DATA BASE FOR HOSPITAL analysis report

The runtime analysis of neo4j, mongodb, mariadb ,Cassandra and redis

BY;

Arkala chandhra shekar jyothi vinay

Matriculation No. 521486.

Peram Divya

Matriculation No. 522137

introduction

In this analysis, the data is generated with the https://www.onlinedatagenerator.com/. Though it is a fake data it is matched with the real data as much as possible. With the help of this we generate a fake data with the same aspect as the original in a .csv format. The five data bases that we are going to use to perform analysis are NEO4J, MONGODB,MARIADB,REDDIS AND CASSANDRA. For the analysis we split the data in 4 different csv files with increasing number of data (25000,50000,75000,100000), and perform query that gives same output for the FIVE-database system with increasing complexity.

Brief introduction on database we are using.

Neo4j:

Neo4j 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.

MongoDB:

MongoDB is a document oriented NoSql type DBMS. MongoDB moves away from the traditional table-based structure of relational databases in favour of documents in “Json” style with dynamic scheme called “Bson”. 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'.

MariaDB:

MariaDB is an open-source relational database management system (RDBMS). A relational database organizes data into data tables in which data types may be related to each other. These relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. Mariadb leverages relational theory for creating tables with attributes (columns) and records (lines) for data storage. Each record must be identified with a primary key, which can be an attribute or a set of attributes. To implement the relationships between the tables, foreign keys are used, which connect a set of attributes (X) from table A with a set of attributes (Y) from table B. With the evolution to ORDBMS, Mariadb has introduced the ability to create datatypes much more complex than the basic ones, necessary for many fields of study and working.

Redis:

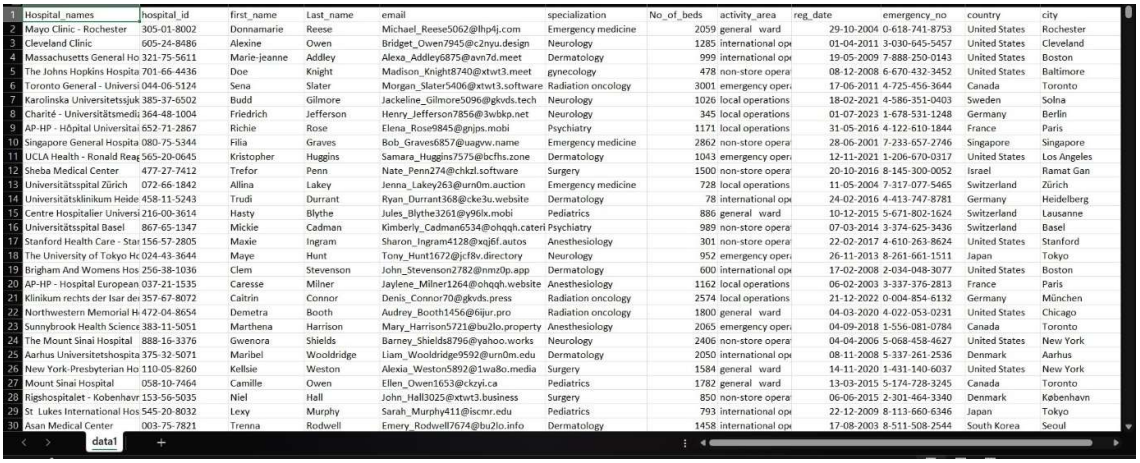
Redis is an open-source, in-memory data structure store used as a database, cache, and message broker. It's known for its high performance, flexibility, and support for various data structures like strings, lists, sets, hashes, and more. Redis is often used in web applications for caching frequently accessed data, managing session states, real-time analytics, messaging queues, and other use cases requiring fast data access and manipulation.

Cassandra:

Cassandra is a highly scalable, distributed NoSQL database system designed to handle large volumes of data across multiple commodity servers while providing high availability and fault tolerance. It is optimized for write-heavy workloads and offers flexible data modeling with a distributed architecture that allows for linear scalability. Cassandra is used in applications where high availability, linear scalability, and fault tolerance are critical requirements, such as in real-time analytics, IoT applications, and large-scale web services.

2. Data Model

As mentioned, before we are going to use a fake generated data for the analysis the data consists of mainly four table hospital,person,specialization,service.



