INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY (IJCET)

ISSN 0976 – 6367(Print)
ISSN 0976 – 6375(Online)
Volume 5, Issue 11, November (2014), pp. 11-22
© IAEME: www.iaeme.com/IJCET.asp
Journal Impact Factor (2014): 8.5328 (Calculated by GISI)
www.jifactor.com

IJCET

©IAEME

COMPARATIVE STUDY OF RELATIONAL AND NON-RELATIONS DATABASE PERFORMANCES USING ORACLE AND MONGODB SYSTEMS

Azhi Faraj[1], Bilal Rashid[2], Twana Shareef[3]

[1](Petroleum Engineering/ Koya University, Koysinjaq KOY45, Iraq)
[2, 3](Computer and statistics/ Sulaymanyah University, Sulaymaniyah, Iraq)

ABSTRACT

With the substantial growth in volume and complexity of data used in the digital world large organizations need to handle a variety of unstructured data. Traditionally relational database systems have been used to store, process and retrieve data although other types of databases such as hierarchical, network, graph have existed before 1970, yet their commercial goals were not achieved until recent years. These databases (called NoSQL or Non-relational databases) are complements to relational databases and are used by world's largest organizations such as Google, Amazon and Facebook. This research compares the performance of relational and non-relational databases namely Oracle, and MongoDB by executing complex queries on a large set of data that is available in document-based mode and is converted to Oracle tables. The results show that data retrieval is significantly faster in MongoDB however some mathematical queries such as aggregation functions(sum, count, AVG) are better suited to Oracle RDBMS.

Keywords: NoSQL Databases, SQL Databases, RDBMS, Non-Relational, MongoDB, OracleDB, SQL Loader, Document Store, Column-Family Store, Graph Store, Key-Value Store.

1. INTRODUCTION

Nowadays, data plays a vital role in producing of every organization's activities. Furthermore, information is utilised as an essential part in the modern world to support complex business decisions based on the collected data. Currently, the rapid growth of data and having a massive amount of data that comes out every day from the web and business applications creates too much for RDBMSs to handle, this has added interest to alternatives to Relational database management systems. RDBMSs is a major technology that stores large portions of structured data which rely on providing ad-hoc querying facilities by using Structured Query Language (SQL). Oracle, MS SQL Server, MS Access, MySQL as well as Sybase are examples of relational database management systems which are utilized to store, manipulate and retrieve data [1]. However, unstructured data is another data model that is becoming easier to capture and access through the new generations of service providers such as Google, Amazon, eBay and Facebook which has Exabyte of data and processing these huge amounts of data enforces distributed storage [2]. Consequently, huge organizations decided to use an alternative for handling data at scale which led to the introduction of non-relational model. The term NoSQL (Not Only SQL) was first used by Carlo Strozzi in the year of 1998, which is a non-relational approach that is used to store and retrieve unstructured data quickly, unlike relational databases which apply mathematical relationships between tables [3]. Storing data in non-relational does not have clear and well-structured mechanism to join data from a diverse table structure to one another this is called free schema architecture. Most NoSQL databases are defined as distributed, horizontally scalable and open source [2]. Moreover, there are several Non-relational database management systems such as MongoDB, Cassandra, OrientDB, Aerospake and etc. Clearly, the performance is a significant part of both type of database (Relational and non-relational database) in order to store huge data rapidly and access the data quickly. This research is focused on Oracle DB as a relational DBMS and MongoDB as a Non-relational DBMS to find out the differences in their performances in order to show which one performs better in data retrieval by executing several queries.

2 NoSQL CATEGORIES

Non-relational database has many categories, the most common ones are Key-value store, Document-based stores, Graph stores and Column family stores.

2.1 Key-value (KV) Store

The simplicity of Key-value stores provides a much more powerful and efficient approach to other types of NoSQL database systems. Typically, data is stored as key-value pairs (key-array pairs) such that values are indexed for retrieval by keys. These data are stored as alpha-numeric in hash tables where the key is unique and the value type is either JSON, BLOB or string [6, 7]. Each standalone table has two columns, one column holds Primary Key (PK) and the other one holds a collection of logical values [8]. This technique is called Row Store (or Tuple Store) because all of the data for a single record are stored together [9]. Examples of key-value DBMSs are DynamoDB, Riak, Redis, Aerospike, FoundationDB, Berkeley DB, Voldemort, SimpleDB and Dynomite [10]. The Key-value data model provides lots of features such as highly scalable distributed data, support for massive storage and high concurrency, and the main vital feature is the speed at which data retrieval is accomplished compared with relational databases [6, 12].

2.2 Column-Oriented (Column Family or Wide-Column) Stores

Column Family stores also referred to as columnar database, column oriented stores, extensible record stores or wide column stores [7, 13]. Generally, Column Family stores have been created for storing and processing a huge quantity of data distributed over diverse servers. Column Family stores is a collection of rows containing several columns which store values contiguously and all columns can be arranged by column family [14]. Unlike the relational databases which store data in structured tables (columns and rows) with a fixed size of fields for each record, In this type of NoSQL, the data is not stored in structured tables; instead data is kept in an enormously distributed architecture processed across many machines [11, 13]. this allows adding new columns in a row without having to insert any values for the existing rows, which means Columns in Column Family databases are extendable and each key can be connected to one or more columns [7, 15]. The advantage of storing data in columns are fast search and high performance on aggregation queries like MAX, MIN, AVG, SUM and COUNT [14, 16]. Examples of Column Family stores include Hadoop/HBase API, Cassandra, Hypertable, Accumulo and Big-Table [10].

