REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie

Département d'informatique

N° d'ordre : SIOD23/M2/2022

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Systèmes d'information, Optimisation et Décision (SIOD)**

SQL , NOSQL and NewSQL Databases: A Theoretical and Practical Comparative Survey

Par :
MELILI ZAKARIA

Soutenu le 27/06/2022 devant le jury composé de :

Guemeida Abdelbasset	MCB	Président
Ben seghier Nadia	MCA	Rapporteur
Meklid Abdessalam	MAA	Examinateur

Année universitaire 2021-2022

Acknowledgments

I want to express by these few lines of thanks at the end of this modest work, the great gratitude to:

We begin by thanking **Dr. Ben Seghier Nadia** who did me the honor of being my supervisor for her follow-up and her enormous support and all the help she provided.

Also address our sincere thanks to the members of the juries for having kindly examined and judged this work.

Abstract

This project adds to the various research works in the field of Sql, NoSQL and Newsql. Each model of them offers a way of organizing and storing data. The purpose of this work is to explain the different SQL, NoSQL and Newsql databases available on the market. And to see in more depth six of them, namely: Mysql, Oracle, Mongodb, Cassandra,Voltdb,Nuodb to offer decision-makers information for possible choices of the best appropriate solution for their businesses. We used multiple tests: insert, update and delete to calculate the execution time of each one of theme and that will used to decide between these solutions.

Key-Words: SQL, NoSql , NewSql , Mysql , Oracle , Mongodb , Cassandra, Voltdb, Nuodb, Run time.

Table of content

List of Figures

List of tables

General Introduction

SQL Databases additionally called RDBMS (Relational Database Management Systems) is the maximum not unusual place and conventional technique to database solutions. The record is saved in a based manner in shape of tables or Relations. With creation of Big Data but, the based technique falls brief to serve the wishes of Big Data structures which might be ordinarily unstructured in nature. Increasing capability of SQL despite the fact that permits large quantity of records to be managed, it does now no longer virtually be counted number as a method to Big Data wishes, which expects rapid reaction and short scalability. To remedy this trouble a brand new type of Database machine known as NoSQL turned into added to offer the scalability and unstructured platform for Big Data applications. NoSQL stands for Not Only SQL. NoSQL databases encompass key value pair, Documents, graph databases or wide – column shops which do now no longer have a widespread schema which it wishes to follow. It is likewise horizontally Scalable instead of vertical scaling in RDBMS. NoSQL furnished brilliant guarantees to be a really perfect database machine for Big Data applications; it but falls brief due to a few principal drawbacks like NoSQL does now no longer assure ACID residences (Atomicity, Consistency, Isolation and Durability) of SQL structures. It is likewise now no longer well matched with in advance variations of database. This is in which NewSQL comes into picture. NewSQL is a modern day improvement with inside the global of database structures. NewSQL is a Relational Database with the scalability residences of NoSQL.

As part of this graduation project, SQL, NoSql and Newql Databases: A Theoretical and Practical Comparative Survey, to guide users towards the most appropriate solutions according to their needs.

To better organize this document, we structured it as follows:

In the first chapter: we will present the evolution of the different generations of data management systems SQL, NOSQL and NewSQL.

The second chapter: will be our theoretical comparative study between the different generations of data management systems

The third chapter: will be the subject of the different experimental results obtained after the execution of a set of tests as well as an interpretation of the performance evaluation results of each database.

Chapter I : SQL , NOSQL and NEWSQL Databases

1. INTRODUCTION :

Database is a collection of data that could be used alone or combined with other data. DBMS (Database Management system) is a software that allows the computer to perform database function of storing, retrieving, adding, deleting and modifying data .he is a collection of interrelated data, set of programs to access the data and an environment that is both convenient and efficient to use.

2. SQL :

2.1. Definition :

Structured query language(SQL), computer language designed for eliciting information from databaes.

In the 1970s computer scientists began developing a standardized way to manipulate databases, and out of that research came SQL. The late 1970s and early ’80s saw the release of a number of SQL-based products. SQL gained popularity when the American National Standards Institute (ANSI) adopted the first SQL standard in 1986. Continued work on relational databases led to improvements in SQL, making it one of the most popular Database languages in existence.

SQL works by providing a way for programmers and other computer users to get desired information from a database using something resembling normal English. On the simplest level, SQL consists of only a few commands: Select, which grabs data, Insert, which adds data to a database , Update, which changes information , and Delete, which deletes information. Other commands exist to create, modify, and administer databases. [1]

2.2. Architecture :

There are two types of architecture: Physical and Logical. Architecture . [2]

- Physical architecture tells about how the data is actually stored in file system of an operating system. Page, extent, database files, transaction log files etc. are core components of physical architecture.

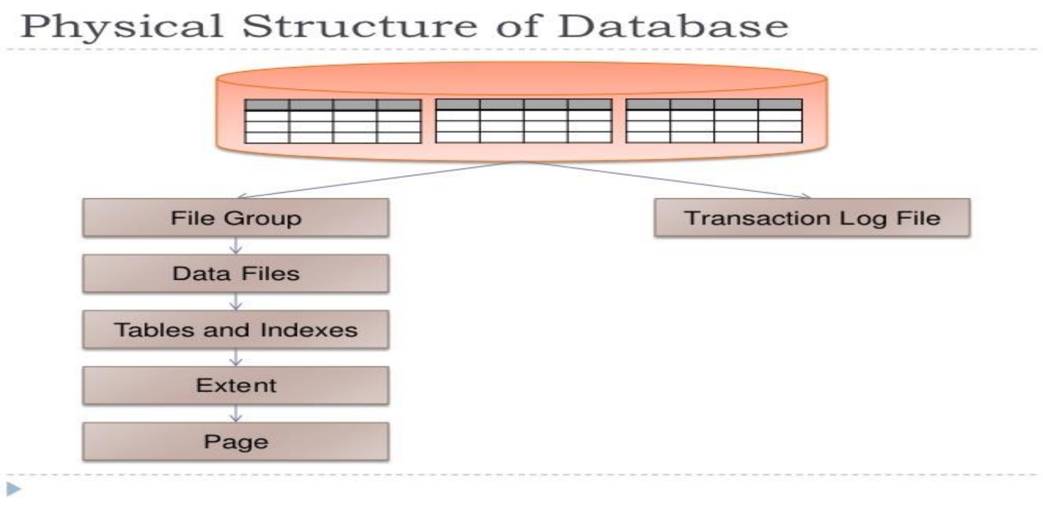


Figure 1: Physical architecture

- Logical architecture tells about how the data is logically grouped and presented to the user. Tables, constraints, views, stored procedures, functions, triggers etc. are core components of logical architecture. [1]

2.3. ACID :

In the context of computer science, ACID stands for: Atomicity, Consistency, Isolation, Durability.

Together, ACID is a set of guiding principles that ensure database transactions are processed reliably. A database transaction is any operation performed within a database, such as creating a new record or updating data within one.

Changes made within a database need to be performed with care to ensure the data within doesn't become corrupted. Applying the ACID properties to each modification of a database is the best way to maintain the accuracy and reliability of a database.
and each component of ACID means : [3]

- Atomicity :

In the context of databases, atomicity means that you either:

- Commit to the entirety of the transaction occurring
- Have no transaction at all

Essentially, an atomic transaction ensures that any commit you make finishes the entire operation successfully. Or, in cases of a lost connection in the middle of an operation, the database is rolled back to its state prior to the commit being initiated.
This is important for preventing crashes or outages from creating cases where the transaction was partially finished to an unknown overall state. If a crash occurs during a transaction with no atomicity, you can't know exactly how far along the process was before the transaction was interrupted. By using atomicity, you ensure that either the entire transaction is successfully completed or that none of it was.

- Consistency :

Consistency refers to maintaining data integrity constraints.
A consistent transaction will not violate integrity constraints placed on the data by the database rules. Enforcing consistency ensures that if a database enters into an illegal state (if a violation of data integrity constraints occurs) the process will be aborted and changes rolled back to their previous, legal state.

Another way of ensuring consistency within a database throughout each transaction is by also enforcing declarative constraints placed on the database.

An example of a declarative constraint might be that all customer accounts must have a positive balance. If a transaction would bring a customer account into a negative balance, that transaction would be rolled back. This ensures changes are successful at maintaining data integrity or they are canceled completely.

- Isolation :

Isolated transactions are considered to be “serializable”, meaning each transaction happens in a distinct order without any transactions occurring in tandem.

Any reads or writes performed on the database will not be impacted by other reads and writes of separate transactions occurring on the same database. A global order is created with each transaction queueing up in line to ensure that the transactions complete in their entirety before another one begins.

Importantly, this doesn’t mean two operations can’t happen at the same time. Multiple transactions can occur as long as those transactions have no possibility of impacting the other transactions occurring at the same time.
Doing this can have impacts on the speed of transactions as it may force many operations to wait before they can initiate. However, this tradeoff is worth the added data security provided by isolation.
Isolation can be accomplished through the use of a sliding scale of permissiveness that goes between what are called optimistic transactions and pessimistic transactions:

- An optimistic transaction schema assumes that other transactions will complete without reading or writing to the same place twice. With the optimistic schema, both transactions will be aborted and retried in the case of a transaction hitting the same place twice.
- A pessimistic transaction schema provides less liberty and will lock down resources on the assumption that transactions will impact other ones. This results in fewer abort and retries, but it also means that transactions are forced to wait in line for their turn more often in comparison to the optimistic transaction approach.

Finding a sweet spot between these two ideals is often where you’ll find the best overall result.