	Hospital_names	hospital_id	first_name	Last_name	email	specialization	No_of_beds	activity_area	reg_date	emergency_no	country	city
2	Mayo Clinic - Rochester	305-01-8002	Donnamarie	Reese	Michael_Reese5062@lhp4j.com	Emergency medicine	2059	general ward	29-10-2004	0-618-741-8753	United States	Rochester
3	Cleveland Clinic	605-24-8486	Alexine	Owen	Bridget_Owen7945@c2nyu.design	Neurology	1285	international ope	01-04-2011	3-030-645-5457	United States	Cleveland
4	Massachusetts General Ho	321-75-5611	Marie-jeanne	Addley	Alexa_Addley6875@avn7d.meet	Dermatology	999	international ope	19-05-2009	7-888-250-0143	United States	Boston
5	The Johns Hopkins Hospita	701-66-4436	Doe	Knight	Madison_Knight8740@xtwt3.meet	gynecology	478	non-store operat	08-12-2008	6-670-432-3452	United States	Baltimore
6	Toronto General - Universi	044-06-5124	Sena	Slater	Morgan_Slater5406@xtwt3.software	Radiation oncology	3001	emergency opera	17-06-2011	4-725-456-3644	Canada	Toronto
7	Karolinska Universitetssjuk	385-37-6502	Budd	Gilmore	Jackeline_Gilmore5096@gkvds.tech	Neurology	1026	local operations	18-02-2021	4-586-351-0403	Sweden	Solna
8	Charité - Universitätsmedi	364-48-1004	Friedrich	Jefferson	Henry_Jefferson7856@3wbkp.net	Neurology	345	local operations	01-07-2023	1-678-531-1248	Germany	Berlin
9	AP-HP - Hôpital Universitai	652-71-2867	Richie	Rose	Elena_Rose9845@gnjps.mobi	Psychiatry	1171	local operations	31-05-2016	4-122-610-1844	France	Paris
10	Singapore General Hospita	080-75-5344	Filia	Graves	Bob_Graves6857@uagvw.name	Emergency medicine	2862	non-store operat	28-06-2001	7-233-657-2746	Singapore	Singapore
11	UCLA Health - Ronald Reag	565-20-0645	Kristopher	Huggins	Samara_Huggins7575@bcfhs.zone	Dermatology	1043	emergency opera	12-11-2021	1-206-670-0317	United States	Los Angeles
12	Sheba Medical Center	477-27-7412	Trefor	Penn	Nate_Penn274@chkzl.software	Surgery	1500	non-store operat	20-10-2016	8-145-300-0052	Israel	Ramat Gan
13	Universitätsspital Zürich	072-66-1842	Allina	Lakey	Jenna_Lakey263@urn0m.auction	Emergency medicine	728	local operations	11-05-2004	7-317-077-5465	Switzerland	Zürich
14	Universitätsklinikum Heide	458-11-5243	Trudi	Durrant	Ryan_Durrant368@cke3u.website	Dermatology	78	international ope	24-02-2016	4-413-747-8781	Germany	Heidelberg
15	Centre Hospitalier Universi	216-00-3614	Hasty	Blythe	Jules_Blythe3261@y96lx.mobi	Pediatrics	886	general ward	10-12-2015	5-671-802-1624	Switzerland	Lausanne
16	Universitätsspital Basel	867-65-1347	Mickie	Cadman	Kimberly_Cadman6534@ohqqh.cateri	Psychiatry	989	non-store operat	07-03-2014	3-374-625-3436	Switzerland	Basel
17	Stanford Health Care - Stan	156-57-2805	Maxie	Ingram	Sharon_Ingram4128@xqj6f.autos	Anesthesiology	301	non-store operat	22-02-2017	4-610-263-8624	United States	Stanford
18	The University of Tokyo Ho	024-43-3644	Maye	Hunt	Tony_Hunt1672@jcf8v.directory	Neurology	952	emergency opera	26-11-2013	8-261-661-1511	Japan	Tokyo
19	Brigham And Womens Hos	256-38-1036	Clem	Stevenson	John_Stevenson2782@nmz0p.app	Dermatology	600	international ope	17-02-2008	2-034-048-3077	United States	Boston
20	AP-HP - Hospital European	037-21-1535	Caresse	Milner	Jaylene_Milner1264@ohqqh.website	Anesthesiology	1162	local operations	06-02-2003	3-337-376-2813	France	Paris
21	Klinikum rechts der Isar der	357-67-8072	Caitrin	Connor	Denis_Connor70@gkvds.press	Radiation oncology	2574	local operations	21-12-2022	0-004-854-6132	Germany	München
22	Northwestern Memorial H	472-04-8654	Demetra	Booth	Audrey_Booth1456@6ijur.pro	Radiation oncology	1800	general ward	04-03-2020	4-022-053-0231	United States	Chicago
23	Sunnybrook Health Science	383-11-5051	Marthena	Harrison	Mary_Harrison5721@bu2lo.property	Anesthesiology	2065	emergency opera	04-09-2018	1-556-081-0784	Canada	Toronto
24	The Mount Sinai Hospital	888-16-3376	Gwenora	Shields	Barney_Shields8796@yahoo.works	Neurology	2406	non-store operat	04-04-2006	5-068-458-4627	United States	New York
25	Aarhus Universitetshospita	375-32-5071	Maribel	Wooldridge	Liam_Wooldridge9592@urn0m.edu	Dermatology	2050	international ope	08-11-2008	5-337-261-2536	Denmark	Aarhus
26	New York-Presbyterian Ho	110-05-8260	Kellsie	Weston	Alexia_Weston5892@1wa8o.media	Surgery	1584	general ward	14-11-2020	1-431-140-6037	United States	New York
27	Mount Sinai Hospital	058-10-7464	Camille	Owen	Ellen_Owen1653@ckzyi.ca	Pediatrics	1782	general ward	13-03-2015	5-174-728-3245	Canada	Toronto
28	Rigshospitalet - Kobenhavr	153-56-5035	Niel	Hall	John_Hall3025@xtwt3.business	Surgery	850	non-store operat	06-06-2015	2-301-464-3340	Denmark	København
29	St Lukes International Hos	545-20-8032	Lexy	Murphy	Sarah_Murphy411@iscmr.edu	Pediatrics	793	international ope	22-12-2009	8-113-660-6346	Japan	Tokyo
30	Asan Medical Center	003-75-7821	Trenna	Rodwell	Emery_Rodwell7674@bu2lo.info	Dermatology	1458	international ope	17-08-2003	8-511-508-2544	South Korea	Seoul