2.3 Document-Based Stores: Document oriented Database

Document-based stores are one of the foremost categories of NoSQL databases systems. Basically, this type of NoSQL is designed to manage, retrieve and store a collection of literal documents in a format like text document, PDF, XML, JSON (JavaScript Object Notation) or BSON (Binary JSON) [6, 11, 7]. The schema for Document model and Relational model are somehow similar, in that the collections represent the tables; the documents represent the rows as well as the Key/Value pairs represent Columns [6]. Each Document based database is a collection of documents which hold semi-structured data inside including many different key value pairs which through the PK its value can be accessed. In addition, owing to the schema, this NoSQL type is more flexible and easy to change; therefore both Keys and Values in a document are entirely searchable, and data are more logically grouped together[6, 7]. More importantly, adding any number of attributes to a document is allowed by users and the domain (data types) can differ from document to document [6, 17]. Furthermore, unlike Key-Value stores, the Document stores databases support more complex data and multiple indexes on documents per database [18]. Scalability and fault-tolerant are the two vital features of Document databases, however document stores are not suitable where a database has lots of relationships and normalization [12, 17]. The most popular document stores are Mongo DB, Couch DB, Terrastore, AmisaDB JasDB, EJDB and iBoxDB [10].

2.4 Graph Database

Graph databases also called Graph-oriented databases are special types of NoSQL databases which store data in the form of a graph [7, 17]. Basically, there are three core abstractions in the graph database model. The first abstraction is Nodes (called Vertexes) which are sort of, equivalent to the tables in the relational database. The second is Relationships between these Nodes, called Edges. And the third abstraction is Properties which are Key-value pair (called column) where the key is a string and value is either a primitive or an array of a primitive type. These Properties have been attached to both Nodes and relationships between Nodes [17, 14]. Therefore, it can be said that the structure of a Graph database is built from a collection of Nodes and Edges. This approach does not store a data inside rows and columns; it stores all data in a network of Nodes and Edges [19]. Additionally, a Graph database provides an important quality technique called index-free adjacency meaning each Node contains a direct pointer to its neighbouring Node, which means every Node does not need a dedicated index. By using this technique, Millions of records are traversed making connection between data smooth. Nonetheless, graph databases offer ACID properties, schema less architecture, rollback support and efficient storage of semi-structured data [7, 17]. Relational databases are not well suited to exploring the relationships among extremely linked data however this can be accomplished in graph databases through the usage of pointers. Application of graph databases include social network applications (Facebook, Twitter and etc.), security and access control, and bioinformatics, as an alternative of the relational model for the purpose of managing huge relationships set of data [17, 13, 19]. Examples of Graph databases systems are Neo4j, Infinite Graph, HyperGraphDB, GraphBase, Trinity, AllegroGraph, and BigData [10].

3. RELATIONAL AND NON-RELATIONAL DATABASE COMPARISON

The most important features and differences of relational and non-relational databases are summarized in the following table [17, 21, 22, and 23]:

Parameters	Relational databases	Non-relational databases
1. Database model	The database model is based on the Relational model approach known as Relational database (RDBMS)	The database model is based on the Model-less approach known as non-relational or NoSQL (NewSQL) database.
2. Data Representation	Stores structured-data in form of tables (columns, rows), relationships among tables or joins.	Stores unstructured-data based on several types of stores such as Key-value pairs, Column Family, Graph database, or Document store.
3. Schema	The data has to fit in predefined tables or structure.	The schema is highly flexible and dynamic.
4. Scaling	Vertically scalable (increasing the power of hardware like CPU, RAM, Hard disk, or etc.)	Horizontally scalable (increasing capacity by adding more machines or database servers)
5. Query Language	SQL (Structured Query Language) is a very powerful database language, which is used to define and manipulate data.	NoSQL does not have a standard query language. UnQL (Unstructured Query Language) is sometimes used in NoSQL databases. The syntax of using UnQL varies from database to database. Most of the NOSQL database providers have created their own query language, for example Cassandra supports CQL (Cassandra query language), MongoDB uses mongo query language etc.
6. Transactional operation	SQL databases are the best for high transactional (delete, update and insert) on data.	NoSQL databases are the best for selecting data.
7. Transactional properties	ACID (Atomicity, Consistency, Isolation and Durability) properties are emphasized by SQL databases.	CAP theory (Consistency, Availability and Partition tolerance) are emphasized by NoSQL databases.
8. Consistency	In relational database Consistency means that all users see the same version of data after the transaction. Thus, relational database provides better consistency than no-relational database.	In non-relational database "Eventual Consistency" means that there is no guarantee for reads and writes after the transaction for all entities in the database will be immediately consistent
9. Normalization/ de-normalization	Relational database uses normalization which simply means dividing single table to smaller tables to minimize data redundancy and improve performance.	In Non-relational database a single table is used to store all records which are called de-normalization. Furthermore, the operations of INSERT, DELETE as well as UPDATE are not easy, but SELECT is very easy.
10. Data Integrity	Comparing to the relational database removes all duplication records to stop inconsistent data from loading the database.	There is a lack of data replication in the non - relational database. It is necessary in a flat database to update each of a person's address manually for certain that all his data are in a good state of consistency.
11. Data Retrieval	SQL is used by relational database which is utilized primary key among tables to collect the behest records.	In non-relational database records can be found by using multiple criteria which is inefficient. The database needs several passes to inspect all records for matches.
12. Data manipulation	Relational database uses (DML: Data Manipulation language) to enter and manipulate data when the structure of the database is finished.	RESTful interface such as (HTTP PUT, POST and DELETE) is utilized by Non-relational database also with different formats including (JSON, THrift and RDF) as well as the Data manipulation APIs are offered by many of them such as Google data store.
13. Features	A key feature of RDBMS is maintenance facility that makes the system to repair, test and back up easily by offering tools for database administrator.	A key feature of NoSQL systems is “shared nothing”, horizontal scaling– replicating and partitioning data over many servers.

