SWINBURNE UNIVERSITY OF TECHNOLOGY

High Distinction Project for Fundamental of Data Management

Topic: Comparison between MySQL and MongoDB.

Name: Cuong Nguyen Student ID: 102840305 Date: 30/10/2020

MySQL and MongoDB Research Report

Cuong Nguyen | Student ID: 102840305

Abstract

Database systems has been an important component of computer systems since time immemorial, because they are crucial for storing and retrieving informations though queries. In this report, MySQL and MongoDB – two of these database management systems – would be assessed and compared, as well as key advantages and limitations. Additionally, the text explores differences between these database products by focusing on how each of them could resolve problems from users, following by practical implementations.

Table of Contents

I. Introduction

In an increasingly connected world today, the demand of capturing, accessing and storing data from different sources is becoming indispensable in modern companies and organisations. This leads to an increasingly higher need for enterprises and institusions to develop their database (or *data store*), and with that, to use modern database products.

While traditional database management systems (DBMS) like MySQL are still prevalent among developers today, non-relational database systems (NoSQL) are becoming more commonplace. According to the survey from StackOverflow about database trends in February 2020, MongoDB is the most popular among NoSQL database products with over a fourth of developers (26.7%) using it, while MySQL is still the most popular DBMS by professional developers, with over a half of professional programmers (53.5%) using it.

II. Methodology

This report aims to explore differences between the difference between relational databases products and NoSQL. The relational DBMS analysed in this report is MySQL, and would be compared with MongoDB. Both database systems would be compared and contrasted by implementing theoretical research about their similarities and differences, following by conducting an experiment on performance in SalesOrdersExample database, with implementations of,queries and joining commands on different datastores (such as SalesOrdersExample and DumasHotel from Distinction report). The aim of the report is to understand the similarities, as well as differences, between two products, so that the rising popularity of MongoDB nowadays among programmers, as stated in Introduction part, could be explained.

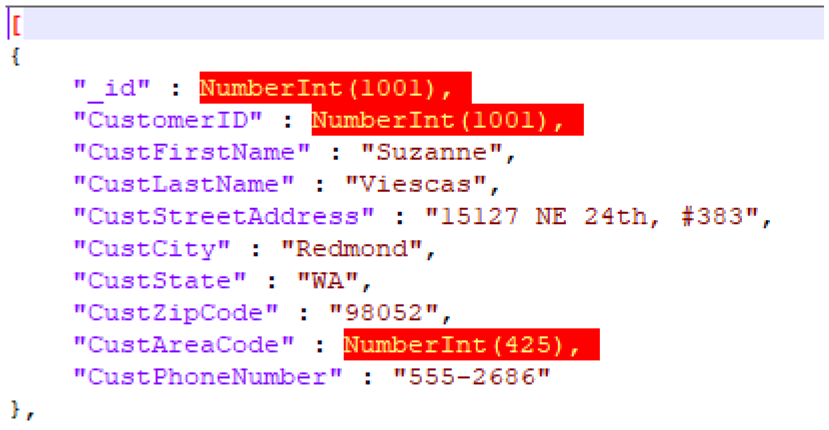
III. Results

This section would explore the aspects of MongoDB and how it could be contrasted with MySQL.

A. A document-based data store

As the name suggests, document-based database means that the system store data in the model of key-value, with value being stored as a document. Each document is self-contained, which means that no schema is required and create a substantial degree of flexibility over the data (Gall, 2019).

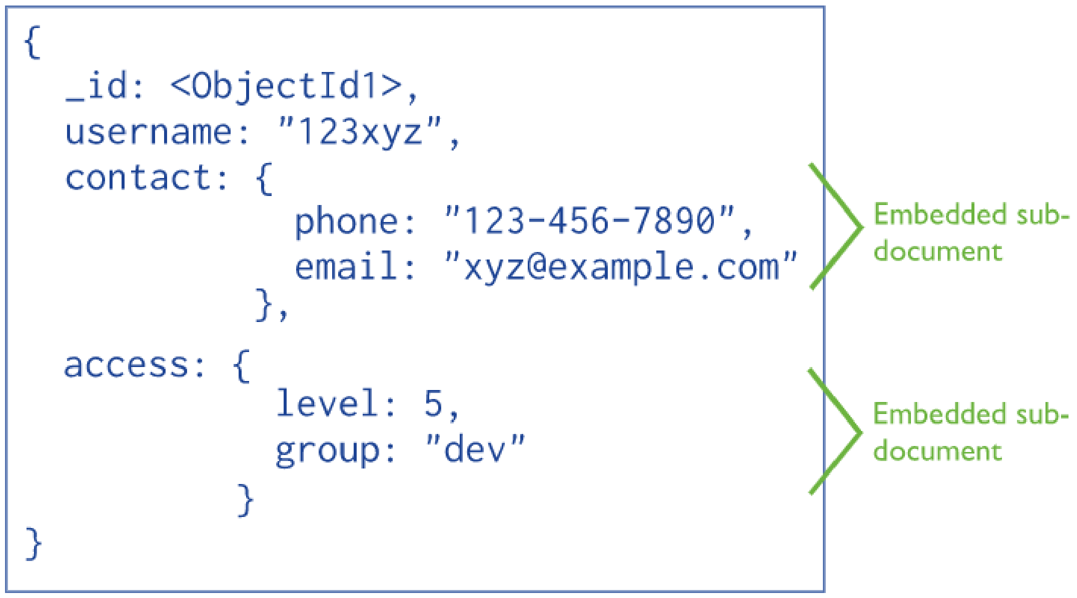
Documents in such a system, like MongoDB, could be saved in different formats such as JSON, XML, or in any other format that could state the information of an object by using structure of the document. An example of this data structure is given in the figure 1, showing the data for a customer in SalesOrdersExample database.



```
[
{
    "_id" : NumberInt(1001),
    "CustomerID" : NumberInt(1001),
    "CustFirstName" : "Suzanne",
    "CustLastName" : "Viescas",
    "CustStreetAddress" : "15127 NE 24th, #383",
    "CustCity" : "Redmond",
    "CustState" : "WA",
    "CustZipCode" : "98052",
    "CustAreaCode" : NumberInt(425),
    "CustPhoneNumber" : "555-2686"
},
```