data1

3. Implementation

For the start of the project, we first need to have the access to the five-database system. For this we are using docker to create a container for the databases.

docker run -d --name mongodb -p 27017:27017 mongo

docker run -d --name neo4jdb -p 7474:7474 -p7687:7687 -e neo4j_auth=neo4j/password neo4j docker run -d --name mariadb -p 3306:3306 -e mysql_root_password=mypass mariadb

docker run -d --name redis -p 6379:6379 redis

docker run -d --name cassandra -p 9042:9042 cassandra:latest

This way we can create three sperate container for all the databases. With the port mentioned which we use to connect through python.

For this analysis we are using python code as it includes many libraries to interact with the three different databases.

```
1
2    import pandas
3    import pymongo
4    import csv
5    import neo4j
6    from cassandra.cluster import Cluster
7    import redis
8    import mysql.connector
```

We have imported libraires pymongo for mongodb, neo4j for neo4j database, MySQL for mariadb, cluster for Cassandra.cluster, redis and we also used pandas and csv to read the csv data that we generated and to return the runtime of all the queries.

After creating container of databases and importing the libraries we can start connecting to the database and import the csv file of the data and checking the runtime by performing queries directly through python.

MongoDB:

First, we make the connection,

```
connect = pymongo.MongoClient("mongodb://localhost:27017")
database = connect["testdb"]
```

As we connect to testdb we actually create a new database in mongodb. After the connection we can start importing the data that we have in csv format.

```
for j in ('1','2','3','4'):
    xm=[]
    xm1=[]
    xm2=[]
    xm3=[]
    dbread = pandas.read_csv("C:/Users/vinay/Desktop/datacsv/data"+j+".csv")
    dbread = dbread.to_dict(orient="records")
    service_coll = database['service']
    hospital_coll = database['hospital']
    person_coll = database['person']
    specialization_coll = database['specialization']
```

```
    for row in dbread:
        service_coll.insert_one({'No_of_beds':row['No_of_beds'],'Hospital_id':row['Hospital_id']})
        hospital_coll.insert_one({'Hospital_names':row['Hospital_names'],'Hospital_id':row['Hospital_id'],
                                  'country':row['country'],'city':row['city'],'reg_date':row['reg_date'],'emergency_no':row['emergency_no']})
        person_coll.insert_one({'first_name':row['first_name'],'last_name':row['last_name'],'email':row['email'],'Hospital_id':row['Hospital_id']})
        specialization_coll.insert_one({'specialization':row['specialization'],'Hospital_id':row['Hospital_id'],'activity_area':row['activity_area']})
```

Here the csv file is read and the also we create the collection as mention in data. After that we just insert the values in collections.