4. METHODOLOGY

Getting the amount of data needed for comparison purposes of NoSQL database systems is a challenge since reliable results require records of around millions and such high values are only possessed by huge organization such as Google, CERN, Facebook and etc.., this might explain the poverty of researches in this area that uses tests to confirm the performance of these systems. For this purpose extensive searches were conducted and data from infochimps.com were used. The amount of records is around 500 thousand from a number of tables that is available in .CSV files which is supported by MongoDB, in order to convert that to Oracle tables we used SQL loader, a tool provided by Oracle, to accomplish the required data in Oracle format.

Moreover the source has all the data in a single CSV file and that does not produce accurate results if converted to a single Oracle table thus several tables were created the following E-R diagram are the highlights of the relationships.

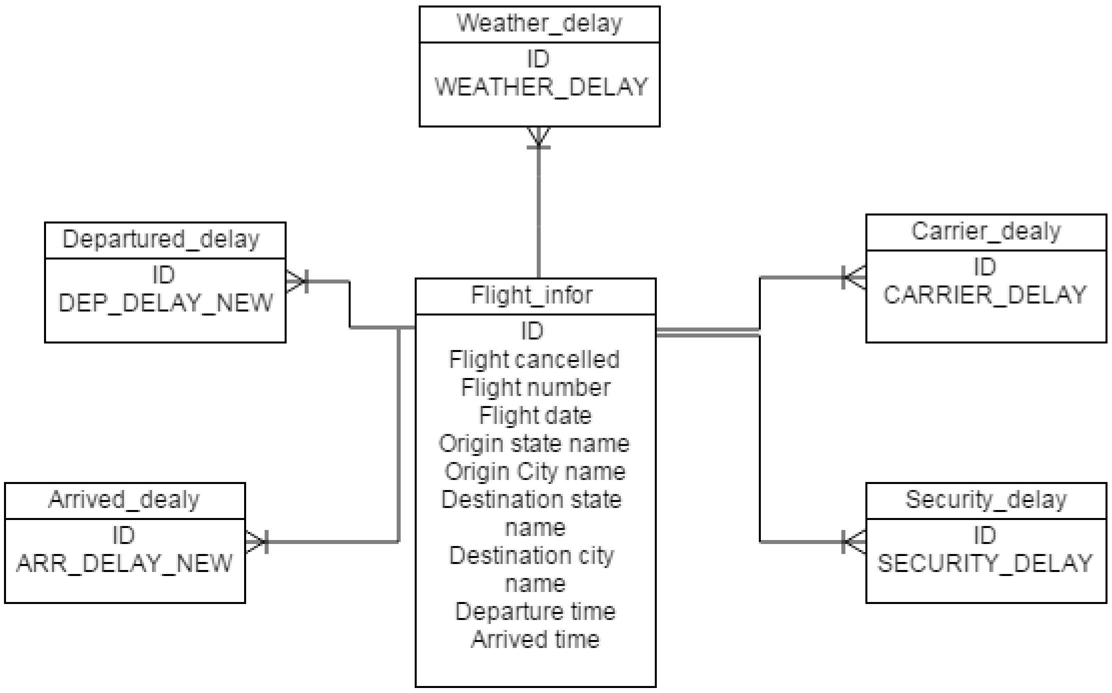


Figure (1): E-R Diagram

It should be noted that the platform on which the tests are conducted along with the features of the machine play an important role in the outcomes of the queries; the following are the properties of the executing computer.

- Intel® Core™2 Duo CPU T5750 @ 2.00GHz × 2
- 4 GB RAM
- 40 GB partition
- Ubuntu 12.04 64bit
- Oracle Database 11g Release 2
- Mongo DB shell version: 2.0.9-rc0-pre-
- Ubuntu 12.10 64-bit

5. RELATED WORK

Although many comparisons has been conducted to determine the differences in performance of relational databases yet, we have not seen many investigations about the performance comparison between relational and non-relational databases in terms of reading and writing data by using several queries on a large set of data. A study concluded that a single approach cannot be providing better results in various circumstances [24]. However, the paper does not show any performance testing between any kind of relational and non-relational databases to prove the point. Another study [25] replaces an existing oracle database with MongoDB and Redis which are two non-relational database systems and shows that as long as the number of records increase the non-relational databases provide higher speed especially MongoDB which clearly surpasses oracle in response time, however the authors have only applied insertion statements and have overlooked data retrieval which is the focus of our study. There are several other studies that attempt to compare relational and non-relational databases in terms of architecture and taxonomy but neither presents actual tests [17] [21] [23].

