



BBM 471 Database Management Systems Experiment

Subject: NoSQL Databases - MongoDB

Berk Kemal Özata
21328298

Ekrem Candemir
21327771

INTRODUCTION

NoSQL (Not-only-SQL) was used by Carlo Strozzi in 1998. The NoSQL concept has been introduced as an alternative to RDBMS. One of the biggest factors of the occurrence of NoSQL database system is Internet. The increase of active users on the internet caused the database schema to change more frequently. Changing the database schema is a costly process. Also, as the number of active users increased, the amount of data to be stored increased. We need to update this table according to the new information in order to add information that is not defined in the relational database to the table. The tables need to be updated according to the new information in order to add a non-defined information to the table in the relational database. After doing this, we need to update the other tables related to this table. Bigger companies like Google (Big Table) and Amazon (DynamoDB) use NoSQL database systems for years.

NoSQL

NoSQL is a horizontally scalable database system. NoSQL systems do not have table and column concepts. The data is stored in a structure similar to JSON and XML instead of tables and columns. If a new column is needed, we can not register on NoSQL systems. NoSQL automatically creates the required field. The data is stored in a structure similar to JSON and XML instead of tables and columns. If a new column field is needed, it is enough to add a new record in NoSQL systems without adding this column field. NoSQL automatically creates the required column field. Due to this reason, changing the recorded information does not constitute a huge cost in NoSQL systems. NoSQL was mentioned to be able to scale horizontally. This means that vertical growth if the system is inadequate, you need to renew that system or buy new hardware. Horizontal growth can also be done by taking additional servers. NoSQL systems also reduce costs with these features. There is no exact standard for NoSQL systems. However, we can specify 4 general groups:

Key-Value Stores: These modeled data consists of a unique key and a value pair associated with that key. It is extremely powerful for web applications. For example Aerospike, Redis and Berkeley DB.

Document Stores: Documents are generally kept in JSON format on this system. Developers can create and update records without needing any main schemes. Examples are DocumentDB, CouchDB, and MongoDB that we use in this project.

Wide-Column Stores: Wide-column stores hold data in columns instead of rows. Wide-column stores in large data-intensive systems can query faster than RDBMSs. Examples are Google BigTable and Cassandra.

Graph Stores: This differs from the others in that it also holds relationships between the data. Examples are AllegroGraph and IBM Graph.

T Y P E S	PERFORMANCE	SCALABILITY	FLEXIBILITY	COMPLEXITY
KEY-VALUE STORE	high	high	high	none
COLUMN STORE	high	high	moderate	low
DOCUMENT	high	variable (high)	high	low
GRAPH DATABASE	variable	variable	high	high

NoSQL vs RDBMS DIFFERENCES

RDBMS is a vertically scalable database system, NoSQL system is a horizontally scalable database system. As mentioned above, if you want to increase the storage space in vertically growing systems, you have to purchase additional hardware or change your existing hardware. In horizontal growing systems, you can store additional data by receiving additional servers.

RDBMSs are transaction based. RDBMSs have ACID (Atomicity, Consistency, Isolation, Durability) acronym for data integrity and system stability.

In short ACID:

Atomicity: The other process does not start until a process is finished. Transactions can not be interrupted.

Consistency: Transactions are compatible. A transaction must conform to rules such as the foreign key defined in the diagram.

Isolation: Operations are performed in isolation from each other. An incomplete operation can not see other connections.

Durability: Once data has been written to the database, that data is there. Whatever happens, the recorded data is not affected.

On NoSQL systems, none of these rules exist. Even many NoSQL systems do not even have transactions. NoSQL systems have BASE acronym.

In short BASE:

Basically available: The data is always available but the data may not be in the most current state.

Soft State: There may be changes even in the system without input time.

Eventual Consistency: The system will be consistent over time because the system does not receive input.

ACID	BASE
Strong Consistency	Weak Consistency
Isolation	Availability First
Focus on "commit"	Best Effort
Nested transactions	Approximated answers
Less Availability	Simpler
Conservative(pessimistic)	Aggressive(optimistic)
Difficult evolution(e.g. schema)	Easier Evolution
	Faster

NoSQL can gain performance by providing horizontal growth. For example, by dividing a data, we can send copies of it to different parts of the distributed system. You do not lose all of it on this count.

There are no tables in NoSQL systems while they are stored in tables and columns given in RDBMS. The data to be added in the RDBMS should be added appropriately to the table. In the RDBMS, that information can not be added to a column of information that is not in a table. However, the table can be changed to suit the information and new information can be added to it. NoSQL systems do not have a specific table definition, so the requested information can be added. NoSQL automatically adds the required fields. In the RDBMS, a join is performed to access the data. In NoSQL, this can be problematic. Because, as mentioned above, the data to be reached may be in different parts.