Figure 1: A snippet of the JSON file used for saving data of a customer retrieved from SalesOrdersExample database.

When it comes to storing data, one key feature of MongoDB is that it encourages using denormalized documents, so that it could “take advantages of MongoDB rich documents”(MongoDB Manual, 2020). This could be seen in the exemple of the model in figure 2:

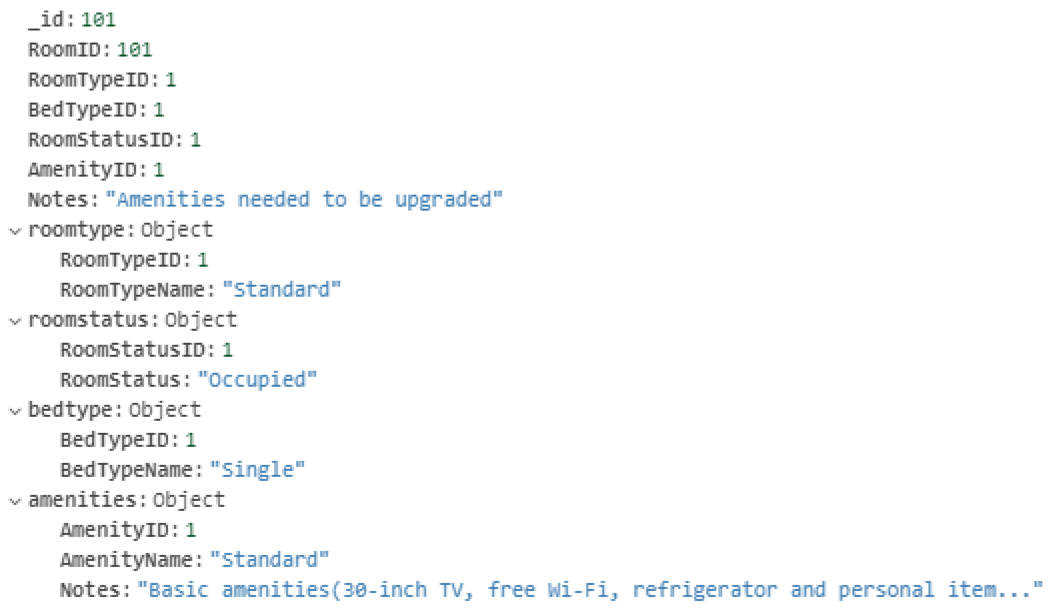


```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Figure 2: Exemple of data model in MongoDB(MongoDB Manual, 2020).

By contrast, using MySQL demands that users have to normalize the data (to third normal form) to prevent insertion, update and deletion anormalies in unnormalized tables (Prateek T L, 2020).

While MongoDB has no strict schema, it could still model relationships between entities like one-to-one and one-to-many, which is similar to MySQL. Figure 3 shows an entity (Room) could connect with other documents like RoomType, BedType, RoomStatus and Amenities by using common IDs for each document (similar to foreign key in MySQL).



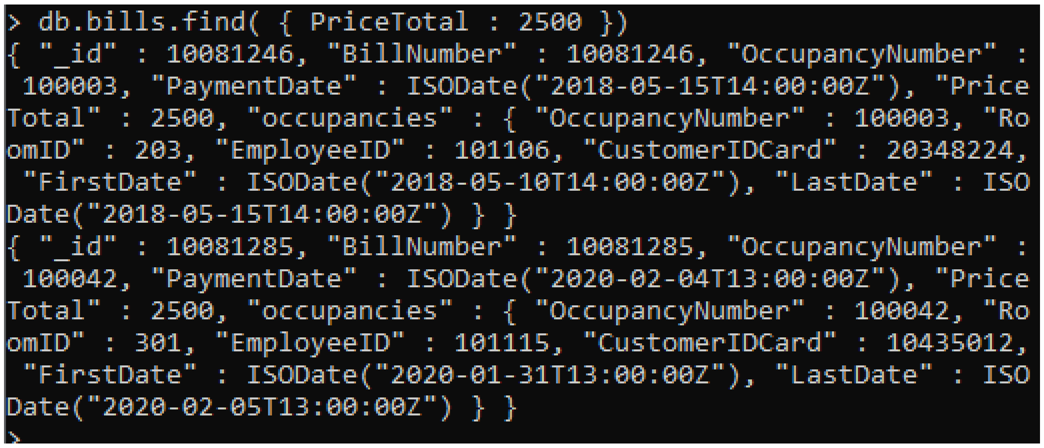
```
_id: 101
RoomID: 101
RoomTypeID: 1
BedTypeID: 1
RoomStatusID: 1
AmenityID: 1
Notes: "Amenities needed to be upgraded"
⌄ roomtype: Object
    RoomTypeID: 1
    RoomTypeName: "Standard"
⌄ roomstatus: Object
    RoomStatusID: 1
    RoomStatus: "Occupied"
⌄ bedtype: Object
    BedTypeID: 1
    BedTypeName: "Single"
⌄ amenities: Object
    AmenityID: 1
    AmenityName: "Standard"
    Notes: "Basic amenities(30-inch TV, free Wi-Fi, refrigerator and personal item..."
```

Figure 3. An extract from DumasHotel database (Distinction project), shows how MongoDB handles the relationship between documents.

B. <u>Comparing type of queries.</u>

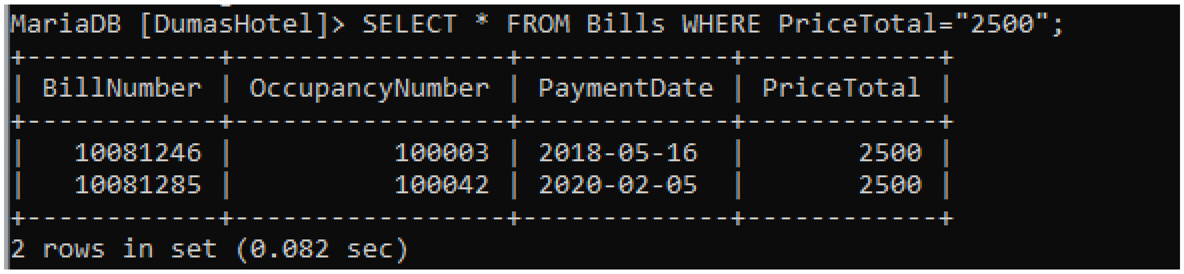
MongoDB uses an idea of CRUD, which stands for Create, Read, Update and Delete operations. It helps developers and administrators to perform tasks relating to change within database, similar to INSERT,SELECT, UPDATE and DELETE respectively in MySQL.

A notable difference between two database systems is the syntax. While in MongoDB, users need to use object query notation (which is identical to JSON), this action is unnecessary when using MySQL for querying. This could be illustrated in an example in figures 4 and 5:



```
> db.bills.find( { PriceTotal : 2500 })
{ "_id" : 10081246, "BillNumber" : 10081246, "OccupancyNumber" :
 100003, "PaymentDate" : ISODate("2018-05-15T14:00:00Z"), "Price
Total" : 2500, "occupancies" : { "OccupancyNumber" : 100003, "Ro
omID" : 203, "EmployeeID" : 101106, "CustomerIDCard" : 20348224,
 "FirstDate" : ISODate("2018-05-10T14:00:00Z"), "LastDate" : ISO
Date("2018-05-15T14:00:00Z") } }
{ "_id" : 10081285, "BillNumber" : 10081285, "OccupancyNumber" :
 100042, "PaymentDate" : ISODate("2020-02-04T13:00:00Z"), "Price
Total" : 2500, "occupancies" : { "OccupancyNumber" : 100042, "Ro
omID" : 301, "EmployeeID" : 101115, "CustomerIDCard" : 10435012,
 "FirstDate" : ISODate("2020-01-31T13:00:00Z"), "LastDate" : ISO
Date("2020-02-05T13:00:00Z") } }
>
```

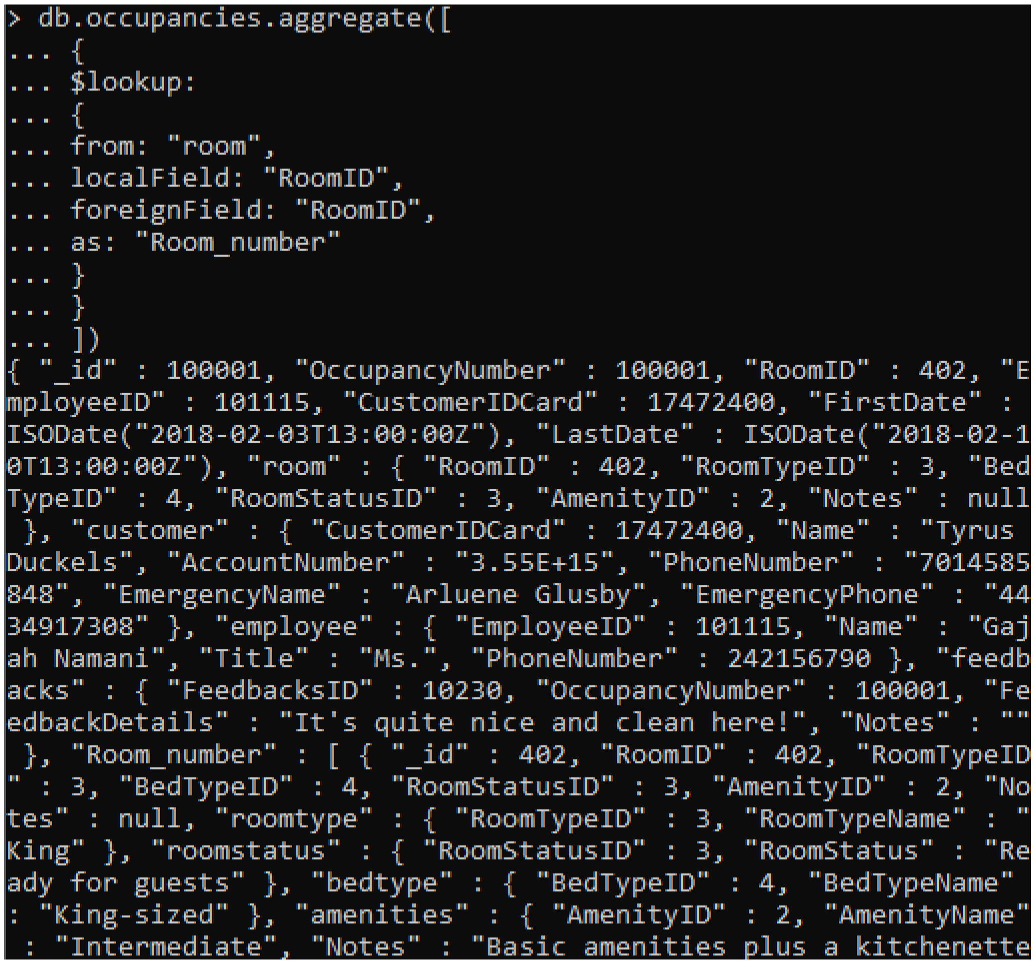
Figure 4: Command for finding data in Bills collection in MongoDB.