6. PERFORMANCE TEST

For each test first the question is introduced then its query in MongoDB and then the query in Oracle format.

6.1 Select

In section one, the outcome of comparing Mongo DB to Oracle DB demonstrates that Mongo DB has a better performance for data selection. Figure (1.1) shows that the execution time for Mongo DB query 1 is (0.633) second; however, Oracle DB retrieved the same amount of data in (3.45) seconds. Furthermore, for all queries, as shown in section one, Mongo DB performed more effectively compared to Oracle DB.

Q1/ Find all flight number and origin state name where flight departure delay is more than 10 minutes.

```
db.airports.find({"DEP_DELAY_NEW":{"$gt":10}},{"FL_NUM":1,"ORIGIN_STATE_NM":1}).e
xplain();
select f.FL_NUM, f.ORIGIN_STATE_NM from flight_info f join departured_delay d on (f.id =
d.id) where d.DEP_DELAY_NEW >10#
```

Q2/ Find all origin state name and Destination state name and flight date where flight is cancelled

```
 db.airports.find({"CANCELLED":1},{"ORIGIN_STATE_NM":1,"DEST_STATE_NM":1,"FL_D
ATE":1}).explain();
select FL_NUM, ORIGIN_STATE_NM, DEST_STATE_NM from flight_info where
CANCELLED=1
```

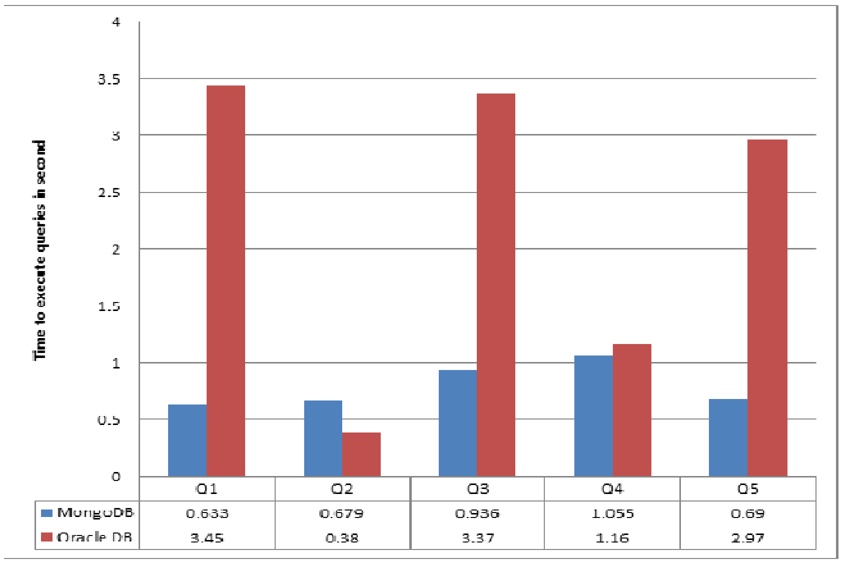
Q3/ Find all Flight Number, Flight Date and Departure Time where Origin State Name is “California” or Security flight delayed is more than 15 minutes.

```
db.airports.find({"$or":[{"SECURITY_DELAY":{"$gt":15}},{"ORIGIN_STATE_NM":"California
"}]},{"FL_NUM":1,"FL_DATE":1,"DEP_TIME":1}).explain();
select f.FL_NUM, f.FL_DATE, f.DEP_TIME from flight_info f join security_delay s
on (f.id = s.id) where f.ORIGIN_STATE_NM = 'California' or s.SECURITY_DELAY >15
```

Q4/ Find Origin City name , Destination City Name, Flight Number and Flight Date where filght is delayed because of carrier or weather delay more than 30 minutes.

```
 db.airports.find({"$or":[{"CARRIER_DELAY":{"$gt":30}},{"WEATHER_DELAY":{"$gt":30}}]
},{"ORIGIN_CITY_NM":1,"DEST_CITY_NAME":1,"FL_NUM":1,"FL_DATE":1}).explain();
select f.FL_NUM, f.FL_DATE, f.ORIGIN_CITY_NAME, f.DEST_CITY_NAME from flight_info f
join carrier_delay c on f.id = c.id join weather_delay w on f.id = w.id where c.CARRIER_DELAY >
30 or w.WEATHER_DELAY >30
```

Q5/ Find all flight number ,Flight Date, Origin State name and Destination State name where arrived delayed is between 10 to 30 minutes
db.airports.find({"ARR_DELAY_NEW":{"$in":[10,30]}},{"ORIGIN_STATE_NM":1,"DEST_STATE_NM":1,"FL_NUM":1,"FL_DATE":1}).explain();
select f.FL_NUM, f.FL_DATE, f.ORIGIN_CITY_NAME, f.DEST_CITY_NAME from flight_info f join arrived_delay a on f.id = a.id where a.ARR_DELAY_NEW >=10 and a.ARR_DELAY_NEW <= 30



	Q1	Q2	Q3	Q4	Q5
🟦 MongoDB	0.633	0.679	0.936	1.055	0.69
🟥 Oracle DB	3.45	0.38	3.37	1.16	2.97

Figure (2): Select

6.2 Aggregation

In section two, the aggregation functions were employed to find how each database performed. The aggregation functions that have been used are (Sum, Count, Avg). For each function, five queries were executed for the purpose of comparing Oracle to Mongo. Oracle DB performed better in all tests for (Sum, Count, Avg) functions. These results have been confirmed in a separate test conducted by a blogger[26] which performs an aggregation query on 1m records MongoDB database and then the same data is transferred to Oracle and the same aggregation query is run, results confirm that for aggregation functions Oracle has a better response time.