Once we create the collections, we just need to perform the queries as many times we want, for the analysis we run it for 31 times.

```
for i in range(0,31):
    mydoc = person_coll.find({}).explain()
    m = mydoc['executionStats']['executionTimeMillis']
    xm.append(m)
    mydoc = hospital_coll.find({"country":"italy"}).explain()
    m1= mydoc['executionStats']['executionTimeMillis']
    xm1.append(m1)
```

In the above code we perform the query 31 time and take the run time of that query and append it in xm array which we can use later to export in csv file later. In this way we perform 4 queries and take their run time. The whole process should be done for 4 different data set of increasing number of data so we drop the collections so that it can be created with new data.

```
print(xm,xm1,xm2,xm3)
service_coll.drop()
hospital_coll.drop()
person_coll.drop()
specialization_coll.drop()
```

Neo4j:

Same as above mongodb we first connect to neo4j and then create node as on the data, add the data of csv file and make the query and take the run time. But in neo4j we are not able to directly load csv as it runs in the docker container and not handle system host files, so we have to copy the file in neo4jdb container.

docker cp E:/database/data/ExportCSV250 neo4jdb:/var/lib/neo4j/import

After we are done with this, we also have to reset the password of the neo4j as the docker container set it as default do we run local localhost:7687 in browser, through which we can change it. Once all this is done, we just need to do as we did in mongodb

```
connect = neo4j.GraphDatabase.driver("neo4j://localhost:7687",auth=('neo4j','password'))
database = connect.session()
```

Connecting to neo4j.

creating constraints for the required fieldsAfter creating constraints, we import data from csv file. Then we create relations so that they can be linked as show in the data we create three relations (person to hospital, no of beds to hospital, specialization to hospital).

```
database.run("create constraint hospital_ids_db for (c:hospital) require c.hospital_id is unique")
database.run("create constraint person_ids_db for (c:person) require c.email is unique")
database.run("create constraint services_ids_db for (c:services) require c.no_of_beds is unique")
database.run("create constraint specialization_ids_db for (c:specialization) require c.specialization_info is unique")

'''load csv with headers from 'file:///datacsv/data"+i+".csv' as line FIELDTERMINATOR',' with line create (b:hospital{hospital_id: tointeger(line.hospital_id)}) merge (

database.run("load csv with headers from 'file:///datacsv/data" +i+".csv' as line FIELDTERMINATOR',' with line create (b:hospital{hospital_id: tointeger(line.hospital_i
database.run("load csv with headers from 'file:///datacsv/data"+i+".csv'as line FIELDTERMINATOR',' with line match (c:hospital {hospital_id: tointeger(line.hospital_id)}
```

Now we can perform the queries;

```
    for i in range(0,31):
        num = database.run("MATCH (n:person) return n")
        time = num.consume()
        xn.append(int(time.result_available_after))
```

xn take the runtime of the query. Now we can just perform the same with different data set and with different queries.

```
print(xn,xn1,xn2,xn3)
database.run("match (n) call {with n optional MATCH (n)-[r]-() DELETE n,r} in transactions of 2000 rows")
database.run("drop constraint hospital_ids_db")
database.run("drop constraint person_ids_db")
database.run("drop constraint services_ids_db")
database.run("drop constraint specialization_ids_db")
```

After we are done, we delete the data and drop the constraints.

Mariadb:

As same we connect add the data and then make query

```
connect = mysql.connector.connect(user='root',password='mypass',port=3306)
mariadb_cursor =connect.cursor()
```

connecting to mariadb

```
mariadb_cursor.execute('create database projdb')
mariadb_cursor.execute('use projdb')

for j in ('1', '2', '3', '4'):
    xm = []
    xm1 = []
    xm2 = []
    xm3 = []

    dbread = pandas.read_csv("C:/Users/vinay/Desktop/datacsv/data" + j + ".csv")
    dbread = dbread.itertuples()
    mariadb_cursor.execute("create table hospital(hospital_name varchar(100),hospital_id int,country varchar(100),city varchar(150),reg_date int,emergency_no int,primary
    mariadb_cursor.execute("create table person (first_name varchar(150),last_name varchar(150),email varchar(150),Hospital_id int,primary key(email),foreign key(hospital
    mariadb_cursor.execute("create table service(No_of_beds int,Hospital_id int , primary key(hospital_id),foreign key(hospital_id) references hospital(hospital_id) ) ")
    mariadb_cursor.execute("create table specialization (specialization varchar(150),Hospital_id int,activity_area varchar(150),primary key(hospital_id),foreign key(hospi
```