Therefore, when designing NoSQL schema we have to consider some constraints. One of the is combining objects into one document, if these objects will be used together. So that, we don't need to join these objects when querying. Also we can duplicate the data if we need these objects separate when read time because disk space is cheap.

In RDBMS, primary keys are not mandatory. In NoSQL, primary keys are required. Because the data is accessed via primary keys.

Some of the RDBMS are open source, such as Postgres, MySQL, etc. Some are closed source such as Oracle Database. NoSQL systems are open-source.

WHY SHOULD YOU CHOOSE NOSQL?

- NoSQL database systems offer higher accessibility compared to RDBMS.
- NoSQL database systems have better read and write performance than RDBMS.
- NoSQL database systems can be expanded horizontally. NoSQL systems can hold huge amounts of data thanks to additional servers without having to purchase new hardware.
- With NoSQL's non-relational nature, there is no need to develop a detailed database schema. This feature saves a lot of time for developers.

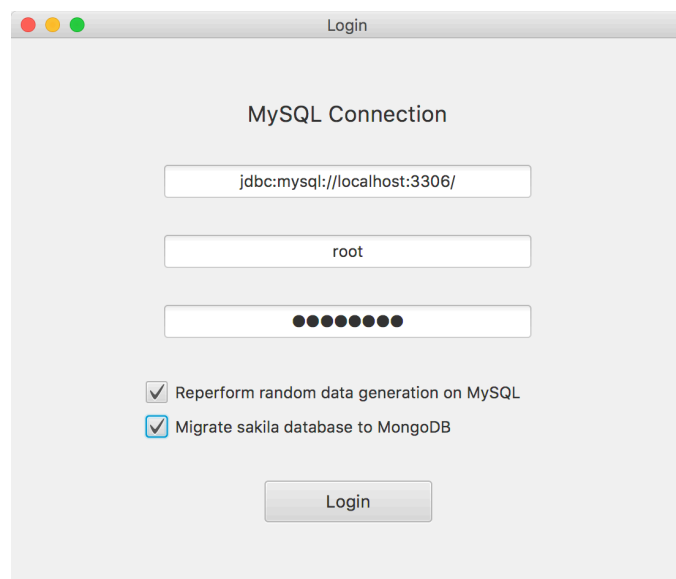
WHY SHOULD YOU CHOOSE RDBMS?

- It is known what columns are for a selected row.
- In most RDBMSs, authentication and access privileges are better than NoSQL.
- You can use SQL. SQL has a standard and SQL is well documented and stable.
- RDBMSs have ACID acronym..

OUR PROGRAM

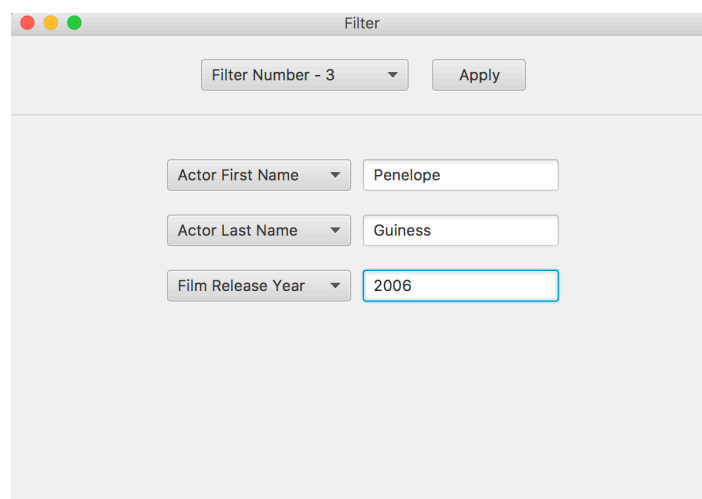
Our program consists of 3 main windows. The first window is the login screen. On this screen, a connection-URL to connect to the MySQL database, a username, and password defined in this database must be entered. At the same time, this window has 2 checkboxes. When the first checkbox is checked, the reproduction is performed on the data using the schema and the data of the "Sakila" database. When the second checkbox is checked, the tables and data in the connected MySQL database are transferred to MongoDB.

The design in MongoDB includes only films collection. Each film document consists of its attributes and also array of categories it belongs to and array of actors in that film.

A screenshot of a macOS-style window titled "Login". The window has a light gray background and standard red, yellow, and green window control buttons in the top-left corner. The title "MySQL Connection" is centered at the top. Below it are three text input fields: the first contains "jdbc:mysql://localhost:3306/", the second contains "root", and the third contains ten black dots representing a password. Below the password field are two checkboxes. The first checkbox is checked and labeled "Reperform random data generation on MySQL". The second checkbox is also checked and labeled "Migrate sakila database to MongoDB". At the bottom center is a "Login" button.

Login Screen

The second window is where the user can enter the data they want to query. At first, the user is asked how many filters he/she wants to use. There are a total of five filters: film.title, film.release_year, actor.first_name, actor.last_name, and category.name. ComboBox and text field pop up as the user-selected number. The user fills in the text fields of the related filter fields by the number of filters selected here. Then the user must press the "Apply" button.