6.2.1 Count

Q1/ Find the total number of Departure delay more than 15 minutes
db.airports.count({"DEP_DELAY_NEW":{"$gt":15}});
select count(DEP_DELAY_NEW) from departured_delay where DEP_DELAY_NEW > 15

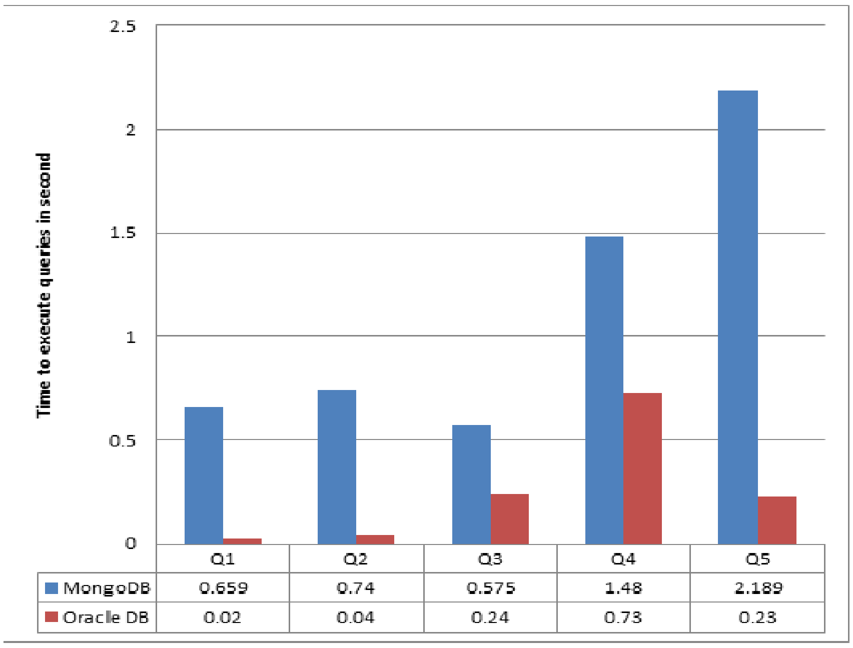
Q2/ Find the total number of Cancelled flight for each Origin State Name.
db.airports.aggregate([{$match:{CANCELLED:1}},{$group:{_id:"$ORIGIN_STATE_NM",count:{$sum:1}}}]);
select ORIGIN_STATE_NM ,count(CANCELLED) from flight_info where CANCELLED = 1 group by ORIGIN_STATE_NM

Q3/ Find the total number of weather delay where Destinationa State name is (Texas) for each Flight number.
db.airports.aggregate([{$match:{DEST_STATE_NM:"Texas","WEATHER_DELAY":{"$gte":1}}},{$group:{_id:"$FL_NUM",count:{$sum:1}}}]);

select f.FL_NUM ,count(w.WEATHER_DELAY) from flight_info f join weather_delay w on (f.id = w.id) where DEST_STATE_NM = 'Texas' group by f.FL_NUM

Q4/ Find the total number of Carrier delay for each Flight number and Flight date where the total number is greater than 5.
db.airports.aggregate([{$match:{CARRIER_DELAY:{"$gt":0}}},{$group:{_id:{FL_DATE:"$FL_DATE",FL_NUM:"$FL_NUM"},count:{$sum:1}}},{$match:{count:{$gt:5}}}]);
select f.FL_NUM, f.FL_DATE ,count(c.CARRIER_DELAY) from flight_info f join carrier_delay c on (f.id = c.id) having count(c.CARRIER_DELAY) > 5 group by f.FL_NUM, f.FL_DATE

Q5/ Find the total number of Arrived delay more than 10 minutes for each Flight date, Flight number and Destination City name when the total number is more than 100.
db.airports.aggregate([{$match:{ARR_DELAY_NEW:{"$gt":10}}},{$group:{_id:{FL_DATE:"$FL_DATE",DEST_CITY_NAME:"$DEST_CITY_NAME"},count:{$sum:1}}},{$match:{count:{$gt:100}}}]);
select f.FL_DATE, DEST_CITY_NAME ,count(a.ARR_DELAY_NEW) from
flight_info f join arrived_delay a on (f.id = a.id) where a.ARR_DELAY_NEW > 10 having
count(a.ARR_DELAY_NEW) > 100 group by f.FL_DATE, f.DEST_CITY_NAME



	Q1	Q2	Q3	Q4	Q5
🟦 MongoDB	0.659	0.74	0.575	1.48	2.189
🟥 Oracle DB	0.02	0.04	0.24	0.73	0.23