- Durability :

The final aspect of the ACID approach to database management is durability.
Durability ensures that changes made to the database (transactions) that are successfully committed will survive permanently, even in the case of system failures. This ensures that the data within the database will not be corrupted by:

- Service outages
- Crashes
- Other cases of failure

Durability is achieved through the use of changelogs that are referenced when databases (or portions of the database) are restarted. [3]

2.4. Limitations of SQL :

- Maintenance Problem :

The maintenance of the relational database becomes difficult over time due to the increase in the data. Developers and programmers have to spend a lot of time maintaining the database. [4]

- Physical Storage :

A relational database is comprised of rows and columns, which requires a lot of physical memory because each operation performed depends on separate storage. The requirements of physical memory may increase along with the increase of data. [3]

- Decrease in performance over time :

The relational database can become slower, not just because of its reliance on multiple tables. When there is a large number of tables and data in the system, it causes an increase in complexity. It can lead to slow response times over queries or even complete failure for them depending on how many people are logged into the server at a given time. [4]

- Lack of Scalability :

While using the relational database over multiple servers, its structure changes and becomes difficult to handle, especially when the quantity of the data is large. Due to this, the data is not scalable on different physical storage servers. Ultimately, its performance is affected lack of availability of data and load time etc. As the database becomes larger or more distributed with a greater number of servers, this will have negative effects like latency and availability issues affecting overall performance. [4]

- Complexity in Structure :

Relational databases can only store data in tabular form which makes it difficult to represent complex relationships between objects. This is an issue because many applications require more than one table to store all the necessary data required by their application logic. [4]

3. NoSQL :

3.1. What is NoSQL :

The rise of Big Data created a demand for horizontally scalable Data Management System. This led to development of different kinds of Database Management System which collectively come under NoSQL. NoSQL Databases are broadly divided into following types:

Document, Graph, Native XML, Key-value, Native object, Table type, and Hybrid Databases. All RDBMS databases are based on the same model, whereas, each of the NoSQL database follows a different model. NoSQL moves away from the hefty standardized form of SQL database and enables simpler data storage solutions. Thus a NoSQL database is optimized for the specific application.[2]

3.2. Architecture Patterns of NoSQL:

Architecture Pattern is a logical way of categorizing data that will be stored on the Database. NoSQL is a type of database which helps to perform operations on big data and store it in a valid format. It is widely used because of its flexibility and a wide variety of services.
The data is stored in NoSQL in any of the following four data architecture patterns. [5]

3.2.1. *Key-Value Store Database:*

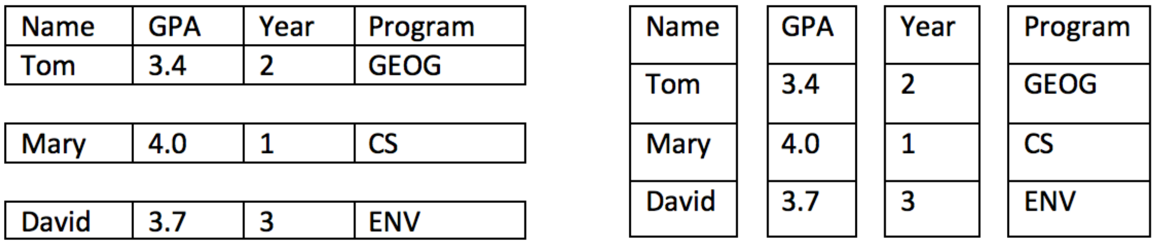
Key-Value is the most intuitive NoSQL data store. Every data item in the database is stored as a key-value pair, similar to a conventional dictionary. A key is typically a unique ID that points to the data with which it is associated. Possible operations to use with key-value include getting the value for a key (get), putting or assigning a value for a key (put), and deleting a key (delete). The value can be any object such as string, number, date, array, etc., so the system is scheme-less. The application is responsible for understanding the type of object and parsing it accordingly. Basho's Riak and Amazon's Dynamo are the most well-known key-value store NoSQL databases.
An example of key-value store is shown in Table 1 below. Note that the address key is associated with a value that contains three attributes: street, city, and state. In a relational database, such a store would be invalid. [5]

Key	**Value**
ID	"a78d1238dfedhz"
Building	"COOR Hall"
Address	{ street: "976 S Forest Mall", city: "Tempe", state: "Arizona" }

Tableau 1:Example of a key-value store.

3.2.2. *Column Store Database:*

Supports data as columns instead of rows, an arrangement that is usually optimized for queries over large datasets. Querying over rows is memory intensive and requires huge disk access especially when each row contains many columns. With column store, columns are grouped into column families, and each column family can have an unlimited number of columns. In this way it is much easier to query the entire collection of columns for all the rows. Google's BigTable, HBase, and Cassandra are the most popular column store based databases.
In the example below, the different structures of row-based store (left) and column-based store (right) are shown . [4]



Name	GPA	Year	Program
Tom	3.4	2	GEOG
Mary	4.0	1	CS
David	3.7	3	ENV

Name
Tom
Mary
David

GPA
3.4
4.0
3.7

Year
2
1
3

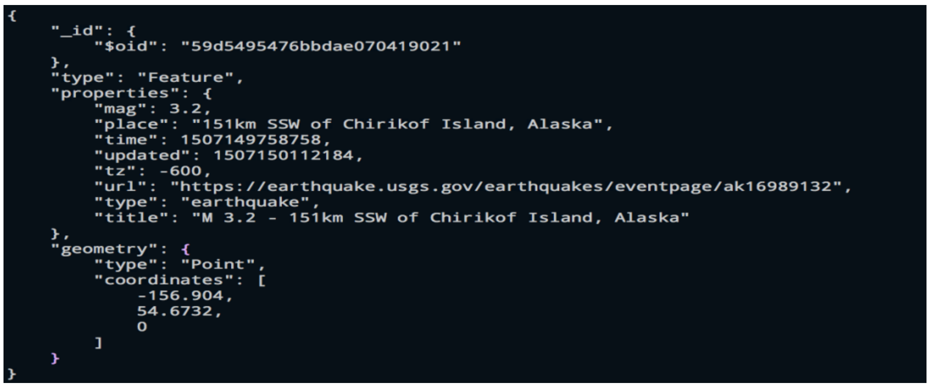
Program
GEOG
CS
ENV

Figure 2: Row-based store versus column-based store.

3.2.3. *Document Database:*

Its similar to key-value store with one difference: it requires that its value stored (called a document) be structured and encoded by metadata. Common encodings include XML, JSON, BSON, and for spatial data, GeoJSON is well accepted by many document store NoSQL databases. MongoDB and Apache CouchDB are examples of this category.

The figure below is showing an earthquake event stored in a MongoDB as a document. The document is in GeoJSON format with earthquake properties and its geometry. [5]



```
{
    "_id": {
        "$oid": "59d5495476bbdae070419021"
    },
    "type": "Feature",
    "properties": {
        "mag": 3.2,
        "place": "151km SSW of Chirikof Island, Alaska",
        "time": 1507149758758,
        "updated": 1507150112184,
        "tz": -600,
        "url": "https://earthquake.usgs.gov/earthquakes/eventpage/ak16989132",
        "type": "earthquake",
        "title": "M 3.2 - 151km SSW of Chirikof Island, Alaska"
    },
    "geometry": {
        "type": "Point",
        "coordinates": [
            -156.904,
            54.6732,
            0
        ]
    }
}
```

Figure 3: Example of an event stored in a MongoDB database.

3.2.4. *Graph Databases:*

Graph databases replace relational tables with structured relational graphs of interconnected key-value pairings. They are similar to object-oriented databases as the graphs are represented as an object-oriented network of nodes (conceptual objects), node relationships (edges) and properties (object attributes expressed as key-value pairs). They are the only of the four NoSQL types discussed here that concern themselves with relations, and their focus on visual representation of information makes them more human-friendly than other NoSQL DMS.[6]

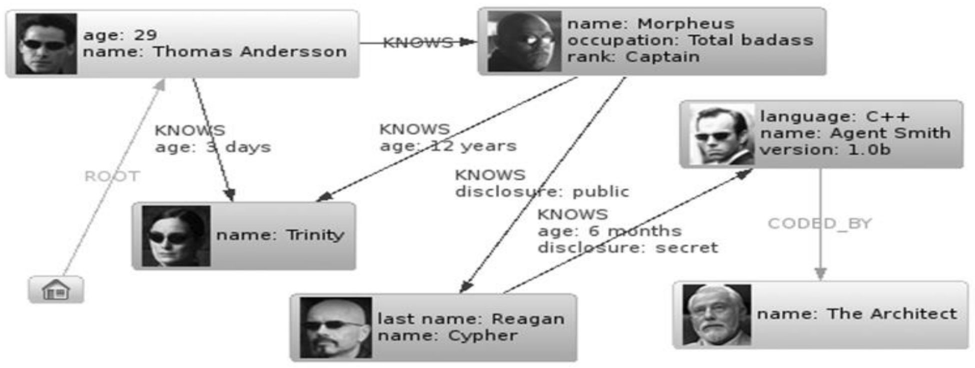


Figure 4: Graph NoSQL Database

3.3. BASE Properties:

NoSQL relies upon a softer model known, appropriately, as the BASE model. This model accommodates the flexibility offered by NoSQL and similar approaches to the management and curation of unstructured data. BASE consists of three principles: [7]

3.3.1. *Basic Availability :*

The NoSQL database approach focuses on the availability of data even in the presence of multiple failures. It achieves this by using a highly distributed approach to database management. Instead of maintaining a single large data store and focusing on the fault tolerance of that store, NoSQL databases spread data across many storage systems with a high degree of replication. In the unlikely event that a failure disrupts access to a segment of data, this does not necessarily result in a complete database outage. [7]

3.3.2. *Soft State :*

BASE databases abandon the consistency requirements of the ACID model pretty much completely. One of the basic concepts behind BASE is that data consistency is the developer's problem and should not be handled by the database. [7]

3.3.3. *Eventual Consistency :*

The only requirement that NoSQL databases have regarding consistency is to require that at some point in the future, data will converge to a consistent state. No guarantees are made, however, about when this will occur. That is a complete departure from the immediate consistency requirement of ACID that prohibits a transaction from executing until the prior transaction has completed and the database has converged to a consistent state.[7]

3.4. limitations of NoSQL :

- Lack of Standardization: [8]

There is no standard that defines rules and roles of NoSQL databases. The design and query languages of NoSQL databases vary widely between different NoSQL products – much more widely than they do among traditional SQL databases.

- Backup of Database :

Backups are a drawback in NoSQL databases. Though some NoSQL databases like MongoDB provide some tools for backup, these tools are not mature enough to ensure proper complete data backup solution.