```
MariaDB [DumasHotel]> SELECT * FROM Bills WHERE PriceTotal="2500";
+------------+----------------+------------+------------+
| BillNumber | OccupancyNumber | PaymentDate | PriceTotal |
+------------+----------------+------------+------------+
|   10081246 |          100003 | 2018-05-16  |       2500 |
|   10081285 |          100042 | 2020-02-05  |       2500 |
+------------+----------------+------------+------------+
2 rows in set (0.082 sec)
```

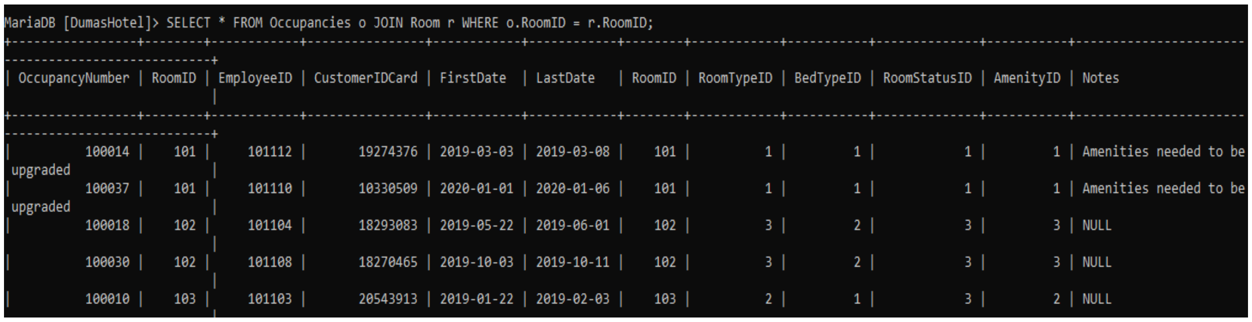
Figure 5: The same command as in Figure 4, but in MySQL. It could be seen that there is no need to use a different style of notation in the query.

There is also a similarity between these two database systems among queries relating to multiple tables. In MongoDB, developers could use *$lookup* command to perform aggregation, which is a new feature of the system from version 3.2 and could be used to performing left outer join between two tables with equality match (similar to using JOIN directly, or LEFT OUTER JOIN in MySQL). That could be seen in the following example using DumasHotel database from Distinction report. But it could not perform with more than two tables, however: which is a major drawback of MongoDB comparing to MySQL.