Figure (3): Count

6.2.2 SUM
Q1/ Find the total minutes for security delay.
db.airports.aggregate([{$match:{SECURITY_DELAY:{“$gte”:1}}},{$group:{_id:null, total:{$sum:”$SECURITY_DELAY”}}}]);
select sum(SECURITY_DELAY) from security_delay where SECURITY_DELAY >=1

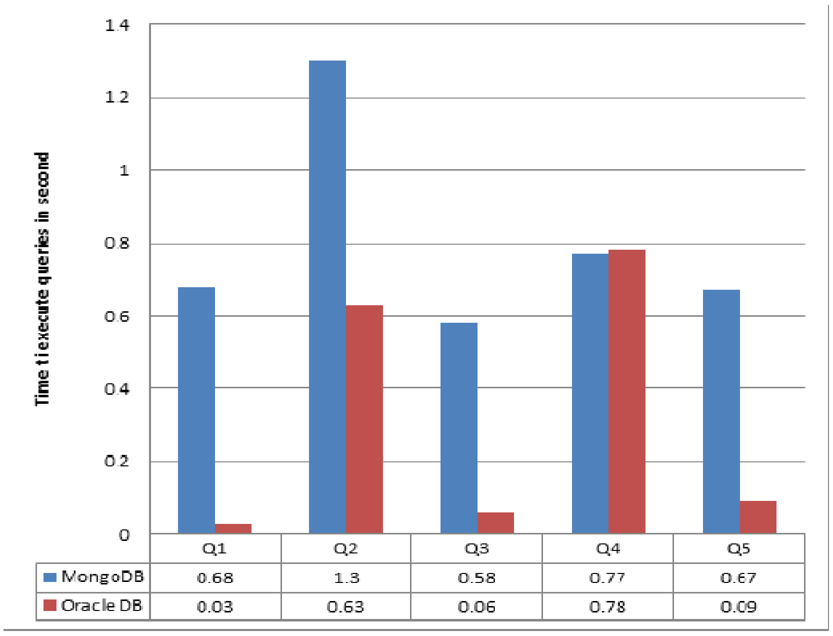
Q2/ Find the total minutes of carrier delay for each flight date and Origin state name
db.airports.aggregate([{$match:{CARRIER_DELAY:{“$gte”:1}}},{$group:{_id:{FL_DATE:”$FL_DATE”,ORIGIN_STATE_NM:”$ORIGIN_STATE_NM”},total:{$sum:”$CARRIER_DELAY”}}}]);

select f.FL_DATE, ORIGIN_STATE_NM ,sum(c.CARRIER_DELAY) from flight_info f join carrier_delay c on (f.id = c.id) group by f.FL_DATE , f.ORIGIN_STATE_NM

Q3/ Find the total minutes for departure delay more than 12 hours and Destination state name is California for each flight date.
db.airports.aggregate([{$match:{DEP_DELAY_NEW:{“$gt”:720},DEST_STATE_NM:”California ”}},{$group:{_id:{FL_DATE:”$FL_DATE”}, total:{$sum:”$DEP_DELAY_NEW”}}}]);
select f.FL_DATE, sum(d.DEP_DELAY_NEW) from flight_info f join eparture_delay d on (f.id = d.id) where d.DEP_DELAY_NEW > 720 and f.DEST_STATE_NM = ‘California’ group by f.FL_DATE

Q4/ Find the total minutes for weather delay for each flight number and flight date when the total minutes is greater than one hour.
db.airports.aggregate([{$match:{WEATHER_DELAY:{“$gte”:1}}},{$group:{_id:{FL_DATE:”$FL_DATE”,FL_NUM:”$FL_NUM”},total:{$sum:”$WEATHER_DELAY”}}},{$match:{total:{$gt:60}}}]);
select f.FL_NUM, f.FL_DATE, sum(w.WEATHER_DELAY) from flight_info f join weather_delay w on (f.id = w.id) having sum(w.WEATHER_DELAY) > 60 group by f.FL_NUM, f.FL_DATE

Q5/ Find the total minutes for Departure delay where the Origin state name is (New York) and the Destination is (California) and the Total minutes is more than two hours for each flight date.
db.airports.aggregate([{$match:{DEP_DELAY_NEW:{“$gte”:1},ORIGIN_STATE_NM:”New York”,DEST_STATE_NM:”California”}},{$group:{_id:{FL_DATE:”$FL_DATE”}, total:{$sum:”$DEP_DELAY_NEW”}}},{$match:{total:{$gt:120}}}]);
select f.FL_DATE, sum(d.DEP_DELAY_NEW) from flight_info f join eparture_delay d on (f.id = d.id) where ORIGIN_STATE_NM = ‘New York’ and DEST_STATE_NM = ‘California’ having sum(d.DEP_DELAY_NEW) > 120 group by f.FL_DATE



	Q1	Q2	Q3	Q4	Q5
🟦 MongoDB	0.68	1.3	0.58	0.77	0.67
🟥 Oracle DB	0.03	0.63	0.06	0.78	0.09