- Consistency:

NoSQL puts a scalability and performance first but when it comes to a consistency of the data NoSQL doesn't take much consideration so it makes it little insecure as compared to the relational database e.g. , in NoSQL databases if you enter same set of data again, it will take it without issuing any error whereas relational databases ensure that no duplicate rows get entry in databases.[8]

4. NewSQL :

4.1. What is NewSQL :

NewSQL systems can be considered as a modification to the traditional SQL systems where it is found to tackle some of SQL shortcomings, as it runs on several nodes on many datacenters giving the authority of data management to the local system.
NewSQL is not only used to overcome some limitations of the SQL, but it could also be an alternative for NoSQL in certain applications where the need for analytics and decision making have to be found on-request with high consistency. [9]

4.2. Architecture of NewSQL :

An ideal DBMS should scale elastically vertically as well as horizontally. It should allow new machines to be introduced easily in a system that's already up and running . [9]
This wasn't possible to be performed efficiently with SQL. So NewSQL uses technology that's used in cloud computing and distributed applications. It implements distributed database technology. The databases are generally distributed. They follow the three tier architecture having three layers: an administrative tier, a transactional tier and a storage tier SQL provided vertical scaling but there was no provision for horizontal scaling. The NewSQL model provides horizontal scaling along with vertical scaling. The databases used are distributed while still providing ACID properties. They work efficiently in heavy load and are very robust. NewSQL also has a provision for distributed services which work with high efficiency. [10]

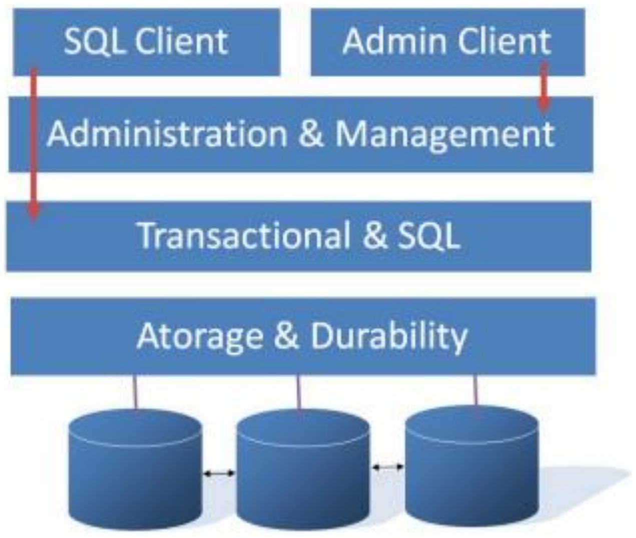


Figure 5: Architecture of a popular NewSQL database NueDB.

4.3. Properties of NewSQL :

This section describes on the characteristics of NewSQL databases. Technical characteristics of NewSQL solutions are : [10]

- SQL as the primary mechanism for application interaction.
- ACID support for transactions.
- An architecture providing much higher per-node performance than available from traditional RDBMS solutions.
- Distributed architecture
- A scale-out, shared-nothing architecture, capable of running on a large number of nodes without suffering bottlenecks.
- In-memory relational database.
- Shared-nothing architectures.

4.4. limitations of NewSQL :

- It’s not appropriate in applications with higher volume than few terabytes.
- It does not support full access to the traditional SQL tools. [9]

5. CONCLUSION :

In this first chapter, we've presented the different data models which have been designed and used to fulfill the needs of data storage and management since the first eras of computing, beginning with flat documents to NewSQL.

In the next chapter, we will focus on comparing the different data models and analyze the available solutions .

Chapter II : Theoretical Comparison

1. INTRODUCTION :

The success of SQL, NoSQL, and NewSQL databases is a reflection of their ability to provide significant functionality and performance benefits for specific domains. After studying certain works in this field such as: [11], [12], [13], [14] and [15] we have summarized the results obtained in this chapter, which presents a theoretical comparison between SQL vs. NoSQL vs NewSQL.

2. Acid vs Base :

nosql still in his early stage .it still have a long way to go through to become richly functional and stable system , because he still in his infant stage there are less advanced expertise in the field it does provide BASE properties ,but it is not reliable as ACID properties wich provided by SQL databases

ACID	BASE
ACID (Atomicity, Consistency, Isolated and Durability)	BASE (Available, Stable state, Eventually consistent)
Focus is on Consistency and Availability	Focus is on Availability and Partition tolerance
Strong Consistency	Weak consistency
pessimistic approach	optimistic approach
Complex mechanisms	Simple and fast
mostly used where data reliability and consistency is very important	mostly used where data availability and speed is important

Tableau 2: ACID VS BASE

3. SQL vs NOSQL vs NewSQL :

- Relational Property : Values are atomic. All of the values in a column have the same data type. Each row is unique. The sequence of columns is insignificant. The sequence of rows is insignificant.
- ACID (atomicity, consistency, isolation, durability) is a set of properties of database transactions intended to guarantee data validity despite errors, power failures, and other mishaps.
- SQL stands for Structured Query Language. It's used for relational databases

- OLTP (Online Transactional Processing) is a type of data processing that executes transaction-focused tasks. It involves inserting, deleting, or updating small quantities of database data
- Scaling in DBMS is the ability to expand the capacity of a database system in order to support larger amounts or requests and/or store more data without sacrificing performance
- A query can either be a request for data results from your database or for action on the data, or for both

The table outlines the difference between SQL, NOSQL and NEWSQL Features

	SQL	**NOSQL**	**NEWSQL**
Data Store Model	Two-dimensional tables	Depends on the database type: Key-value Document databases Graph Column-family databases	Combines relational model of SQL databases with the versatile scalability and speed of NoSQL databases.
Structure	schema-fixed	schema-free	NewSQL is schema-fixed as well as a schema-free database
Database transactions	ACID (Atomicity, Consistency, Isolation, and Durability) is a standard for SQL	generally follow the BASE (Basically Available, Soft State, Eventual Consistency) model	It promotes ACID properties.
Main features	Cross-platform support, multi-level security	Scalability, flexibility, high performance	Scalability, low latency, high read/write performance.
Relational Property	Yes	No	Yes
ACID	Yes	No, rather provides for CAP or BASE support	Yes,
SQL	Support for SQL	No support for old SQL	Yes, proper support for Old SQL
OLTP	Inefficient for OLTP databases.	It supports such databases, but it is not the best suited.	supports OLTP databases and is highly efficient
Scaling	Vertical scaling	Only Vertical scaling	Vertical + Horizontal scaling
Query Handling	Can handle simple queries with ease and fails when they get complex in nature	Better than SQL for processing complex queries	Highly efficient in processing complex queries and smaller queries.
Distributed Databases	No	Yes	Yes

Tableau 3: SQL ,NoSQL and NewSQL comparison

4. Selected SQL systems :

4.1. Mysql :

MySQL, is a relational database management system. It is based on structure query language (SQL), which is used for modifying , adding and removing information in the database.

4.2. Oracle :

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management.

4.3. Main features Between theme :

Mysql	Oracle
- Relational Database System - Easy to use - High Flexibility - Memory efficiency	- Availability - Scalability and Performance - Security -Backup and Recovery

Tableau 4 : Main Features of Mysql and Oracle

5. Selected NoSql systems :

5.1. Cassandra :

Is an open source distributed database management system. It is an Apache Software Foundation top-level project [cas1] designed to handle very large amounts of data spread out across many commodity servers while providing a highly available service with no single point of failure. Cassandra is a column-oriented database, its highly available nature and highly distributed leaves no single point of failure, this means that each node of its cluster can respond to any request. It supports replications on several data centers.

5.2. Mongodb :

MongoDB is a flexible NoSQL document database platform designed to overpass the relational databases approach and the limitations of different NoSQL solutions

5.3. Main Features Between Theme :

Cassandra	**Mongodb**
- Decentralized : every node in the cluster have the same role there is no point of failure data distributes across the cluster (evry node have diffrent data) however there is no master so that every node can service any request - Scalability : Read and write throughput increases linearly as new machines are added, without stopping or interrupting applications. - Fault-tolerant : Data is automatically replicated to multiple nodes for fault tolerance. Replication is supported across multiple data centers. Failed nodes can be replaced non-stop	- Replication: When the data only depends on a single database it have a high risk of failure such as a server crash, service interruptions, or even a hardware failure. Replication allows you to avoid all of these vulnerabilities by deploying multiple servers for disaster recovery and backup. - Sharding: Sharding is the process of splitting larger datasets across multiple distributed collections that helps to deal with particularly large datasets - Load balancing : it ensures that every user has a consistent view and quality experience with the data they need to access

Tableau 5 : Main Featurs Between Cassandra and Mongodb

6. Selected NewSQL Systemes :

6.1. VoltDB

VoltDB is an in-memory, scale-out SQL database purpose- built to power a new generation of applications that thrive on fast, smart data. Tapping the lightning speed and real time analytics of VoltDB, organizations are able to add context and intelligence to data-the instant it arrives-to make real-time transactional decisions that maximize business value.

6.2. NuoDB :

Launched in 2010 by industry-renowned database architect Jim Starkey and accomplished software CEO Barry Morris, the company is based in Cambridge .NuoDB is a distributed, peer to peer system that provides an in-memory database service with ACID transactions. A NuoDB database appears to the developer and operator as a single, logical system. In practice, a NuoDB database can be running in multiple locations with hosts added and removed according to demand

6.3. Main Features Between theme :

Voltdb	**Nuodb**
- Compatibility with a wide range of product - SQL Familarity - Database replication for disaster recovery - Real_time personalizastion	- Elastic Scalability : NuoDB maintains ACID safeguards and a standards-based SQL interface while providing simple and flexible scaling. This means that you only add capacity when you need it - and give it up when you don't. - Active Active Database : NuoDB's Elastic SQL Database provides active benefits as part of the database, eliminating the need for additional software. It also supports reading and writing to multiple host available after ACID guarantees.

Tableau 6 : Main Features Between voltdb and Nuodb

7. General Comparison of Selected Databases:

The fellowing table presents a general comparison of selected databases on some technical attributes

Category	**Nosql**		**Newsql**		**Sql**	
Features	**Mongodb**	**Casandra**	**VoltDB**	**NuoDB**	**MySQL**	**oracle**
Implementation	C++	JAVA	JAVA	C++	C and C++	C and C++
Data Storage	Memory Mapped files	hard disk	In-memory Dic_based storage	In-memory Dic based storage	hard disk.	hard disk
Transaction concepts	Atomic operation With a single Document posible	Atomicity& isolation For single operation	ACID	ACID	ACID complaint focus on integrity.	ACID
Supported programing languages	c,c++,java ,erlang…	C++,erlang ,java,php…	C++,erlang ,java,php…	Net,c,c++ Java ,php..	C ,C++	Delphi,c,c++, Erlang
Map Reduce	Yes	Yes	No	No	Yes	No
Replication	Master-Slave Replication	Selectable Replication factor	Master-Slave Replication	Yes	Master-Slave and Master-Master approach	Multi-source replication Source-replica replication
Partitioning	Sharding	Sharding with no single point failur	Sharding	Data is Dynamically Stored/cached on the nodes	Consistent hashing with user defined where the stored procedure e runs.	Sharding Horizontal partition
License type	GPL9	Apache 2.0	GNU Affero General Public	commercial	GPL+FLOSS / proprietary	commercial

Tableau 7: Comparison of Selected Databases on Some Technical Attributes.

