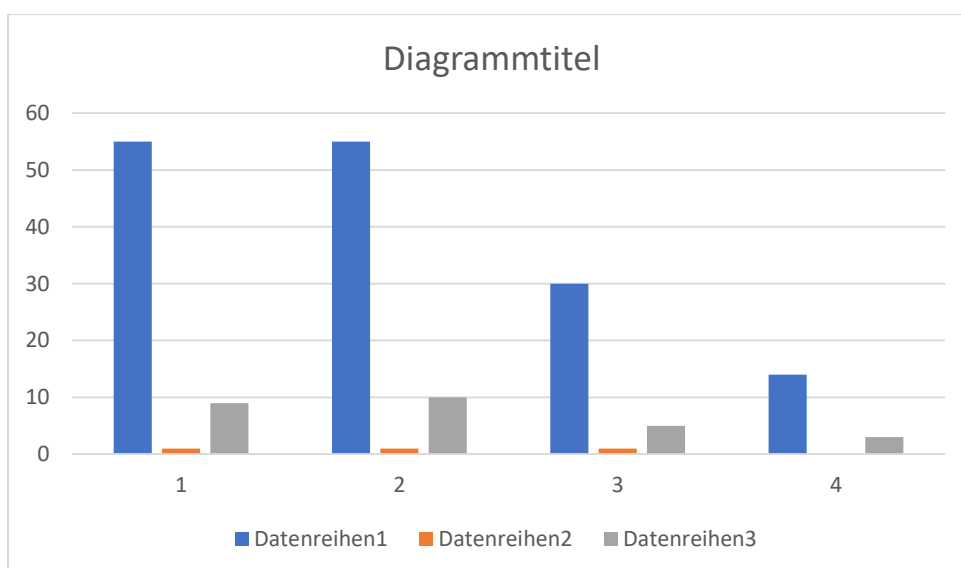
```
xn.append(time.result_available_after)
num = db.run("MATCH (n:"+j+") where n.Credit_Card_Type='UnionPay' return n")
```

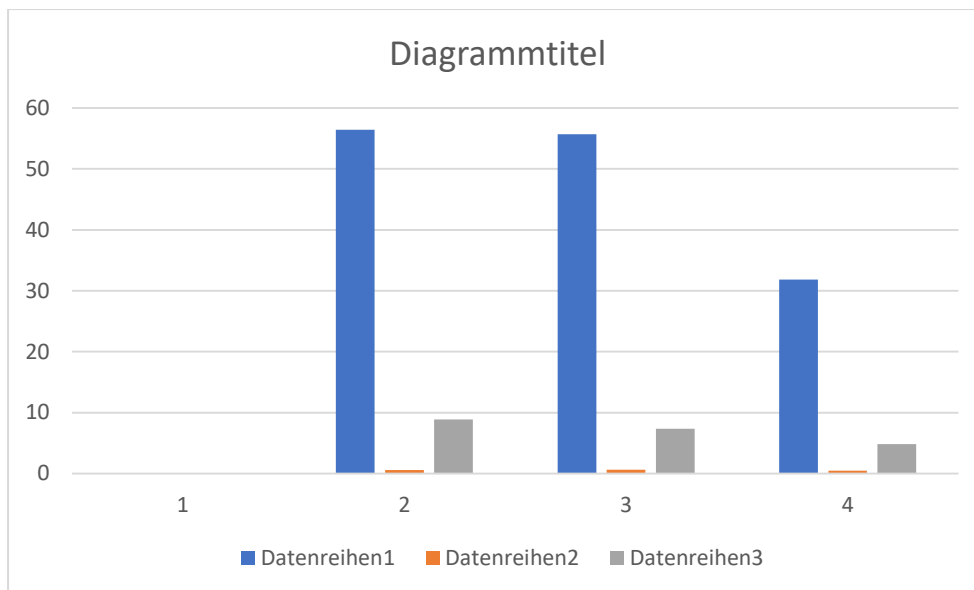
MariaDB:

```
for i in range(0,31):
    curr.execute("select * from "+j+" where Credit_Card_Type = 'UnionPay'")
    curr.fetchall()
```

Performance:

db25	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
db50	56,43333	0,566667	8,9	1,745152	0,495536	0,943398	0,624483	0,177322	0,337584
db75	55,7	0,633333	7,366667	2,253146	0,481894	1,328742	0,806263	0,172441	0,475476
db100	31,86667	0,5	4,833333	2,940899	0,5	0,968963	1,052368	0,178919	0,346733





Third Query:

MongoDB:

```
query3 = csv_db.find({'City':'Arlington'})
query3_time = query3.explain()
```

Neo4j:

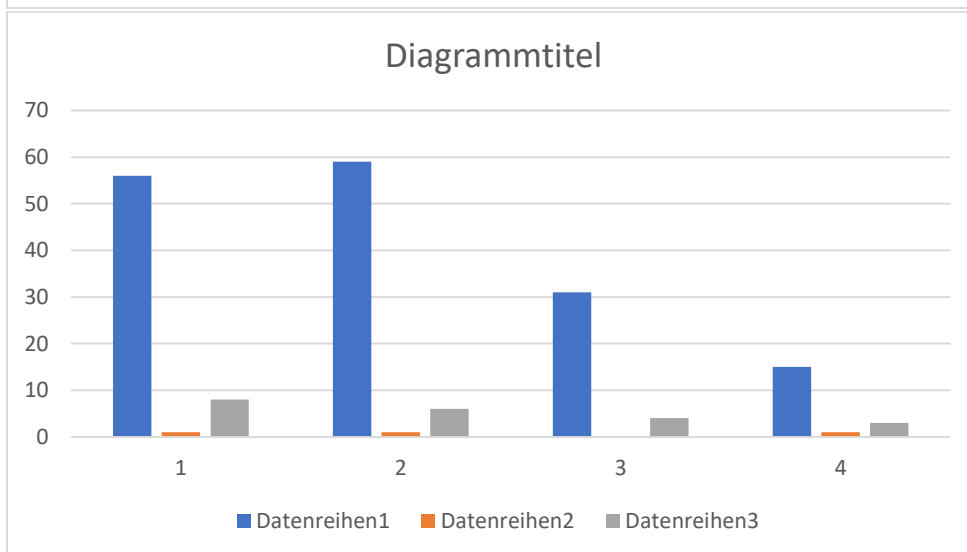
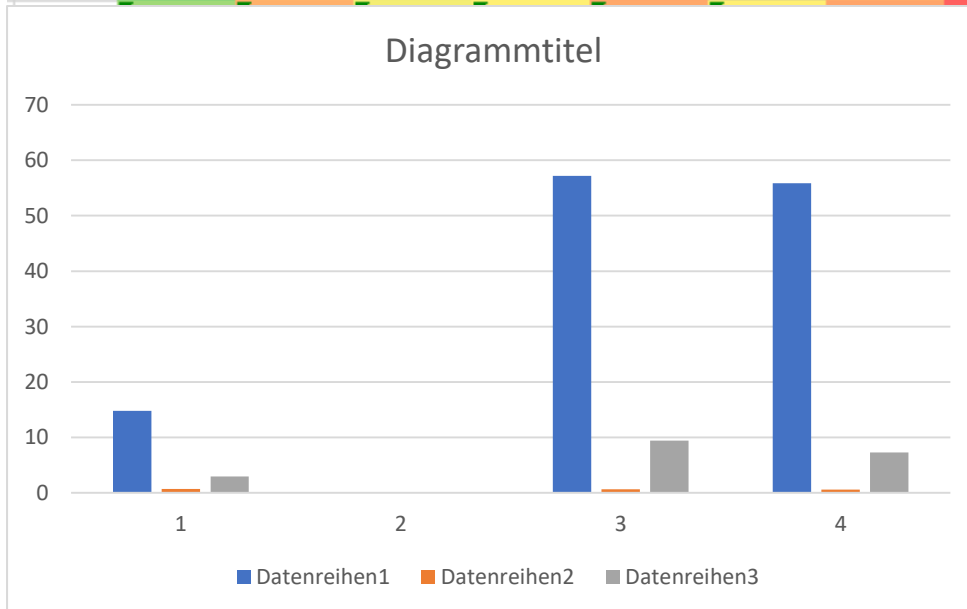
```
xn1.append(int(summ.result_available_after))
num = db.run("MATCH (c:"+j+") where c.City = 'Arlington' return c")
```

MariaDB:

```
for i in range(0,31):
    curr.execute("select * from "+j+" where City = 'Arlington'")
    curr.fetchall()
```

Performance:

db25	14,8	0,724138	3	1,301281	1,310345	1	0,465649	0,468892	0,357839
db50	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
db75	57,2	0,666667	9,433333	3,197916	0,471405	1,453349	1,144339	0,168687	0,520065
db100	55,86667	0,6	7,3	1,687865	0,489898	#DIV/0!	0,603984	0,175305	#DIV/0!