Figure (4): Sum

6.2.3 Average

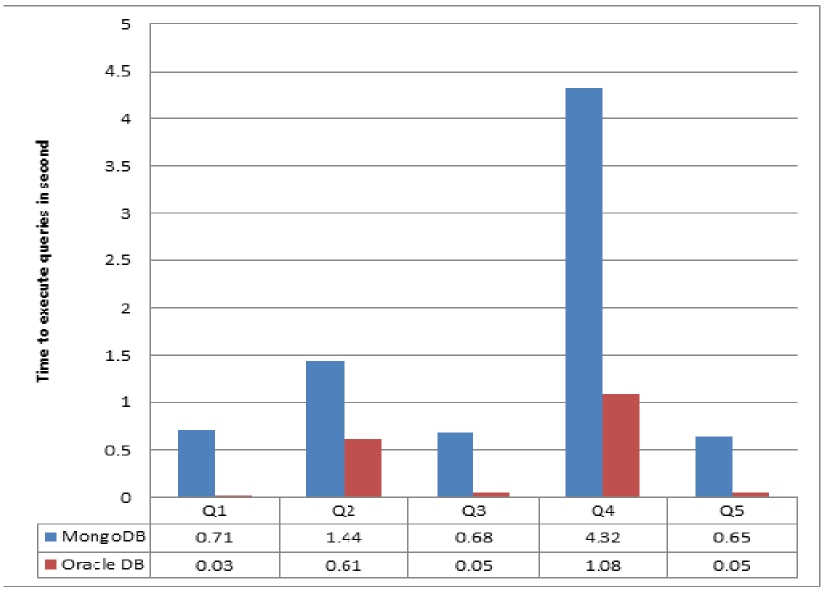
Q1/ Find the average minutes for weather delay.
db.airports.aggregate([{$match:{WEATHER_DELAY:{“$gte”:1}}},{$group:{_id:null, avg:{$avg:”$WEATHER_DELAY”}}}]);
select avg(WEATHER_DELAY) from weather_delay Where WEATHER_DELAY >=1

Q2/Find the average minutes of carrier delay for each flight number and Origin city name
db.airports.aggregate([{$match:{CARRIER_DELAY:{“$gte”:1}}},{$group:{_id:{FL_NUM:”$FL_NUM”,ORIGIN_CITY_NAME:”$ORIGIN_CITY_NAME”},avg:{$avg:”$CARRIER_DELAY”}}}]);
select f.FL_NUM, f.ORIGIN_CITY_NAME, avg(c.CARRIER_DELAY) from flight_info f join carrier_delay c on (f.id = c.id) where c.CARRIER_DELAY >=1 group by f.FL_NUM, f.ORIGIN_CITY_NAME

Q3/ Find the average minutes for security delay is more than one hour and the Destination state name is (New York) for each day.
db.airports.aggregate([{$match:{SECURITY_DELAY:{“$gt”:60},DEST_STATE_NM:”New York”}},{$group:{_id:{FL_DATE:”$FL_DATE”}, avg:{$avg:”$SECURITY_DELAY”}}}]);
select f.FL_DATE, avg(s.SECURITY_DELAY) from flight_info f join security_delay s on (f.id = s.id) where s.SECURITY_DELAY > 60 and DEST_STATE_NM = ‘New York’ group by f.FL_DATE

Q4/ Find the total minutes for Departure delay for each flight number and day of flight when the departure minutes is more than 60minutes.
db.airports.aggregate([{$match:{DEP_DELAY_NEW:{“$gte”:1}}},{$group:{_id:{FL_DATE:”$FL_DATE”,FL_NUM:”$FL_NUM”},avg:{$avg:”$DEP_DELAY_NEW”}}},{$match:{avg:{$gt:60}}}]);
select f.FL_NUM, f.FL_DATE, avg(d.DEP_DELAY_NEW) from flight_info f join eparture_delay d on (f.id = d.id) having avg(d.DEP_DELAY_NEW) > 60 group by f.FL_NUM, f.FL_DATE

Q5/ Find the average minutes for Carrier delay more than 30 minutes for each flight date, where the Origin state name is (California) and the Destination is (New York) and the average minutes is more than 60 minutes.
db.airports.aggregate([{$match:{CARRIER_DELAY:{“$gt”:30},ORIGIN_STATE_NM:”California”,DEST_STATE_NM:”New York”}},{$group:{_id:{FL_DATE:”$FL_DATE”},avg:{$avg:”$CARRIER_DELAY”}}},{$match:{avg:{$gt:60}}}]);
select f.FL_DATE, avg(c.CARRIER_DELAY) from flight_info f join carrier_delay c
on (f.id = c.id) where c.CARRIER_DELAY > 30 and (f.ORIGIN_STATE_NM = ‘California’ and f.DEST_STATE_NM = ‘New York’) having avg(c.CARRIER_DELAY) > 60 group by f.FL_DATE



	Q1	Q2	Q3	Q4	Q5
🟦 MongoDB	0.71	1.44	0.68	4.32	0.65
🟥 Oracle DB	0.03	0.61	0.05	1.08	0.05

Figure (5): Average

7. CONCLUSION

The purpose of this paper is to demonstrate the differences of response time in relational and non-relational databases by executing a number of queries on the target systems, namely Oracle and MongoDB. Our results show that for such a high number of records MongoDB does a better job while retrieving data however it fails to exceed Oracle's speed when it comes to aggregation functions. This proves that NoSQL databases are no replacement for relational databases, the two types can coexist and it is only the needs of the customer that specifies which one suits better. This research can be further enhanced by involving several NoSQL database systems and using a higher number of records.