8. Conclustion :

SQL provides vertical scaling with ACID properties while NoSQL is suitable for horizontal scaling providing BASE. However NoSQL does not provide ACID properties which are necessary for a reliable database. The need of modern enterprises where data is growing day by day and all they work with is Big Data especially even while working in OLTP system, NewSQL is the best choice. NewSQL is improvement of SQL providing horizontal scaling while maintaining ACID properties. This not only allows working with Big Data by providing the ability to work concurrently, it also maintains ACID properties. NewSQL has found the sweet spot between consistency, scalability, speed and availability. While still being in its infant stage, NewSQL ticks all the right boxes to make it an ideal database for Big Data OLTP applications..

All the deployments applied to the different SQL, NoSQL AND Newsql solutions presented previously, all the experimental results will be presented in the next chapter.

Chapter III: Practical comparison

1. Introduction:

The fast development of the internet and cloud computing has encouraged the availability of databases to be able to effectively store and process extensive data and demand high performance when reading and writing.

In this study , trying to measure the response times on the query insert Delete Update using mysql,oracle in sql ,mongodb,cassandra in Nosql and voltdb,nuodb in Newsql

2. Development environment:

- Processor : Intel® Core(TM) i3-7020U CPU @ 2.30 GHz
- RAM : 12.0 Go
- System : Microsoft Windows10 Professional 64 bits
- Hard Disk: 1000 GB

The comparative study carried out used the different tools and systems SQL, NoSQL and NEWSQL, summarized in the following table:

System	**Version**
Operation System	Windows10
Mysql(exampp)	Version 3.3.0
Oracle	Version 21c express edition
Mongodb	Version 5.0.6
Cassandra	Version 3.11.12
Voltdb	/
Nuodb	/

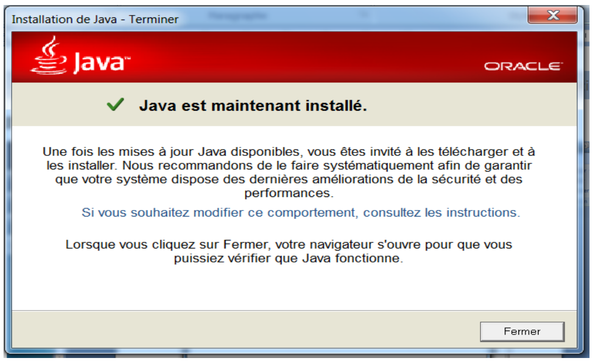
Tableau 8 : Versions of systems and tools used

3. Configuration and Installation of DB SQL for Windows 10 :

3.1. Configuration and creation of environment variables :

3.1.1. User variables :

- download java 8 or jdk 1.8.0_251 and install it


Installation de Java - Terminer

Java est maintenant installé.

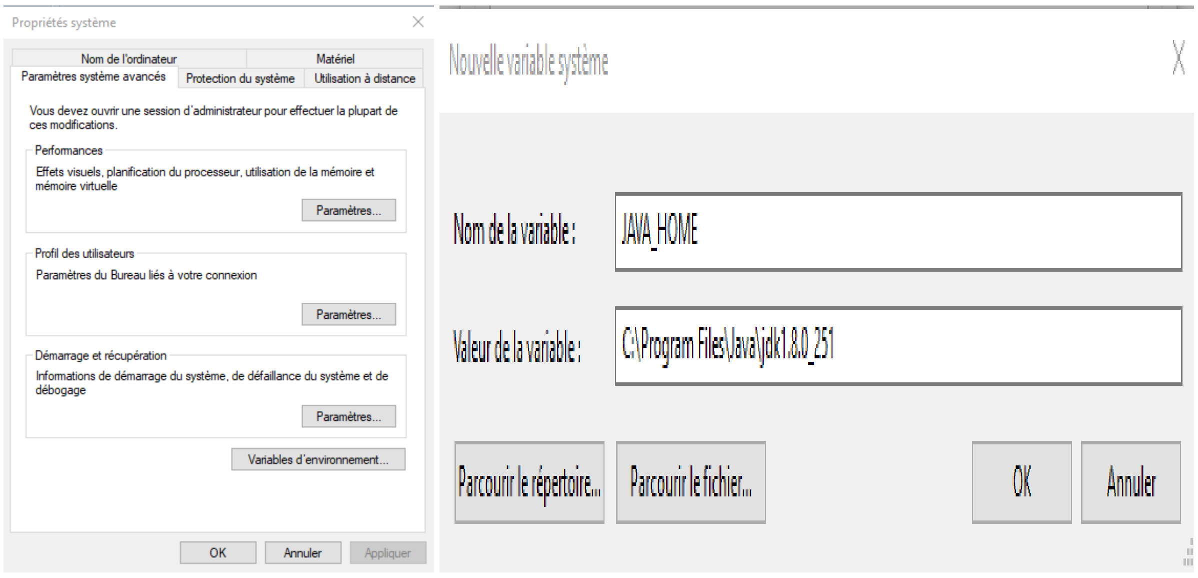
Une fois les mises à jour Java disponibles, vous êtes invité à les télécharger et à les installer. Nous recommandons de le faire systématiquement afin de garantir que votre système dispose des dernières améliorations de la sécurité et des performances.

Si vous souhaitez modifier ce comportement, consultez les instructions.

Lorsque vous cliquez sur Fermer, votre navigateur s'ouvre pour que vous puissiez vérifier que Java fonctionne.

Fermer

- Create the variable: JAVA_HOME


Propriétés système

Nom de l'ordinateur | Matériel | Paramètres système avancés | Protection du système | Utilisation à distance

Vous devez ouvrir une session d'administrateur pour effectuer la plupart de ces modifications.

Performances

Effets visuels, planification du processeur, utilisation de la mémoire et mémoire virtuelle

Paramètres...

Profil des utilisateurs

Paramètres du Bureau liés à votre connexion

Paramètres...

Démarrage et récupération

Informations de démarrage du système, de défaillance du système et de débogage

Paramètres...

Variables d'environnement...

OK | Annuler | Appliquer

Nouvelle variable système

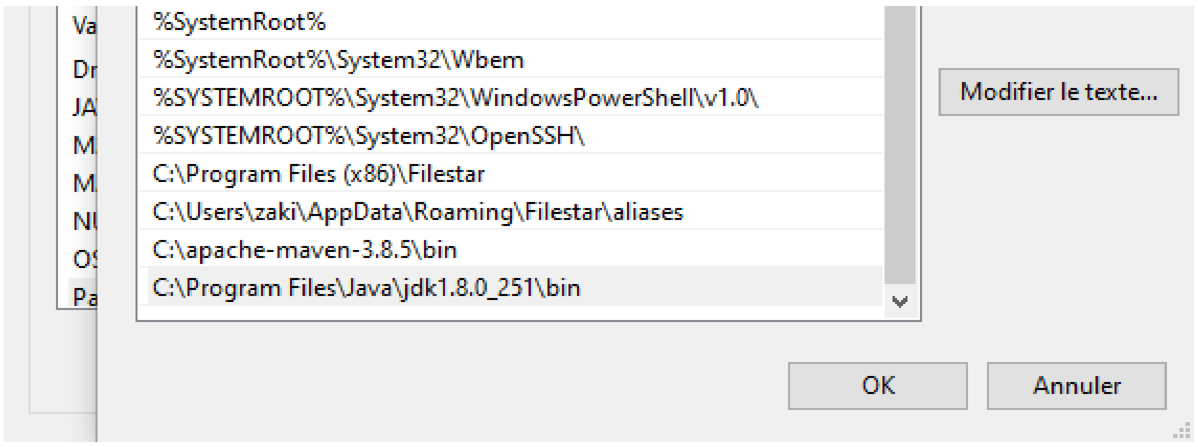
Nom de la variable : JAVA_HOME

Valeur de la variable : C:\Program Files\Java\jdk1.8.0_251

Parcourir le répertoire... | Parcourir le fichier... | OK | Annuler

3.1.2. System variables:

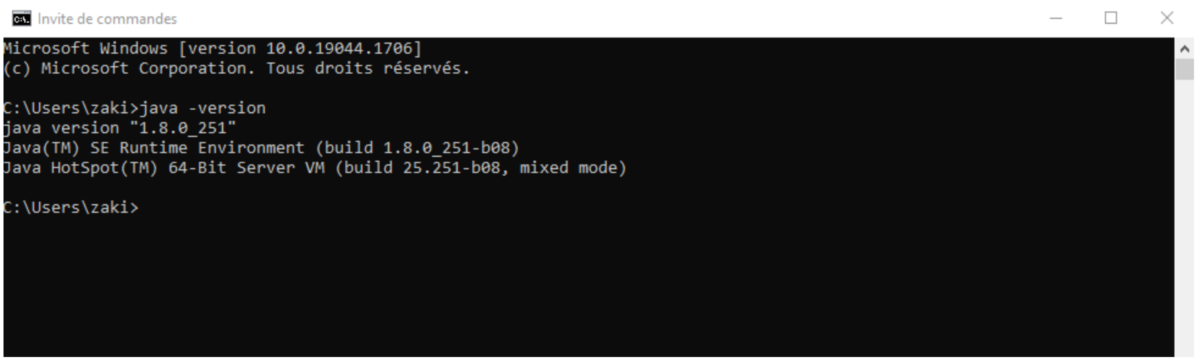
- **Path Java :**


%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files (x86)\Filestar
C:\Users\zaki\AppData\Roaming\Filestar\aliases
C:\apache-maven-3.8.5\bin
C:\Program Files\Java\jdk1.8.0_251\bin

Modifier le texte...