After connection we create database projdb, read the data in csv using pandas and then create tables as show in data. Then we load the data in the tables we created.

```
for rows in dbread:
    insert_query = ("insert into hospital(hospital_name,hospital_id,country,city,reg_date,emergency_no) values('"+rows.hospital_name+"',"+str(rows.hospital_id)+",'"+row
    mariadb_cursor.execute(insert_query)
    insert_query2 =("insert into person (email,first_name,last_name,hospital_id)values('"+rows.email+"','"+rows.first_name+"','"+rows.last_name+"',"+str(rows.hospital_i
    mariadb_cursor.execute(insert_query2)
    insert_query3 =("insert into service (hospital_id,No_of_beds) values ("+str(rows.No_of_beds)+","+str(rows.hospital_id)+")")
    mariadb_cursor.execute(insert_query3)
    insert_query4 = ("insert into specialization (hospital_id, activity_area,  specialization) values ("+str(rows.hospital_id)+",'"+rows.activity_area+"','"+rows.specia
    mariadb_cursor.execute(insert_query4)
connect.commit()
```

Now we just perform the queries

```
    for i in range(0,31):
        mariadb_cursor.execute("select*from person")
        mariadb_cursor.fetchall()
        mariadb_cursor.execute("show profiles;")
        cm = mariadb_cursor.fetchall()
        xm.append(cm[14][1])
```

xm gives us the time it took for the query. This way we can run queries for different data set. After we are done, we can need to delete the database.

```
mariadb_cursor.execute('drop database projdb')
```

After we are done with running all the queries for different data sets, we can export it to a csv file to analyze

Redis:

Same as above mongodb we first connect to redis and then create node as on the data, add the data of csv file and make the query and take the run time

```
# Connect to Redis
redis_client = redis.StrictRedis(host='localhost', port=6379, db=0)
#redis_client = redis.StrictRedis(host='localhost', port=6379, db=0)

for j in ('1', '2', '3', '4'):
    xm = []
    xm1 = []
    xm2 = []
    xm3 = []
    dbread = pandas.read_csv(f"C:/Users/vinay/Desktop/datacsv/data{j}.csv")
    dbread = dbread.to_dict(orient="records")
```

After creating constraints, we import data from csv file. Then we create relations so that they can be linked as show in the data we create three relations (person to hospital, no of beds to hospital , specialization to hospital).

```
    for row in dbread:
        # Use Redis hash to store data
        redis_client.hset(f"service:{row['Hospital_id']}", "No_of_beds", row['No_of_beds'])
        redis_client.hset(f"hospital:{row['Hospital_id']}", "Hospital_names", row['Hospital_names'])
        redis_client.hset(f"hospital:{row['Hospital_id']}", "country", row['country'])
        redis_client.hset(f"person:{row['Hospital_id']}", "first_name", row['first_name'])
        redis_client.hset(f"specialization:{row['Hospital_id']}", "specialization", row['specialization'])
```

Once we create the collections, we just need to perform the queries as many times we want, for the analysis we run it for 31 times

After creating constraints, we import data from csv file. Then we create relations so that they can be linked as show in the data we create three relations (person to hospital, no of beds to hospital , specialization to hospital).

```
for i in range(0,31):
    start_time = time.time()
    # Query 1: Find all persons
    persons=[]
    for keys in redis_client.keys("person:*"):
        persons.append(redis_client.hgetall(keys))
    #print("Query 1 - All Persons:", persons)
```

In this way we perform 4 queries and take their run time. The whole process should be done for 4 different data set of increasing number of data so we drop the collections so that it can be created with new data.

```
# Clear Redis data at the end of each iteration (you may adjust this based on your use case)
redis_client.flushdb()

print(xm, xm1, xm2, xm3)
```

Cassandhra :

Same as above mongodb we first connect to cassandhra and then create node as on the data, add the data of csv file and make the query and take the run time

Connecting cassandhra from cassandhra custer

```
# Connect to Cassandra
cluster = Cluster([''])
session = cluster.connect()
```