REFERENCES

[1] Begg, C. and Connolly, T. (2010) Database System: A practical Approach to design, Implementation, and Management.5th ed., New York, Addison-Wesley.

[2] Satapathy, S. Patra, M. and Paddy, R., RDMS to No-SQL: Reviewing Some Next-Generation Non-Relational Database’s (IJAEST) International Journal of Advanced Engineering Sciences and Technology 2011 Volume 11 No 1. P 15-30.

[3] Bolton, D. Definition of NoSQL. [Online] Last Accessed 25 September 2014 at: “http://cplus.about.com/od/n/g/Definition-Of-Nosql.htm”

[4] PERDUE, T. NoSQL: An Overview of NoSQL Databases. [Online] Last Accessed 25 August 2014 at: “http://newtech.about.com/od/databasemanagement/a/Nosql.htm”

[5] Fowler, M. (2012) NoSQL Definition [Online] Last Accessed 27 August 2014 at: “http://martinfowler.com/bliki/NosqlDefinition.html”

[6] A B M Moniruzzaman and Syed Akhter Hossain, NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison, International Journal of Database Theory and Application, Vol. 6, No. 4. 2013.

[7] Opeyemi Michael Ajayi, a Perspective of NoSQL: User Experience and Scalability of Cassandra and MongoDB.

[8] Types of NoSQL Databases, [Online] Last Accessed 29 August 2014 at: “http://nosql.rishabhagrawal.com/2012/07/types-of-nosql-databases.html”

[9] What is a NoSQL Key-Value Store? [Online] Last Accessed 14 August 2014 at: "http://www.aerospike.com/what-is-a-nosql-key-value-store/"
[10] List Of NoSQL Databases, [Online] Last Accessed 14 August 2014 at: "http://nosql-database.org/"
[11] Leavitt, N.; Will NoSQL Databases Live Up to Their Promise?, vol.43, no.2, pp.12-14, Feb. 2010doi: 10.1109/MC.2010.58
[12] Clarence J M Tauro, Aravindh S and Shreeharsha A.b. Article: Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases. International Journal of Computer Applications 48(20):1-4, June 2012.
[13] Manoj V, Comparative Study of NoSQL Document, Column Store Databases and Evaluation of Cassandra. International Journal of Database Management Systems (IJDMS) Vol.6, No.4, August 2014.
[14] NoSQL, [Online] Last Accessed 18 August 2014 at: "http://www.w3resource.com/mongodb/nosql.php"
[15] Apache Cassandra 1.0 Documentation, [Online] Last Accessed 8 August 2014 at: "http://www.datastax.com/docs/1.0/ddl/column_family"
[16] Girish Kumar. [Online] Last Accessed 8 August 2014 at: "http://www.datastax.com/docs/1.0/ddl/column_family"
[17] Ameya Nayak, Anil Poriya and Dikshay Poojary. Type of NOSQL Databases and its Comparison with Relational Databases. International Journal of Applied Information Systems (IJAIS) Foundation of Computer Science FCS, New York, USA Volume 5– No.4, March 2013.
[18] Rick Cattell. 2011. Scalable SQL and NoSQL data stores. *SIGMOD Rec*. 39, 4 (May 2011), 12-27.
[19] Charu Tyagi. Comparative Analysis of Relational Databases and Graph Databases 2012.
[20] Azharuddin Khan, (2011) Difference between SQL and NoSQL: Comparison. [Online] Last accessed 25 July 2013 at: "http://www.thewindowsclub.com/difference-sql-nosql-comparision"
[21] Meetali Bageshwari, Pradnesh Adurkar, Ankit Chandrakar. Clinical Database: RDBMS V/S Newer Technologies (NoSQL And Xml Database); Why Look Beyond RDBMS and Consider the Newer. International Journal of Computer Engineering & Technology (IJCET), Volume 5, Issue 3, March (2014), pp. 73-83
[22] Luke P, SQL vs NoSQL Database Differences Explained with few Example DB. [Online] Last accessed 20 September 2013 at: "http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/".
[23] Nishtha Jatana, Sahil Puri, Mehak Ahuja, Ishita Kathuria, Dishant Gosain, "A Survey and Comparison of Relational and Non-Relational Database," International Journal of Engineering Research & Technology (IJERT), vol. I, no. 6, 2012.
[24] Florian Eckerstorfer. Performance of NoSQL Databases, November 19, 2011.
[25] Yogesh Punia, Rinkle Aggarwal. Implementing Information System Using MongoDB and
[26] Redis. International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3, No.2, Pages: 16 - 20 (2014).
[27] [Online] Last Accessed 14 November 2014 at: http://blog.jooq.org/2013/12/19/mongodb-lightning-fast-aggregation-challenged-with-oracle/.
[28] Sanjeev Kumar Jha, Pankaj Kumar and Dr. A.K.D.Dwivedi. An Experimental Analysis of MYSQL Database Server Reliability. International Journal of Computer Engineering & Technology (IJCET), Volume 3, Issue 2, March (2012), pp. 354 - 371.