OK | Annuler

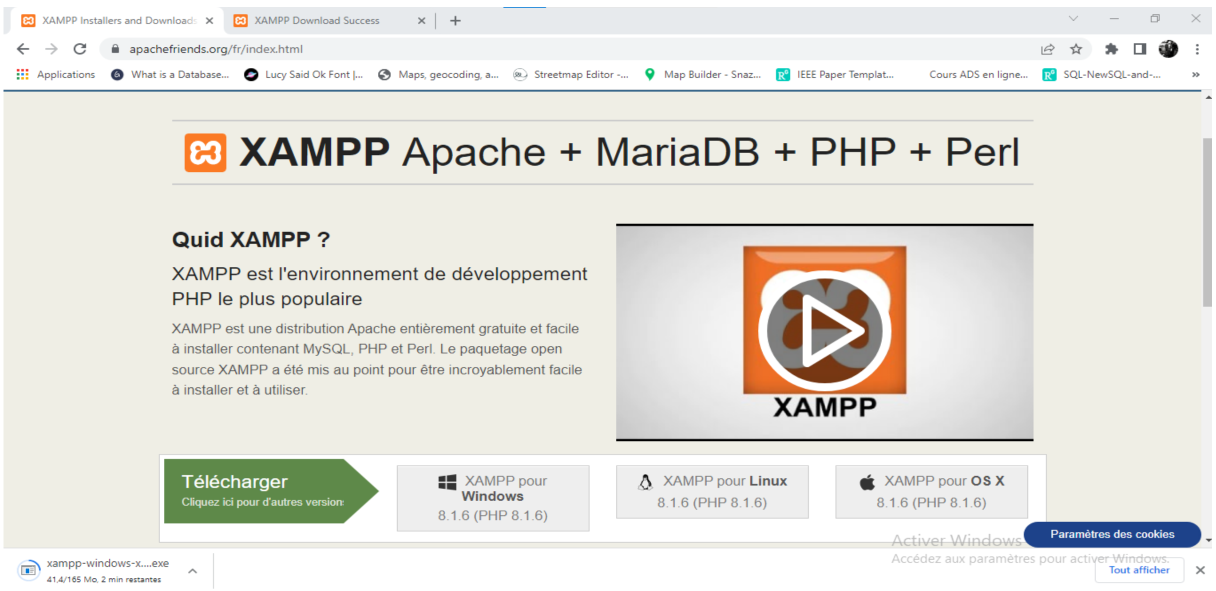
- **Test java version :**



```
Microsoft Windows [version 10.0.19044.1706]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\zaki>java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)

C:\Users\zaki>
```

3.2. Installation of mysql (exampp) :

- Download and install xampp version 3.3.0 [16]

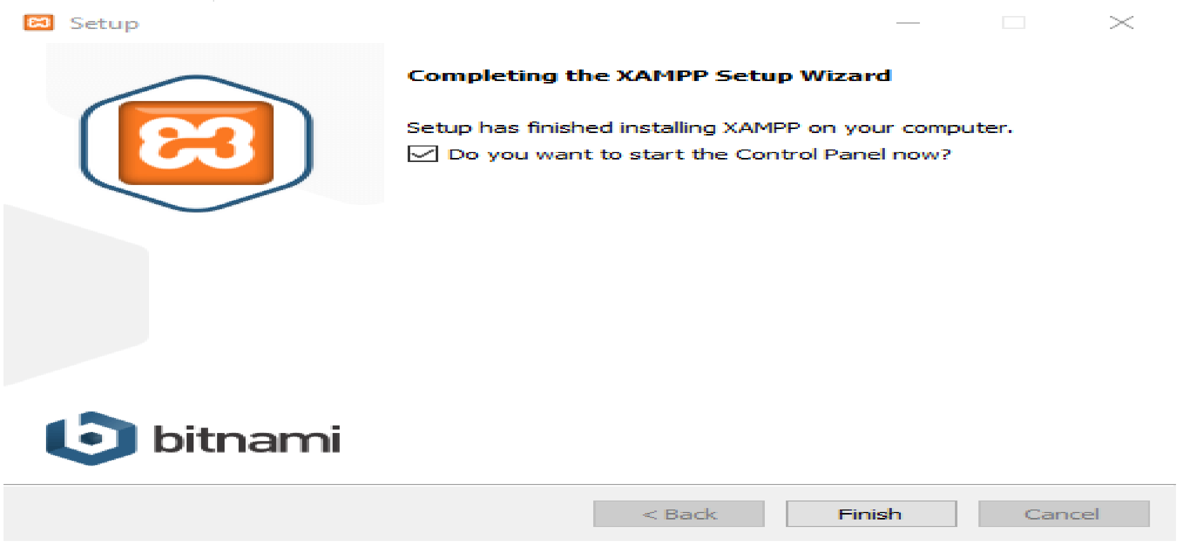


XAMPP Apache + MariaDB + PHP + Perl

Quid XAMPP ?

XAMPP est l'environnement de développement PHP le plus populaire

XAMPP est une distribution Apache entièrement gratuite et facile à installer contenant MySQL, PHP et Perl. Le paquetage open source XAMPP a été mis au point pour être incroyablement facile à installer et à utiliser.

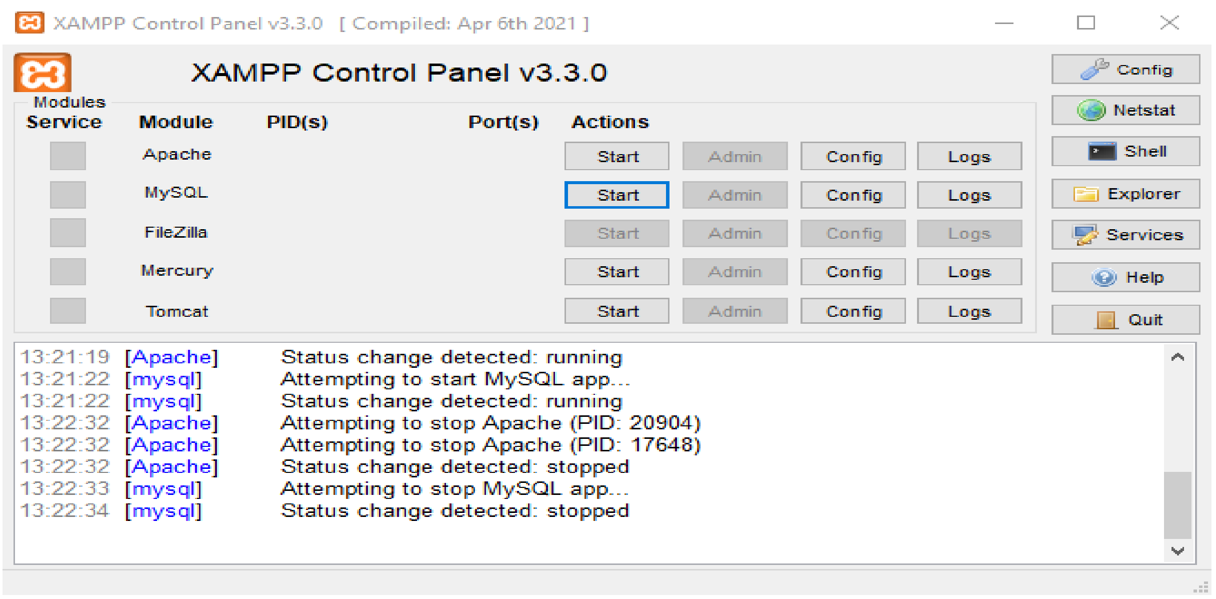


Completing the XAMPP Setup Wizard

Setup has finished installing XAMPP on your computer.

☑ Do you want to start the Control Panel now?

- Run xampp by clicking xampp control panel

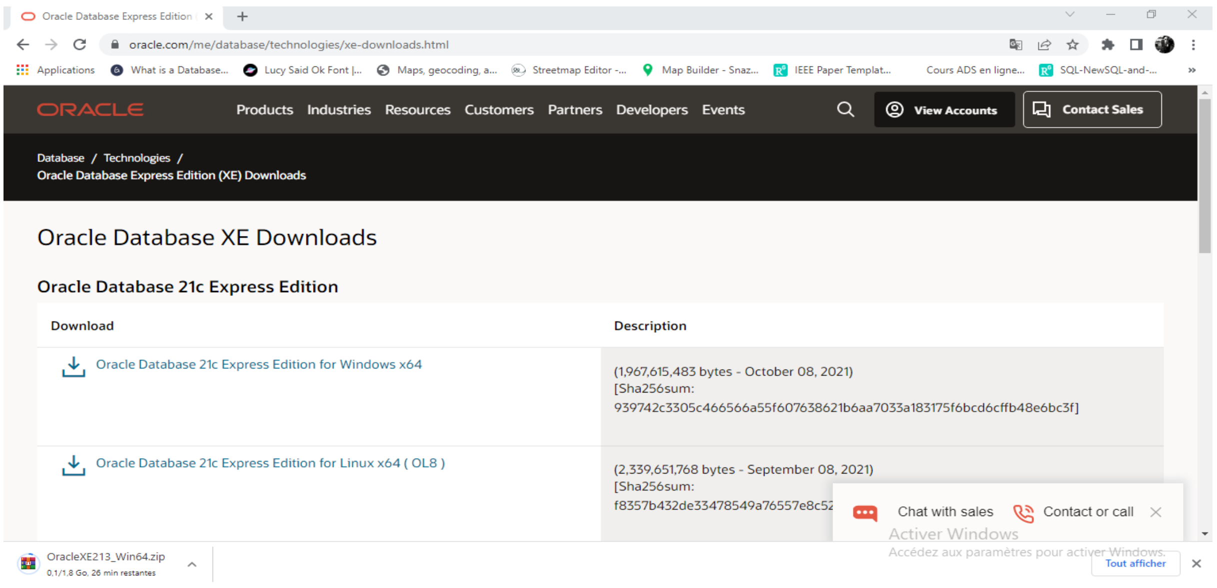

XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]
XAMPP Control Panel v3.3.0
Config
Netstat
Shell
Explorer
Services
Help
Quit