A screenshot of a macOS-style window titled "Filter". The window has a light gray background and standard red, yellow, and green window control buttons in the top-left corner. At the top, there is a dropdown menu labeled "Filter Number - 3" and an "Apply" button. Below this, there are three rows of filter controls. Each row consists of a dropdown menu and a text input field. The first row has a dropdown labeled "Actor First Name" and a text field containing "Penelope". The second row has a dropdown labeled "Actor Last Name" and a text field containing "Guinness". The third row has a dropdown labeled "Film Release Year" and a text field containing "2006".

Filter Screen

The third window is the window where the results are displayed. MySQL on the left and MongoDB databases on the right side show the query execution time, table rendering time, and the result of the queried data.

MySQL				MongoDB			
Query Execution Time: 149 ms Table Rendering Time: 197 ms				Query Execution Time: 0 ms Table Rendering Time: 454 ms			
film_id	actor_id	title		_id	title		
1	1	ACADEMY DINOSAUR	A Epic Drama of a Femin	2	ACE GOLDFINGER	A Astounding Epistle of a Dat	
2	1	ACE GOLDFINGER	A Astounding Epistle of a	4	AFFAIR PREJUDICE	A Fanciful Documentary of a	
3	1	ADAPTATION HOLES	A Astounding Reflection c	8	AIRPORT POLLOCK	A Epic Tale of a Moose And a	
4	1	AFFAIR PREJUDICE	A Fanciful Documentary c	9	ALABAMA DEVIL	A Thoughtful Panorama of a l	
5	1	AFRICAN EGG	A Fast-Paced Documenta	13	ALI FOREVER	A Action-Packed Drama of a	
6	1	AGENT TRUMAN	A Intrepid Panorama of a	24	ANALYZE HOOSIERS	A Thoughtful Display of a Exp	
7	1	AIRPLANE SIERRA	A Touching Saga of a Hur	30	ANYTHING SAVANNAH	A Epic Story of a Pastry Chef	
8	1	AIRPORT POLLOCK	A Epic Tale of a Moose Ar	34	ARABIA DOGMA	A Touching Epistle of a Madn	
9	1	ALABAMA DEVIL	A Thoughtful Panorama o	35	ARACHNOPHOBIA ROLLERCOASTER	A Action-Packed Reflection c	
10	1	ALADDIN CALENDAR	A Action-Packed Tale of a	65	BEHAVIOR RUNAWAY	A Unbelieveable Drama of a S	
11	1	ALAMO VIDEOTAPE	A Boring Epistle of a Butl	92	BOWFINGER GABLES	A Fast-Paced Yarn of a Waitr	
12	1	ALASKA PHANTOM	A Fanciful Saga of a Hunt	122	CARRIE BUNCH	A Amazing Epistle of a Stude	
13	1	ALI FOREVER	A Action-Packed Drama c	171	COMMANDMENTS EXPRESS	A Fanciful Saga of a Student	
14	1	ALICE FANTASIA	A Emotional Drama of a A	222	DESERT POSEIDON	A Brilliant Documentary of a l	
15	1	ALIEN CENTER	A Brilliant Drama of a Cat	258	DRUMS DYNAMITE	A Epic Display of a Crocodile	
16	1	ALLEY EVOLUTION	A Fast-Paced Drama of a	275	EGYPT TENENBAUMS	A Intrepid Story of a Madmar	
17	1	ALONE TRIP	A Fast-Paced Character S	277	ELEPHANT TROJAN	A Beautiful Panorama of a Lu	
18	1	ALTER VICTORY	A Thoughtful Drama of a	301	FAMILY SWEET	A Epic Documentary of a Tea	
19	1	AMADEUS HOLY	A Emotional Display of a f	313	FIDELITY DEVIL	A Awe-Inspiring Drama of a T	

Result Screen

RESULTS

We observe that the queries that require joining tables NoSQL side outperforms. Because we hold the whole data in one collection called films. We can consider it as joined version of all tables in terms of SQL. On the SQL side, before querying join must be performed.

When we filter by only actor or category, SQL side just performs querying on these tables. But on NoSQL side, we have to build a pipeline that extracts distinct actors/categories that matches the query. In this case SQL outperforms.

	MySQL	MongoDB
actor.first_name = DUSTIN	6 ms	1039 ms
film.release_year = 2006 category.name = Horror actor.first_name = PENELOPE	349 ms	0 ms
category.name = Action	1 ms	511 ms

REFERENCES

- 1) <http://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL>
- 2) <https://www.mongodb.com/nosql-explained>
- 3) <http://devveri.com/nosql-nedir>
- 4) <https://blog.kodcu.com/2014/03/nosql-nedir-avantajlari-ve-dezavantajlari-hakkinda-bilgi/>
- 5) <https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/>