```
> db.occupancies.aggregate([
... {
... $lookup:
... {
... from: "room",
... localField: "RoomID",
... foreignField: "RoomID",
... as: "Room_number"
... }
... }
... ])
{ "_id" : 100001, "OccupancyNumber" : 100001, "RoomID" : 402, "E
mployeeID" : 101115, "CustomerIDCard" : 17472400, "FirstDate" :
ISODate("2018-02-03T13:00:00Z"), "LastDate" : ISODate("2018-02-1
0T13:00:00Z"), "room" : { "RoomID" : 402, "RoomTypeID" : 3, "Bed
TypeID" : 4, "RoomStatusID" : 3, "AmenityID" : 2, "Notes" : null
 }, "customer" : { "CustomerIDCard" : 17472400, "Name" : "Tyrus
Duckels", "AccountNumber" : "3.55E+15", "PhoneNumber" : "7014585
848", "EmergencyName" : "Arluene Glusby", "EmergencyPhone" : "44
34917308" }, "employee" : { "EmployeeID" : 101115, "Name" : "Gaj
ah Namani", "Title" : "Ms.", "PhoneNumber" : 242156790 }, "feedb
acks" : { "FeedbacksID" : 10230, "OccupancyNumber" : 100001, "Fe
edbackDetails" : "It's quite nice and clean here!", "Notes" : ""
 }, "Room_number" : [ { "_id" : 402, "RoomID" : 402, "RoomTypeID
" : 3, "BedTypeID" : 4, "RoomStatusID" : 3, "AmenityID" : 2, "No
tes" : null, "roomtype" : { "RoomTypeID" : 3, "RoomTypeName" : "
King" }, "roomstatus" : { "RoomStatusID" : 3, "RoomStatus" : "Re
ady for guests" }, "bedtype" : { "BedTypeID" : 4, "BedTypeName"
: "King-sized" }, "amenities" : { "AmenityID" : 2, "AmenityName"
 : "Intermediate", "Notes" : "Basic amenities plus a kitchenette
```

Figure 6. $lookup command for joining Room and Occupancies table and result.

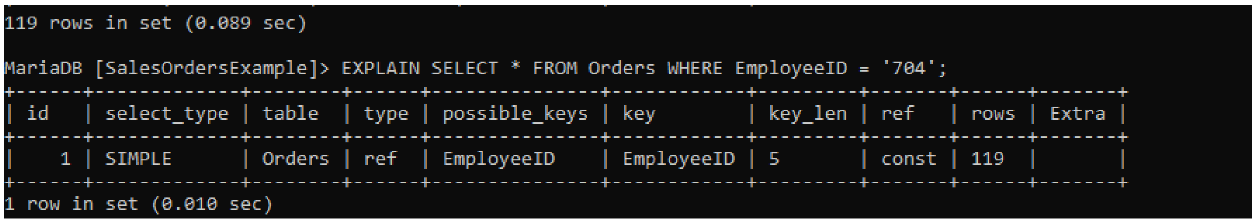