Modules Service	Module	PID(s)	Port(s)	Actions			
	Apache			Start	Admin	Config	Logs
	MySQL			Start	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
13:21:19  [Apache]   Status change detected: running
13:21:22  [mysql]    Attempting to start MySQL app...
13:21:22  [mysql]    Status change detected: running
13:22:32  [Apache]   Attempting to stop Apache (PID: 20904)
13:22:32  [Apache]   Attempting to stop Apache (PID: 17648)
13:22:32  [Apache]   Status change detected: stopped
13:22:33  [mysql]    Attempting to stop MySQL app...
13:22:34  [mysql]    Status change detected: stopped
```

3.3. Installation of Oracle :

- Download and install Oracle version 21c express Edition [17]

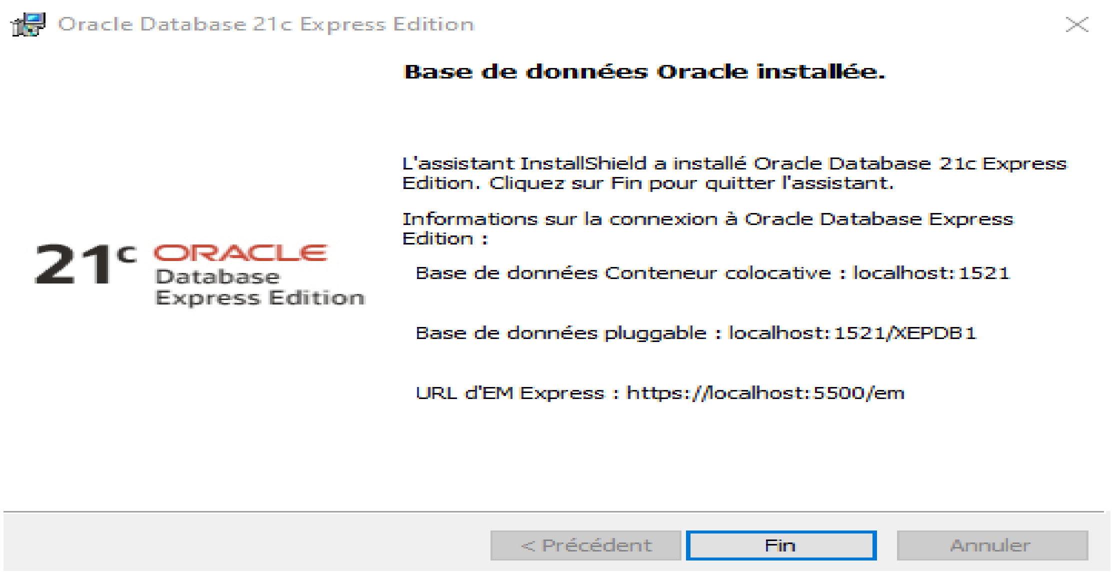

Oracle Database Express Edition
oracle.com/me/database/technologies/xe-downloads.html
ORACLE
Products Industries Resources Customers Partners Developers Events
View Accounts
Contact Sales
Chat with sales
Contact or call
OracleXE213_Win64.zip

Database / Technologies /
Oracle Database Express Edition (XE) Downloads

Oracle Database XE Downloads

Oracle Database 21c Express Edition

Download	Description
Oracle Database 21c Express Edition for Windows x64	(1,967,615,483 bytes - October 08, 2021) [Sha256sum: 939742c3305c466566a55f607638621b6aa7033a183175f6bcd6cffb48e6bc3f]
Oracle Database 21c Express Edition for Linux x64 (OL8)	(2,339,651,768 bytes - September 08, 2021) [Sha256sum: f8357b432de33478549a76557e8c52


Oracle Database 21c Express Edition
21c ORACLE Database Express Edition
< Précédent
Fin
Annuler

Base de données Oracle installée.

L'assistant InstallShield a installé Oracle Database 21c Express Edition. Cliquez sur Fin pour quitter l'assistant.

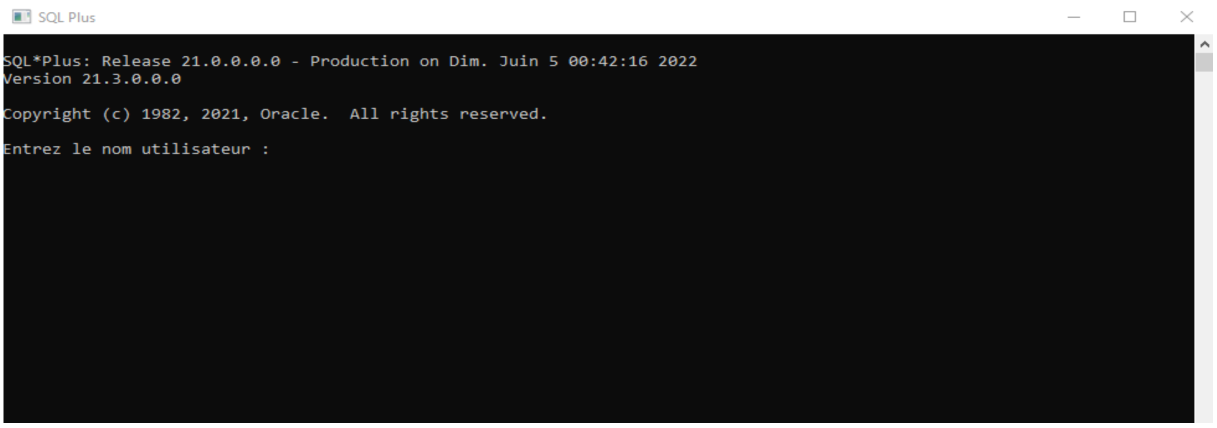
Informations sur la connexion à Oracle Database Express Edition :

Base de données Conteneur colocative : localhost:1521

Base de données pluggable : localhost:1521/XEPDB1

URL d'EM Express : https://localhost:5500/em

- Click on sql plus to start oracle

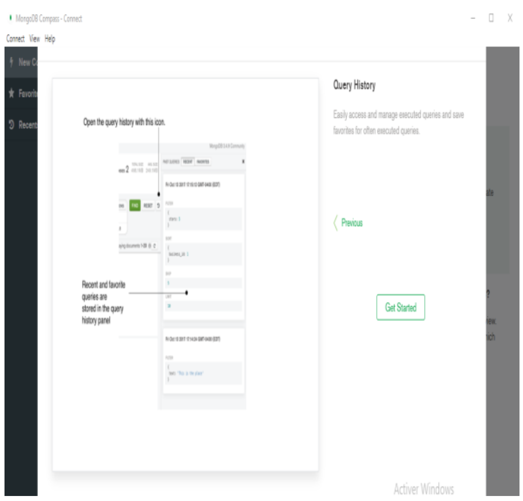


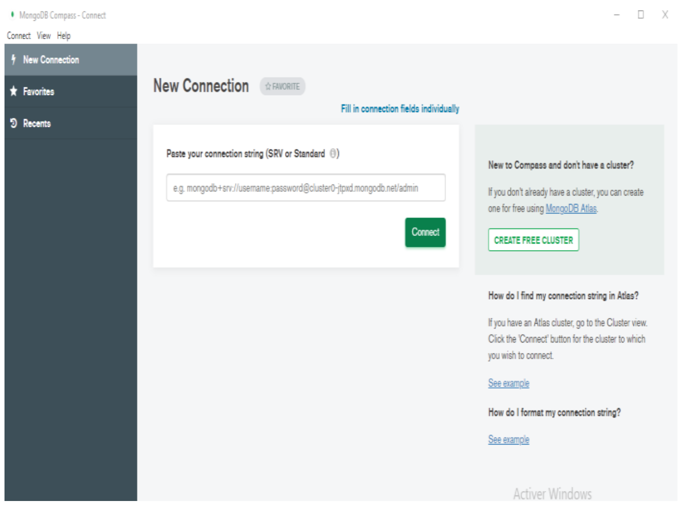
```
SQL*Plus: Release 21.0.0.0.0 - Production on Dim. Juin 5 00:42:16 2022
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle.  All rights reserved.

Entrez le nom utilisateur :
```

4. Configuration and Installation of DB Nosql for Windows 10 :

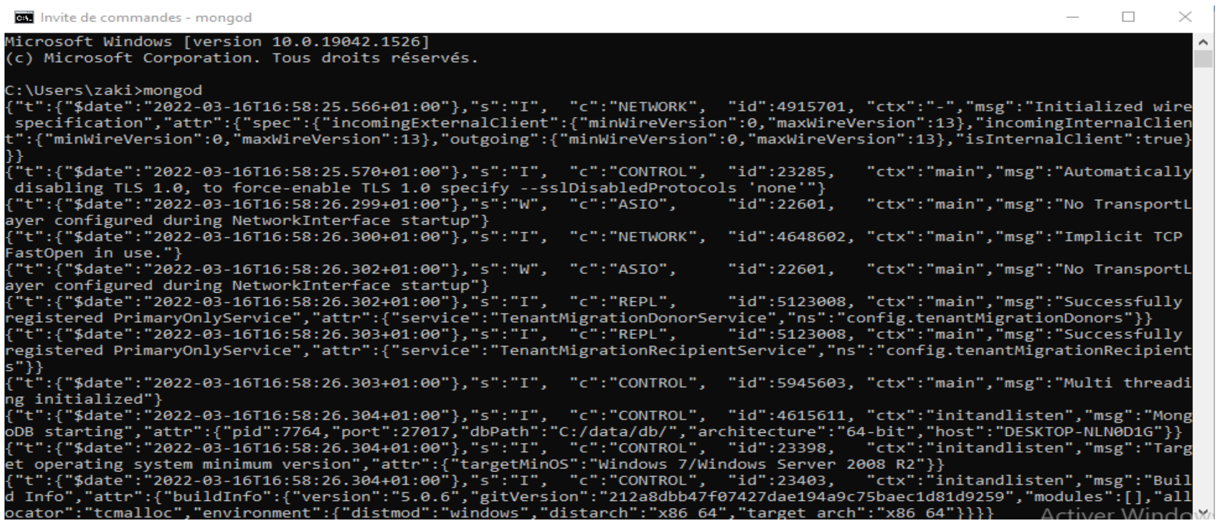
4.1. Installation Mongodb version 5.0.6 msi :

- Download Mongodb 5.0.6 msi zip file. [18]
- Extract the zip file to the prepared directory (disk c)
- Run mongodb.msi and install mongodb
- Click on get started


MongoDB Compass - Connect
Connect View Help
New C
Favorit
Recent
Open the query history with this icon.
Recent and favorite queries are stored in the query history panel
Query History
Easily access and manage executed queries and save favorites for often executed queries.
Previous
Get Started
Activer Windows

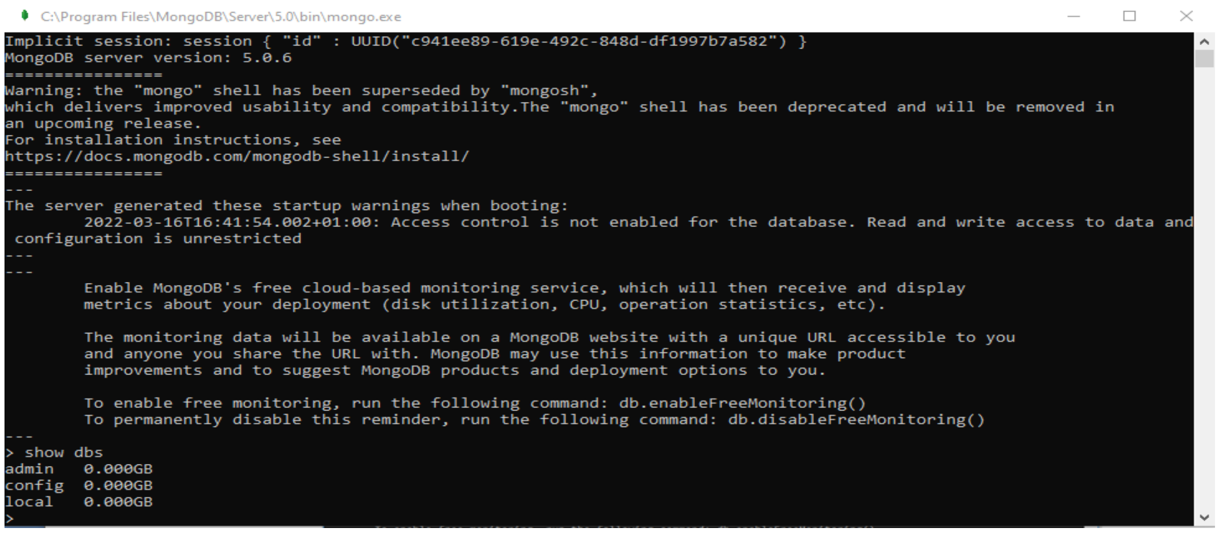

MongoDB Compass - Connect
Connect View Help
New Connection
Favorites
Recents
New Connection
FAVORITE
Fill in connection fields individually
Paste your connection string (SRV or Standard
e.g. mongodb+srv://username:password@cluster0-jtpxd.mongodb.net/admin
Connect
New to Compass and don't have a cluster?
If you don't already have a cluster, you can create one for free using MongoDB Atlas.
CREATE FREE CLUSTER
How do I find my connection string in Atlas?
If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect.
See example
How do I format my connection string?
See example
Activer Windows

- Create a rep data and then rep db: C:\data\db
- Server connection (mongod)