creating constraints for the required fields

After creating constraints, we import data from csv file. Then we create relations so that they can be linked as show in the data we create three relations (person to hospital, no of beds to hospital , specialization to hospital).

```
# Load data from CSV files
for j in ('1', '2', '3', '4'):
    # Create tables
    session.execute("CREATE TABLE IF NOT EXISTS service (No_of_beds int, Hospital_id text, PRIMARY KEY (Hospital_id))")
    session.execute("CREATE TABLE IF NOT EXISTS hospital (Hospital_id text, Hospital_names text, country text, city text, reg_date text, emergency_no text, PRIMARY KEY (H
    session.execute("CREATE TABLE IF NOT EXISTS person (first_name text, last_name text, email text, Hospital_id text, PRIMARY KEY (Hospital_id, email))")
    session.execute("CREATE TABLE IF NOT EXISTS specialization (specialization text, Hospital_id text, activity_area text, PRIMARY KEY (Hospital_id, specialization))")
    dbread = pandas.read_csv("C:/Users/vinay/Desktop/datacsv/data"+j+".csv")
    dbread = dbread.to_dict(orient="records")
    for row in dbread:
        session.execute(f"INSERT INTO service (No_of_beds, Hospital_id) VALUES ({row['No_of_beds']}, '{row['Hospital_id']}')")
        session.execute(f"INSERT INTO hospital (Hospital_id, Hospital_names, country, city, reg_date, emergency_no) VALUES ('{row['Hospital_id']}', '{row['Hospital_names'
        session.execute(f"INSERT INTO person (first_name, last_name, email, Hospital_id) VALUES ('{row['first_name']}', '{row['last_name']}', '{row['email']}', '{row['Hos
        session.execute(f"INSERT INTO specialization (specialization, Hospital_id, activity_area) VALUES ('{row['specialization']}', '{row['Hospital_id']}', '{row['activi
```

Once we create the collections, we just need to perform the queries as many times we want, for the analysis we run it for 31 times.

```
    for i in range(0, 31):
        start_time = time.time()
        rows = session.execute("SELECT * FROM person")
        xm.append(int((time.time() - start_time) * 1000))
```

In this way we perform 4 queries and take their run time. The whole process should be done for 4 different data set of increasing number of data so we drop the collections so that it can be created with new data.

```
    # Drop tables
    session.execute("DROP TABLE IF EXISTS service")
    session.execute("DROP TABLE IF EXISTS hospital")
    session.execute("DROP TABLE IF EXISTS person")
    session.execute("DROP TABLE IF EXISTS specialization")

# Close the Cassandra connection
cluster.shutdown()
```

problems

When implementing this code there are many problems as we need to connect five different databases, we get an error if we try to connect all at the same time and then perform the queries. For this, it is much better to connect one database system at time and perform all the queries we need the close so we can connect to the other data base. The other problem that arises is that it is necessary to copy the files inhe neo4j container as it cannot be imported directly, so we need to copy the files first through accessing the terminal. The other problem is that, as we take time directly from the system it sometimes need to be converted in millisecond. And as for mariadb it is really hard to fetch the time so it takes lot effort to get the time from the system as it shows all the queries performed in list at max 14 queries, so we have to make a separate array to get all and then divide it later. And then convert them in millisecond same for redis. More on Cassandra took lot of time compared to any other database .

4. Database Analysis

Now that the data is loaded and queries are performed, we can have the data of the run time let's see the performance of the databases

Query 1,

The first query is the simple one where we just check the persons and their data.

MongoDB:

```
mydoc = person_coll.find({}).explain()
```

Neo4j:

```
mydoc = hospital_coll.find({"country":"italy"}).explain()
m1= mydoc['executionStats']['executionTimeMillis']
```

MariaDB:

```
mariadb_cursor.execute("select*from person")
mariadb_cursor.fetchall()
```

Cassandhra:

```
rows = session.execute("SELECT * FROM person")
```

Redis:

```
for keys in redis_client.keys("person:*"):
    persons.append(redis_client.hgetall(keys))
```

query 1	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis
first value	68	11	221	531	26.524	22	19	401	181	25.763	16	31	575	171	38.081	13	40	755	135	51.003
mean	4.93548	12.1935	206.74194	120.4193548	17.0856774	2.09677	16.9355	480.1613	60	25.52974	1.93548	24.0645	556.2903	85.58065	38.19428	1.45161	32.9032	724.2903	64.96774	57.63429
standard deviation	11.7527	4.75476	40.576465	83.89885392	4.936141554	3.69267	2.28509	133.7755	26.09783	0.320695	2.62661	2.82769	47.51108	25.55674	0.137675	2.16778	4.83505	24.27423	17.92004	13.7767
confidence level	4.20556	1.70144	40.576465	83.89885392	1.766343112	1.32138	0.81769	133.7755	26.09783	0.114752	0.9399	1.01186	47.51108	25.55674	0.049265	0.77572	1.73017	24.27423	17.92004	4.929839

Query 2,

The second query is about the company, where we check the company data, which are registered in Italy.