```
MariaDB [DumasHotel]> SELECT * FROM Occupancies o JOIN Room r WHERE o.RoomID = r.RoomID;
+-----------------+--------+------------+----------------+------------+------------+--------+------------+-----------+--------------+-----------+-----------------------
---------------------------+
| OccupancyNumber | RoomID | EmployeeID | CustomerIDCard | FirstDate  | LastDate   | RoomID | RoomTypeID | BedTypeID | RoomStatusID | AmenityID | Notes
                           |
+-----------------+--------+------------+----------------+------------+------------+--------+------------+-----------+--------------+-----------+-----------------------
---------------------------+
|          100014 |    101 |     101112 |       19274376 | 2019-03-03 | 2019-03-08 |    101 |          1 |         1 |            1 |         1 | Amenities needed to be
 upgraded                  |
|          100037 |    101 |     101110 |       10330509 | 2020-01-01 | 2020-01-06 |    101 |          1 |         1 |            1 |         1 | Amenities needed to be
 upgraded                  |
|          100018 |    102 |     101104 |       18293083 | 2019-05-22 | 2019-06-01 |    102 |          3 |         2 |            3 |         3 | NULL
                           |
|          100030 |    102 |     101108 |       18270465 | 2019-10-03 | 2019-10-11 |    102 |          3 |         2 |            3 |         3 | NULL
                           |
|          100010 |    103 |     101103 |       20543913 | 2019-01-22 | 2019-02-03 |    103 |          2 |         1 |            3 |         2 | NULL
```

Figure 7. Joining commandsin MySQL with similar tables to Figure 6 and result.

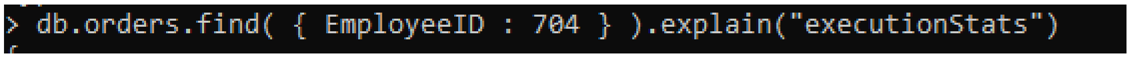
C. Performance of two database products.

When it comes to performance of MongoDB and MySQL, in reality it depends on the type of projects and databases that developers work on. It could be stated that "MongoDB can accept large amounts of unstructured data much faster than MySQL" (Smallcombe, 2020). To prove this assumption, an experiment was conducted using data from Orders collection in SalesOrdersExample database. The idea is to finding data from orders where EmployeeID is 704, and the result could be seen in following figures of 8, 9 and 10.



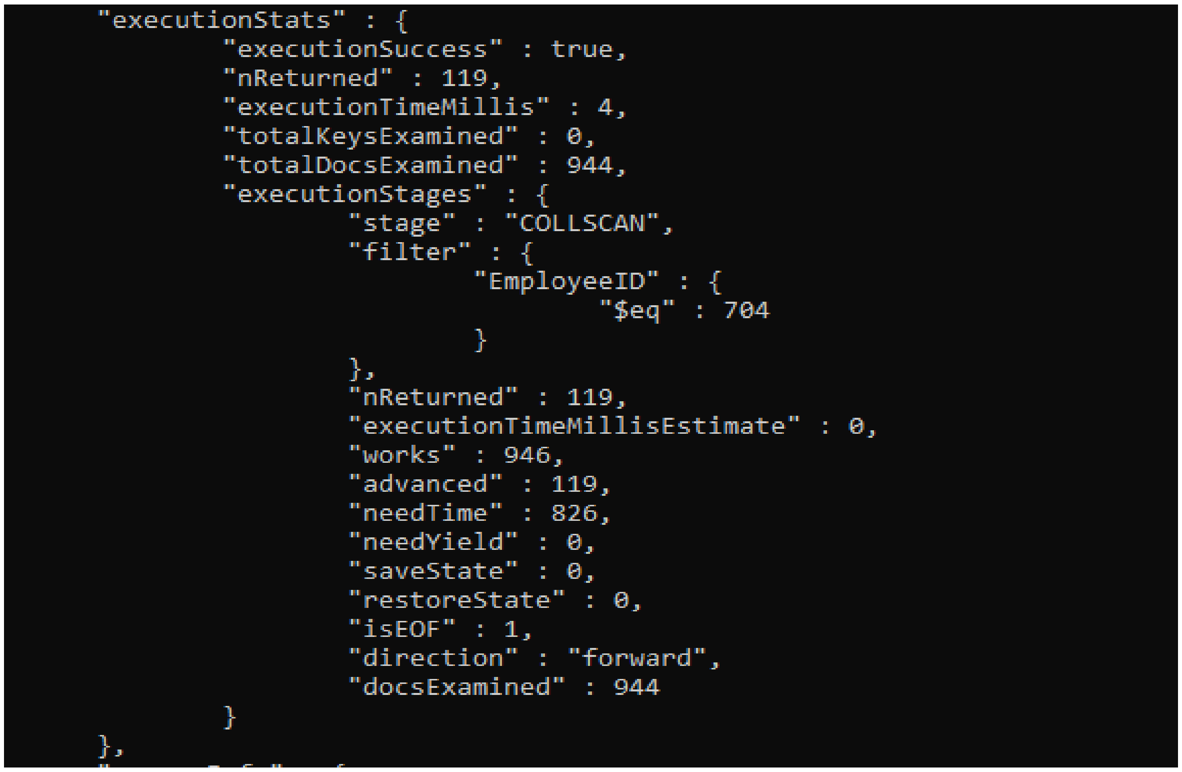
```
119 rows in set (0.089 sec)

MariaDB [SalesOrdersExample]> EXPLAIN SELECT * FROM Orders WHERE EmployeeID = '704';
+------+-------------+--------+------+---------------+------------+---------+-------+------+-------+
| id   | select_type | table  | type | possible_keys | key        | key_len | ref   | rows | Extra |
+------+-------------+--------+------+---------------+------------+---------+-------+------+-------+
|    1 | SIMPLE      | Orders | ref  | EmployeeID    | EmployeeID | 5       | const | 119  |       |
+------+-------------+--------+------+---------------+------------+---------+-------+------+-------+
1 row in set (0.010 sec)
```

Figure 8: The result of analysing performance of the query in MySQL.



```
> db.orders.find( { EmployeeID : 704 } ).explain("executionStats")
```

Figure 9: The query for analysing performance for the searching command in MongoDB.