Fourth Query:

MongoDB:

```
query4_codes = csv_db.find({'Credit Card Number':3853-6481-4247-4031})
query4_time = query4_codes.explain()
```

Neo4j:

```

xn2.append(int(summ.result_available_after))
num = db.run("Match (u:"+j+") where u.Credit_Card_Number =3853-6481-4247-4031 return u")
summ = num.consume()

```

MariaDB:

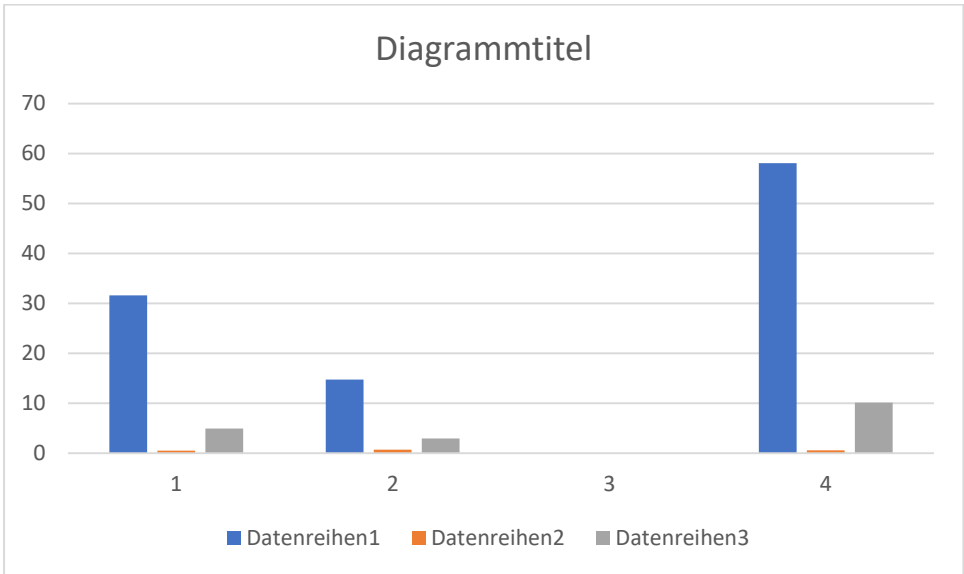
```

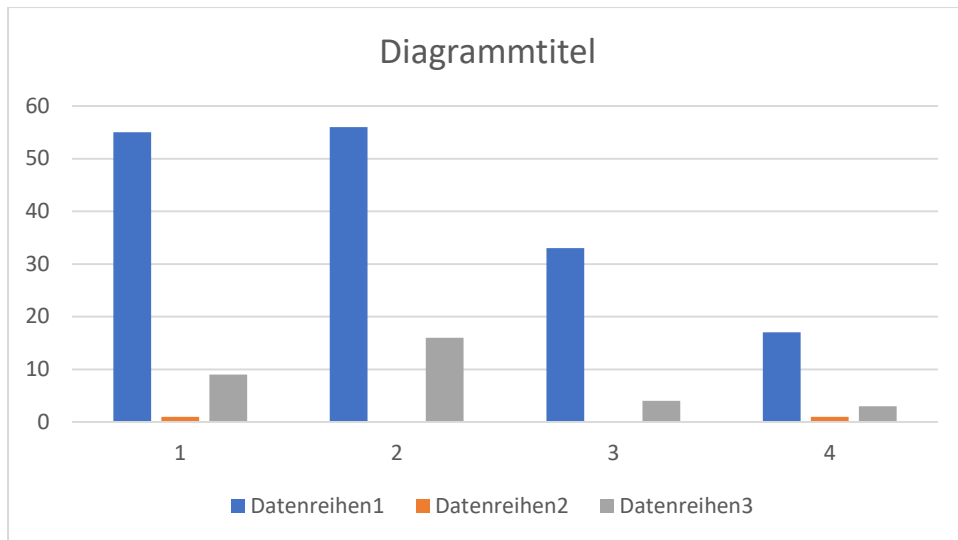
for i in range(0,51):
    curr.execute("select * from "+j+" where Credit_Card_Number=3853-6481-4247-4031")
    curr.fetchall()

```

Performance:

db25	31,6	0,533333	4,966667	1,356466	0,498888	0,835996	0,485396	0,178521	0,299152
db50	14,76667	0,766667	3	1,085766	0,760847	0,68313	0,388529	0,272261	0,24445
db75	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
db100	58,06667	0,633333	10,13333	7,11774	0,481894	1,49963	2,547004	0,172441	0,536626





RECAPICULATING:

If we will have a look at the graphs, the overall time consumption of MongoDB is the most among three and neo4j is the least, whereas MariaDB in middle of both.

There is a slight variation in the First Query where for db75 and db100 time consumption is maximum for MariaDB.

This happens because the complexity of the mariadb query in db75 and db100, is more than as of MongoDB so the time reduces.

So, we can make an analysis that even though we perform same queries at these three databases the performance of neo4j is the best in short as well as long run.

THE END