MongoDB:

```
mydoc = hospital_coll.find({"country":"italy"}).explain()
m1= mydoc['executionStats']['executionTimeMillis']
```

Neo4j:

```
num = database.run("MATCH (n:hospital) where n.country='Italy' return n")
time = num.consume()
xn1.append(int(time.result_available_after))
```

MariaDB:

```
mariadb_cursor.execute("select*from person")
```

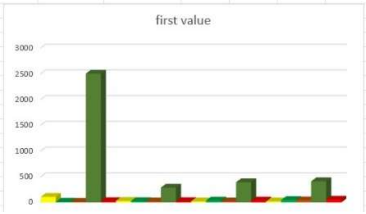
Cassandhra:

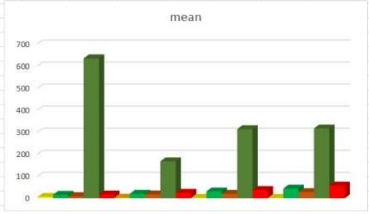
```
start_time = time.time()
rows = session.execute("SELECT * FROM hospital WHERE country='italy' ALLOW FILTERING")
```

Redis:

```
italy_hospitals = []
for key in redis_client.keys("hospital:*"):
    if redis_client.hget(key, "country") == b'Italy':
        italy_hospitals.append(redis_client.hgetall(key))
```

query 2	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis
first value	109	13	12	2501	25.014	31	26	20	289	25.135	21	37	21	393	37.376	20	47	40	411	54.194
mean	6.12903226	15.483871	10.225806	634.2903226	16.70374194	2.612903	20.35484	17.58065	167.8065	25.01174	1.903226	31.12903	20.35484	313.3548	37.50619	1.645161	44.09677	28.51613	316.6452	57.07619
standard deviation	18.8452817	6.69073007	3.1388274	396.158179	4.668124	5.301719	5.498084	3.367532	26.29537	0.143336	3.541048	4.316076	1.695314	24.30127	0.16559	3.450558	6.595966	3.749159	19.71498	13.24304
confidence level	6.74357352	2.39420301	1.1231943	141.7607788	1.670436024	1.897161	1.967428	1.205034	9.409504	0.051291	1.267125	1.54446	0.606649	8.695936	0.059255	1.234743	2.360293	1.341595	7.054786	4.738875





Query 3,

The third query gives us the data of hospitals that are registered as the specialization and Find hospitals with specialization in gynecology

MongoDB:

```
query3_hospital_id = []
for x in query3:
    query3_hospital_id.append(x['hospital_id'])
y= hospital_coll.find({'hospital_id':{'$in':query3_hospital_id}},{'Hospital_name':1,'hospital_id':1,'_id':0}).explain()
```

Neo4j:

```
num = database.run("MATCH (n:hospital) where n.country='Italy' return n")
time = num.consume()
xn1.append(int(time.result_available_after))
num = database.run("MATCH (s:specialization)-[r:sconnectedtoc]->(c:hospital) where s.specialization = 'Surgery' return c.Hospital_name, c.hospital_id, c.country")
time = num.consume()
xn2.append(int(time.result_available_after))
```

MariaDB:

```
start_time = time.time()
mariadb_cursor.execute("SELECT * FROM hospital WHERE country = 'italy'")
mariadb_cursor.fetchall()
```

Cassandhra:

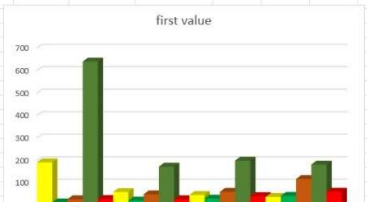
```
start_time = time.time()
rows = session.execute("SELECT * FROM hospital WHERE country='italy' ALLOW FILTERING")
```

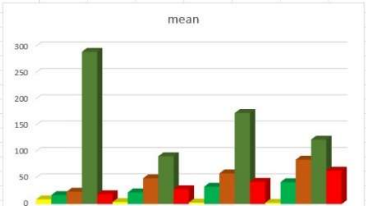
Redis:

```
gynecology_hospitals = []
for key in redis_client.keys("specialization:*"):
    if redis_client.hget(key, "specialization") == b'gynecology':
        hospital_id = key.decode().split(":")[1]
        hospital_info = redis_client.hgetall(f"hospital:{hospital_i
```

query 3

	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis
first value	192	14	28	640	29.453	60	22	50	173	28.083	47	31	61	200	41.964	38	43	118	182	61.273
mean	8.83870968	17.0322581	23.387097	289	18.571935	3.741935	21.93548	48.77419	90.70968	28.04797	2.903226	32.90323	58.29032	172.7097	47.04758	2.387097	41.51613	84.25806	122.3226	63.14793
standard deviation	33.4751682	7.03669585	5.8346983	94.26831986	5.40955768	10.32681	2.90645	13.13628	18.16092	0.109078	8.105573	4.387334	3.895353	19.00797	0.19752	6.553391	6.179419	15.53968	15.15806	13.8348
confidence level	11.978715	2.518003	2.0878816	33.73286517	1.9357497	3.695335	1.040041	4.700672	6.498682	0.039212	2.900489	1.569958	1.393909	6.801789	0.070636	2.345058	2.211236	5.560702	5.424144	4.950484