```
	"executionStats" : {
		"executionSuccess" : true,
		"nReturned" : 119,
		"executionTimeMillis" : 4,
		"totalKeysExamined" : 0,
		"totalDocsExamined" : 944,
		"executionStages" : {
			"stage" : "COLLSCAN",
			"filter" : {
				"EmployeeID" : {
					"$eq" : 704
				}
			},
			"nReturned" : 119,
			"executionTimeMillisEstimate" : 0,
			"works" : 946,
			"advanced" : 119,
			"needTime" : 826,
			"needYield" : 0,
			"saveState" : 0,
			"restoreState" : 0,
			"isEOF" : 1,
			"direction" : "forward",
			"docsExamined" : 944
		}
	},
```

Figure 10. The result for the command in Figure 9.

It is noticeable from the example that the runtime of the query in MongoDB is 4 millliseconds (0.004 s), which is significantly faster than in MySQL, which run the query in 89 milliseconds (0.089 s). That is despite the fact that on this scenario, while MySQL used key (EmployeeID) to scan and find data needed, MongoDB did not use a key, rather it searched though the entire collection (944 documents) to find matched data. It could thus be concluded that MongoDB could have a faster runtime for a query comparing to MySQL.

But, it's worth noticing that "it's difficult to measure precisely how much faster MongoDB is than MySQL when handling large projects" (Smallcombe, M, 2020). With a straightly defined data structure, from the number of data of 10 000, the relational database like MySQL could show a better performance, and when using indexes, the finding of necessary data in MongoDB could be decelerated comparing to MySQL(Ceresnak and Chovankova, 2019). Also, the real speed that programmers would experience could be affected by a variety of factors, such as their network bandwidth, the distance between working location and database servers, and the data structure.

Generally speaking, while higher speed could be a major benefit for developers using MongoDB, its impact could be influenced by external circumstances. Consequently, programmers should also focus on other database characteristics like security or reliability.when choosing their database system to work.

D. Transaction, concurrency and isolation level.

a. Transaction and concurrency

A worth noticing characteristic of MongoDB when it comes to transaction is that , a transaction in MongoDB does not follow ACID (atomicity, consistency, isolation, and durability) principle for transaction like in MySQL. If the operation requires the modification of different embedded documents within a single document (for example, using db.collections.updateMany( )), the operation as a whole is not atomic, even when the change of data in each document could be considered atomic(MongoDB Manual, 2020).

Due to this problem involving transactions, MongoDB has several approaches to resolve the data duplication or missing data. One solution is the creation of unique indexes on a field that only possesses unique data like a primary key in MySQL, by using following command:

db.collection.createIndex(<key and index characteristics>, {unique : true})

This type of command could also be used on multiple fields, which is similar to a complex primary key for a table in MySQL.

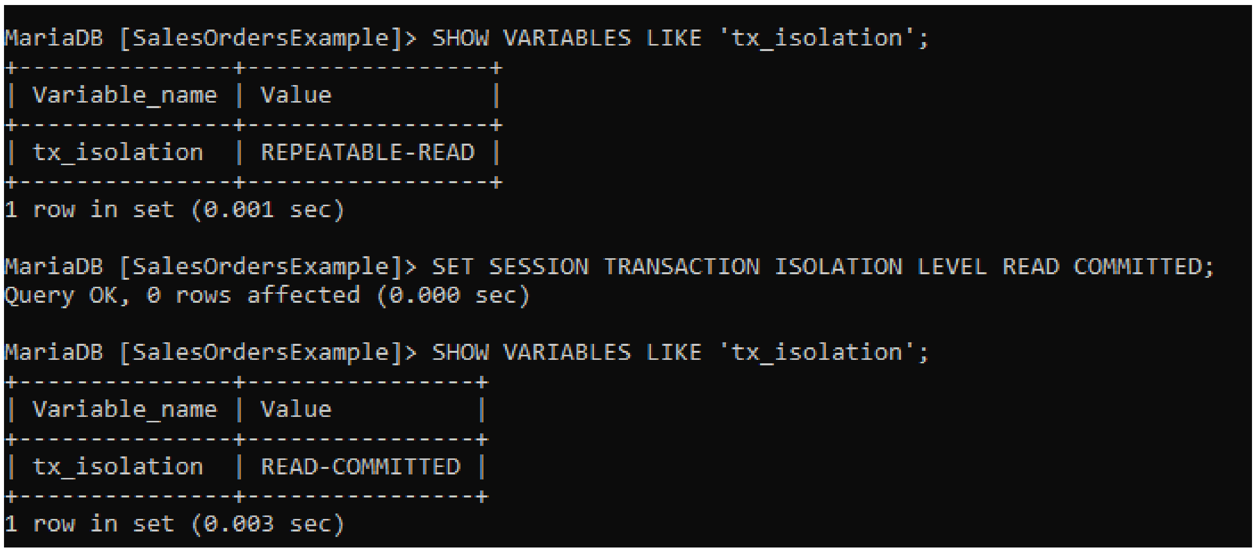
Another approach is citing the expected current value of a field for querying write operations.

b. Comparison for isolation levels

By default, the isolation level of MongoDB when not using sessions is read uncommited, which also applies for replicated sets and sharded clusters. (For the case of using sessions, the default level of isolation is snapshot).Particularly for a multi-document transaction, a transaction could not commit some of its changes while rolling back the others, and the mismatch of data updated during a read operation could happen. All of them means that “MongoDB uses dirty read semantics, which includes the possibility for doubled or missing records” (Allen, 2016).

To resolve this problem, from version 3.6 MongoDB introduced “causal consistency”, which means that it could implement causal operations with respect to the relationship of the sessions. However, this method is used only for read and write sessions with “majority” concern, that is when document read are guarantee not to be rolled back, and not for a multi-document transaction.

By contrast, MySQL has more isolation levels (read uncommited, read committed, repeatable read and serializable), and programmers could change their isolation level used to be more suitable with their demand for speed or for consistency, which could be seen in Figure 11.