```
Invite de commandes - mongod
Microsoft Windows [version 10.0.19042.1526]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\zaki>mongod
{"t":{"$date":"2022-03-16T16:58:25.566+01:00"},"s":"I",  "c":"NETWORK",  "id":4915701, "ctx":"-","msg":"Initialized wire
 specification","attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":13},"incomingInternalClien
t":{"minWireVersion":0,"maxWireVersion":13},"outgoing":{"minWireVersion":0,"maxWireVersion":13},"isInternalClient":true}
}}
{"t":{"$date":"2022-03-16T16:58:25.570+01:00"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"main","msg":"Automatically
 disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2022-03-16T16:58:26.299+01:00"},"s":"W",  "c":"ASIO",     "id":22601,   "ctx":"main","msg":"No TransportL
ayer configured during NetworkInterface startup"}
{"t":{"$date":"2022-03-16T16:58:26.300+01:00"},"s":"I",  "c":"NETWORK",  "id":4648602, "ctx":"main","msg":"Implicit TCP
FastOpen in use."}
{"t":{"$date":"2022-03-16T16:58:26.302+01:00"},"s":"W",  "c":"ASIO",     "id":22601,   "ctx":"main","msg":"No TransportL
ayer configured during NetworkInterface startup"}
{"t":{"$date":"2022-03-16T16:58:26.302+01:00"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"main","msg":"Successfully
registered PrimaryOnlyService","attr":{"service":"TenantMigrationDonorService","ns":"config.tenantMigrationDonors"}}
{"t":{"$date":"2022-03-16T16:58:26.303+01:00"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"main","msg":"Successfully
registered PrimaryOnlyService","attr":{"service":"TenantMigrationRecipientService","ns":"config.tenantMigrationRecipient
s"}}
{"t":{"$date":"2022-03-16T16:58:26.303+01:00"},"s":"I",  "c":"CONTROL",  "id":5945603, "ctx":"main","msg":"Multi threadi
ng initialized"}
{"t":{"$date":"2022-03-16T16:58:26.304+01:00"},"s":"I",  "c":"CONTROL",  "id":4615611, "ctx":"initandlisten","msg":"Mong
oDB starting","attr":{"pid":7764,"port":27017,"dbPath":"C:/data/db/","architecture":"64-bit","host":"DESKTOP-NLN0D1G"}}
{"t":{"$date":"2022-03-16T16:58:26.304+01:00"},"s":"I",  "c":"CONTROL",  "id":23398,   "ctx":"initandlisten","msg":"Targ
et operating system minimum version","attr":{"targetMinOS":"Windows 7/Windows Server 2008 R2"}}
{"t":{"$date":"2022-03-16T16:58:26.304+01:00"},"s":"I",  "c":"CONTROL",  "id":23403,   "ctx":"initandlisten","msg":"Buil
d Info","attr":{"buildInfo":{"version":"5.0.6","gitVersion":"212a8dbb47f07427dae194a9c75baec1d81d9259","modules":[],"all
ocator":"tcmalloc","environment":{"distmod":"windows","distarch":"x86_64","target_arch":"x86_64"}}}}
```

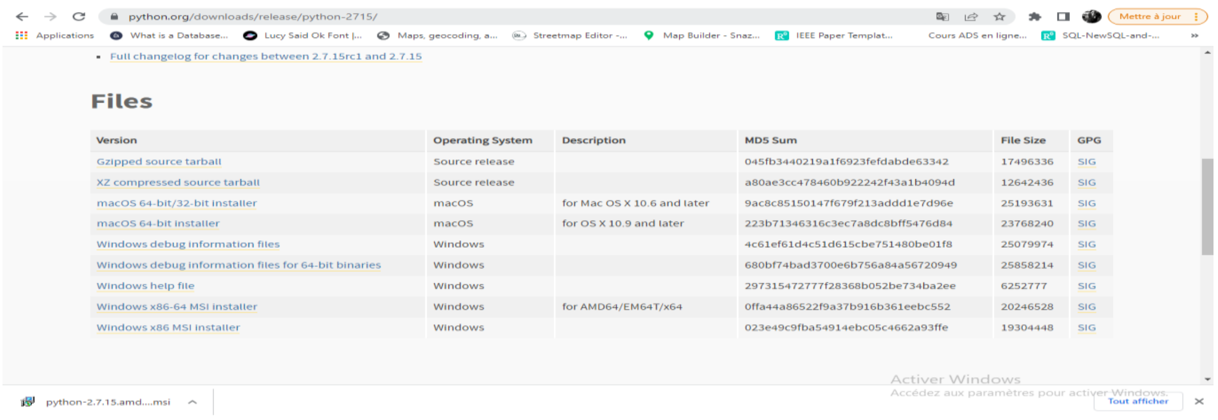
- Shell connection (mongo)



```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
Implicit session: session { "id" : UUID("c941ee89-619e-492c-848d-df1997b7a582") }
MongoDB server version: 5.0.6
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
---
The server generated these startup warnings when booting:
        2022-03-16T16:41:54.002+01:00: Access control is not enabled for the database. Read and write access to data and
 configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
>
```

4.2. Installation Cassandra Version 3.11.12 :

- Install python v2.7.15 [19]



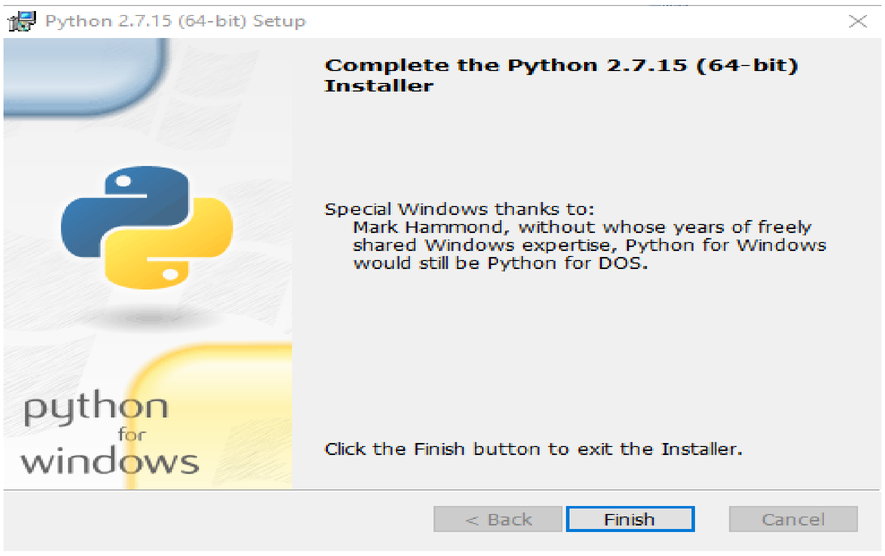
python.org/downloads/release/python-2715/

- Full changelog for changes between 2.7.15rc1 and 2.7.15

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		045fb3440219a1f6923fefdabde63342	17496336	SIG
XZ compressed source tarball	Source release		a80ae3cc478460b922242f43a1b4094d	12642436	SIG
macOS 64-bit/32-bit installer	macOS	for Mac OS X 10.6 and later	9ac8c85150147f679f213addd1e7d96e	25193631	SIG
macOS 64-bit installer	macOS	for OS X 10.9 and later	223b71346316c3ec7a8dc8bff5476d84	23768240	SIG
Windows debug information files	Windows		4c61ef61d4c51d615cbe751480be01f8	25079974	SIG
Windows debug information files for 64-bit binaries	Windows		680bf74bad3700e6b756a84a56720949	25858214	SIG
Windows help file	Windows		297315472777f28368b052be734ba2ee	6252777	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64	0ffa44a86522f9a37b916b361eebc552	20246528	SIG
Windows x86 MSI installer	Windows		023e49c9fba54914ebc05c4662a93ffe	19304448	SIG

python-2.7.15.amd....msi



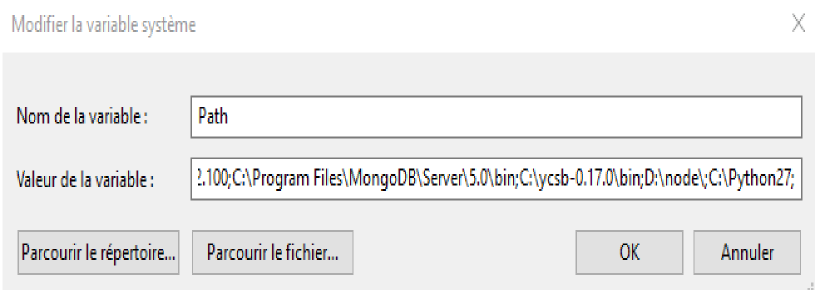
Python 2.7.15 (64-bit) Setup

Complete the Python 2.7.15 (64-bit) Installer

Special Windows thanks to:
Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

Click the Finish button to exit the Installer.

- Add python to user and system path:



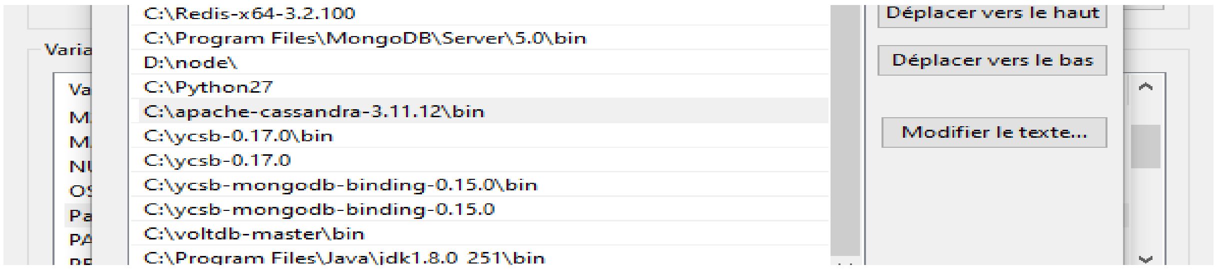
Modifier la variable système

Nom de la variable : Path

Valeur de la variable : 2.100;C:\Program Files\MongoDB\Server\5.0\bin;C:\ycsb-0.17.0\bin;D:\node\;C:\Python27;

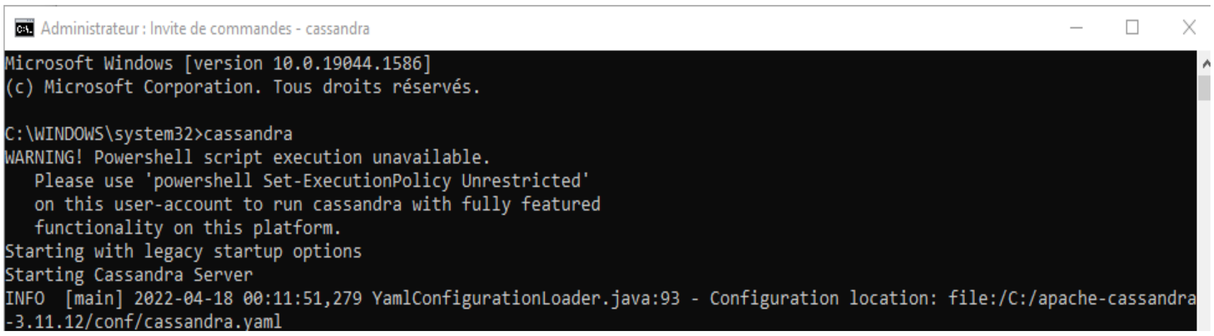
A. Install Cassandra v 3.11.12.tar.gz

- **Download and install Cassandra 3.11.12 [20]**
- **Add Cassandra to user variable path**



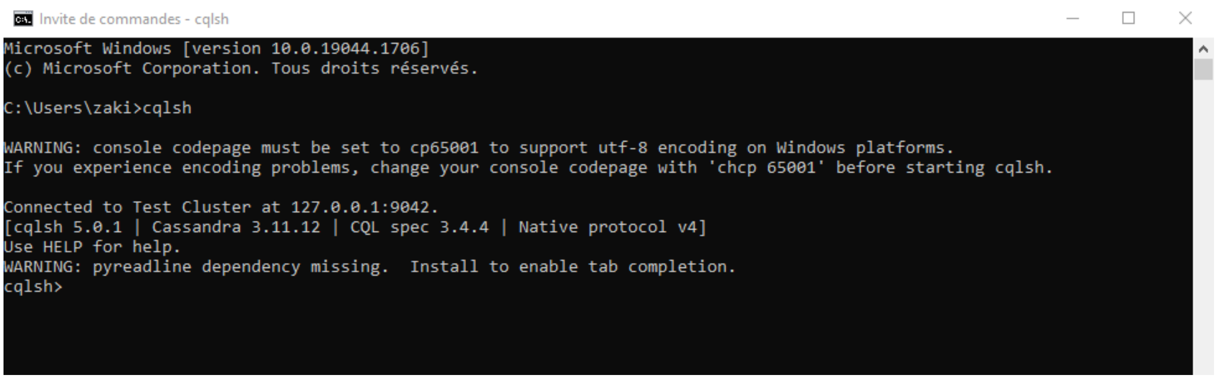
```
C:\Redis-x64-3.2.100
C:\Program Files\MongoDB\Server\5.0\bin
D:\node\
C:\Python27
C:\apache-cassandra-3.11.12\bin
C:\ycsb-0.17.0\bin
C:\ycsb-0.17.0
C:\ycsb-mongodb-binding-0.15.0\bin
C:\ycsb-mongodb-binding-0.15.0
C:\voltdb-master\bin
C:\Program Files\Java\jdk1.8.0_251\bin
```

- **Type Cassandra in cmd to launch the server**