Query 4,

The fourth query gives us the data of hospitals that are registered , hospitals with beds between 500 and 1200

MongoDB:

```
for X in query4_codes:
    query4_hospital_id.append(X['hospital_id'])
query4_person = person_coll.find({'hospital_id':{'$in':query4_hospital_id}},{'first_name':1,'Last_name':1,'city':1,'_id':0}).explain()
query4_hospital = hospital_coll.find({'hospital_id':{'$in':query4_hospital_id}},{'country':1,'city':1,'_id':0}).explain()
m3 = query4_time['executionStats']['executionTimeMillis']
```

Neo4j:

```
xn2.append(int(time.result_available_after))
num = database.run("match (a:services)-[:aconnectedtoc]->(c:hospital)<-[:pconnectedtoc]-(p:person) where a.no_of_beds>1000 and a.no_of_beds<1200 return a.no_of_beds, c
time = num.consume()
xn3.append(int(time.result_available_after))

n.append(xn)
```

MariaDB:

```
start_time = time.time()
mariadb_cursor.execute("SELECT * FROM service WHERE No_of_beds > 500 AND No_of_beds < 1200")
mariadb_cursor.fetchall()
xm3.append(int((time.time() - start_time) * 1000))
```

Cassandhra:

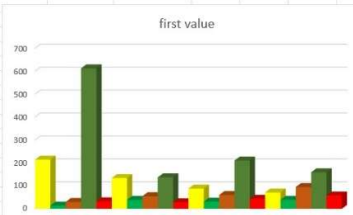
```
start_time = time.time()
rows = session.execute("SELECT * FROM service WHERE No_of_beds > 500 AND No_of_beds < 1200 ALLOW FILTERING")
xm3.append(int((time.time() - start_time) * 1000))
```

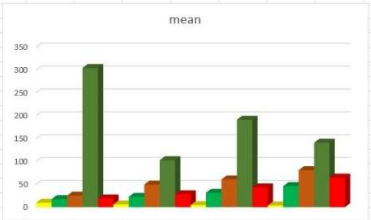
Redis:

```
bed_range_hospitals = []
for key in redis_client.keys("service:*"):
    beds = int(redis_client.hget(key, "No_of_beds"))
    if 500 < beds < 1200:
        hospital_id = key.decode().split(":")[1]
        hospital_info = redis_client.hgetall(f"hospital:{hospital_id}")
```

query 4

	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis	neo4j	mongodb	mariadb	casssandra	redis
first value	214	14	30	612	32.636	134	39	55	137	28.083	88	31	60	210	43.009	71	39	95	159	57.936
mean	9.74193548	17.5806452	25.193548	304.516129	18.98541935	5.967742	22.6129	49.41935	102.7742	28.04797	4.354839	31.54839	60.64516	191.3548	42.96765	3.645161	46.06452	80.96774	141.0645	64.64706
standard deviation	37.3604653	9.47926155	6.7078161	90.50071145	5.789602737	23.41971	4.701938	15.16102	12.3829	0.109078	15.32771	3.310187	12.10851	18.16865	0.458726	12.37205	7.79965	11.30938	15.32221	14.97586
confidence level	13.3690252	3.39204785	2.4003171	32.38466858	2.071744662	8.380482	1.682536	5.4252	4.431081	0.039032	5.484849	1.184513	4.332895	6.501449	0.16415	4.4272	2.791018	4.046937	5.482881	5.358935





5. Conclusion.

So, after analysing the different data sets of 25000,50000,75000,100000, and performing different complex queries, we can now make conclusions on the behaviour of the data base system. As we see from the graphs, we can understand that the overall time consumption of neo4j,mongodb is less and cassandra is the greatest whereas rest of are in middle of both. But still in some cases when compared to queries like in second and third query, even though the output is same the time MongoDB take is greater than MariaDB whereas for other queries the MariaDB is greater. This happens because the complicity of the MariaDB query in second and third. Cassandra took a lot of time for compiling the quries, is lesser than of MongoDB so the time reduces. So, we can make an analysis that even though we perform same queries at these five databases the performance of neo4j is the best in short as well as long run.