```
MariaDB [SalesOrdersExample]> SHOW VARIABLES LIKE 'tx_isolation';
+---------------+-----------------+
| Variable_name | Value           |
+---------------+-----------------+
| tx_isolation  | REPEATABLE-READ |
+---------------+-----------------+
1 row in set (0.001 sec)

MariaDB [SalesOrdersExample]> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0.000 sec)

MariaDB [SalesOrdersExample]> SHOW VARIABLES LIKE 'tx_isolation';
+---------------+----------------+
| Variable_name | Value          |
+---------------+----------------+
| tx_isolation  | READ-COMMITTED |
+---------------+----------------+
1 row in set (0.003 sec)
```

Figure 11: The default isolation level of the database in MySQL, and the method to change it to read committed.

While MySQL by default support repeatable read level, it could also support snapshot semantics, which is used concurrently with read committed to improve the performance. Consequently, it is probable that developers using MySQL face less problems about loss update and dirty read – which could be considered an advantage of MySQL over MongoDB.

E. <u>MapReduce</u>

An unique feature of MongoDB ,compared to MySQL, is its support for MapReduce model, which is, according to IBM, "a programming paradigm that enables massive scalability across hundreds or thousands of servers in a Hadoop cluster".

Comparing to SELECT statements in MySQL, MapReduce could enable the access of multiple sources and types of data to users in diffferent nodes on a cluster of servers. Thanks to this ability, a tremendous amount of data could be rapidly retrieved and processed, giving to users insights about the data – which could be a major advantage in the time of big data.

F. Summary

In the end, it is on account of the users to choose the most suitable database products to resolve their problem.

Here is a table citing strength and weaknesses of each database system examined in the report.

	MySQL	MongoDB
Advantages	+ Have a simple, easy to learn data structure. + Strict schema and flexible isolation level support. + Can support joining multiple tables. +Ease in management.	+ Flexible schema. + Can support unnormalized data structure, as well as unstructured and semi-structured data. + Generally better performance. + Could be used for voluminous amount of data by tools like MapReduce.
Disadvantages	+ Difficulties in scalability (compared to MongoDB). + Slower runtime. + Need to normalize data to prevent duplication	+ Only permit to join two tables. + Does not perform well for complex transactions + Vulnerable to dirty read.

IV. Conclusions

There is not a database product that could be suitable for all problems faced by developers and administrators, and there are numerous factors that could affect to the efficiency of database systems used. In reality, the choice of database system depends on the demands and requirements set from users. In case when complex transactions is needed, such as in accounting systems, and in legacy systems that are not designed for a relational database, MySQL would be the best suited for the task. In other applications where there is no need for a clear schema definition, or the data is unstructured or structured with high growth potential like real-time analytics, mobile apps and internet of things, MongoDB could be truly supportive for their development (Sarig, 2017).

V. References

Allen, J, *A Quick Primer on Isolation Levels and Dirty Reads*, InfoQ, viewed 30 October 2020, < https://www.infoq.com/articles/Isolation-Levels/ >

Ceresnak, R, Chovankova, O, 2019, '*Comparison of Query Performance Between MySQL and MongoDB Database*', Central European Researchers Journal, Vol.5 Issue 1, pp. 45-51.

Gall, R, 2019, *Different types of NoSQL databases and when to use them*, Packt Hub, viewed 28 October 2020, < https://hub.packtpub.com/different-types-of-nosql-databases-and-when-to-use-them/ >

IBM, 2020, *What is Apache MapReduce?* ,viewed 30 October 2020, < https://www.ibm.com/analytics/hadoop/mapreduce >

MongoDB Manual, 2020,*Atomicity and Transactions*, last accessed 30 October 2020, < https://docs.mongodb.com/manual/core/write-operations-atomicity/ >

MongoDB Manual, 2020,*Data Model Design*, last accessed 30 October 2020, < https://docs.mongodb.com/manual/core/data-model-design/ >

Prateek, T.L, 2019,*What is Normalization in SQL and what are its types?* , Edureka!, viewed 29 October 2020, < https://www.edureka.co/blog/normalization-in-sql/ >

Sarig, M, 2017, *MongoDB Vs MySQL: The Differences Explained*, Panoply, viewed 30 October 2020, < https://blog.panoply.io/mongodb-and-mysql >

Smallcombe, M, 2020, *MongoDB vs. MySQL: Detailed Comparison of Performance and Speed*, Xplenty, viewed 30 October 2020, < https://www.xplenty.com/blog/mongodb-vs-mysql/ >

StackOverflow, 2020, *2020 Developer Survey*, viewed 28 October 2020, < https://insights.stackoverflow.com/survey/2020#technology-databases-professional-developers4 >