```
Microsoft Windows [version 10.0.19044.1586]
(c) Microsoft Corporation. Tous droits réservés.

C:\WINDOWS\system32>cassandra
WARNING! Powershell script execution unavailable.
   Please use 'powershell Set-ExecutionPolicy Unrestricted'
   on this user-account to run cassandra with fully featured
   functionality on this platform.
Starting with legacy startup options
Starting Cassandra Server
INFO  [main] 2022-04-18 00:11:51,279 YamlConfigurationLoader.java:93 - Configuration location: file:/C:/apache-cassandra
-3.11.12/conf/cassandra.yaml
```

- **Run cqlsh Cassandra (cql_Schell)**



```
Microsoft Windows [version 10.0.19044.1706]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\zaki>cqlsh

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.12 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh>
```

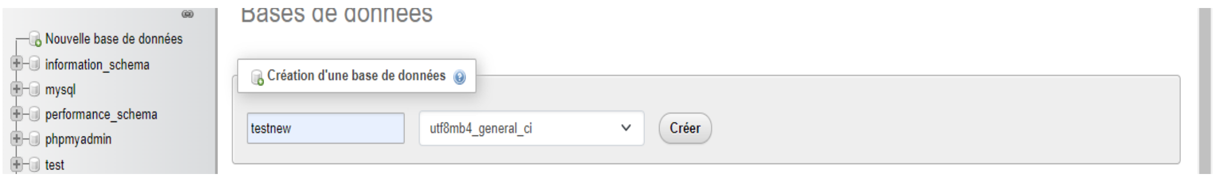
5. EXPERIMENTAL RESULT :

We will start by making a query for each data base. This stage is done to ensure the query syntax is suitable for each SQL, Nosql, NewSql databases that will be used.

1. Insert users :

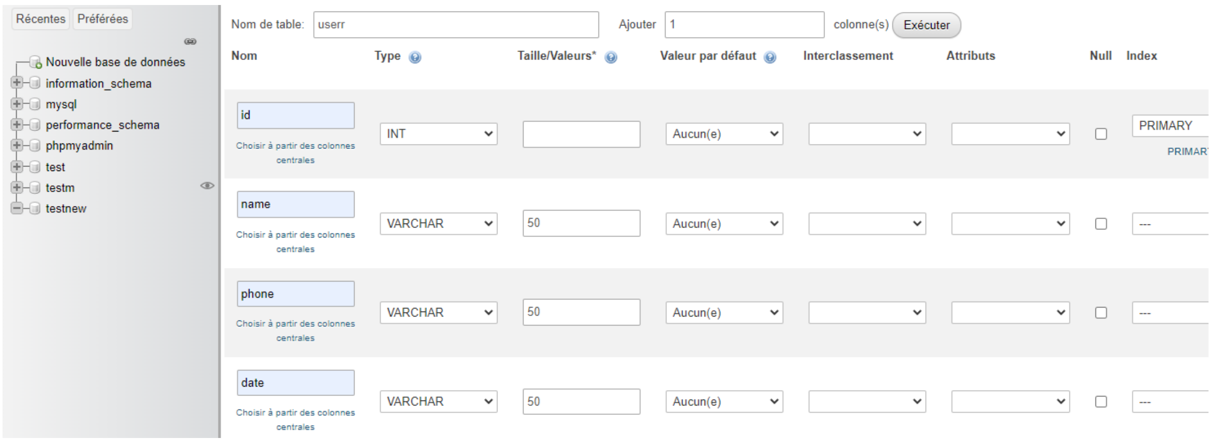
In the INSERT operation, we generated objects to be inserted in the database. Each object has an ‘id’, a ‘name’ a ‘phone’ and a ‘date’. And we kept the same structure for all the databases. We took a set of 1 inserts and we computed how much it took to insert the rows in the database. After that we took 10 and 100 until we got to 1000

➢ Mysql(exampp) test:
  - Create database (testnew)



testnew | utf8mb4_general_ci

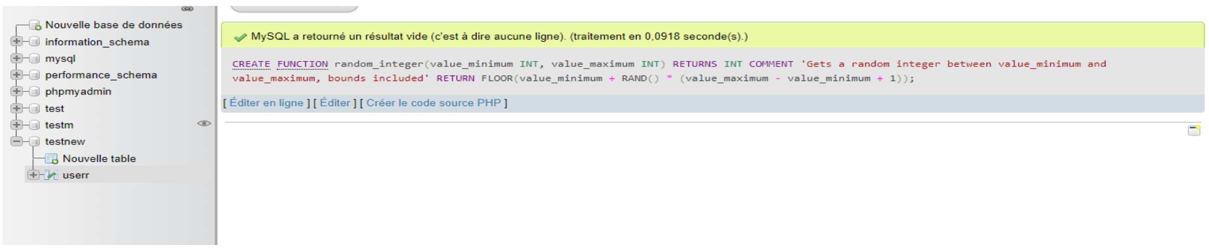
- Create table userr



Nom de table: userr   Ajouter 1 colonne(s)

Nom	Type	Taille/Valeurs*	Valeur par défaut	Interclassement	Attributs	Null	Index
id	INT		Aucun(e)			☐	PRIMARY
name	VARCHAR	50	Aucun(e)			☐	---
phone	VARCHAR	50	Aucun(e)			☐	---
date	VARCHAR	50	Aucun(e)			☐	---

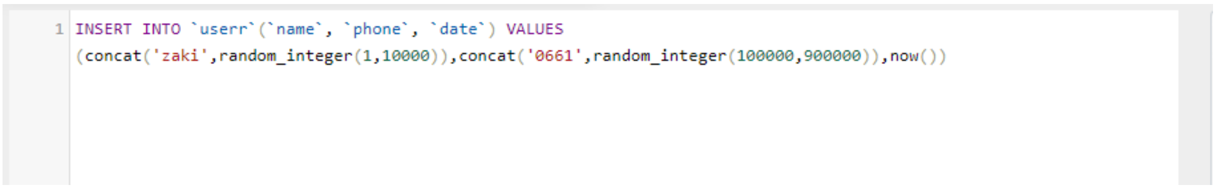
- Create function :



MySQL a retourné un résultat vide (c'est à dire aucune ligne). (traitement en 0,0918 seconde(s).)

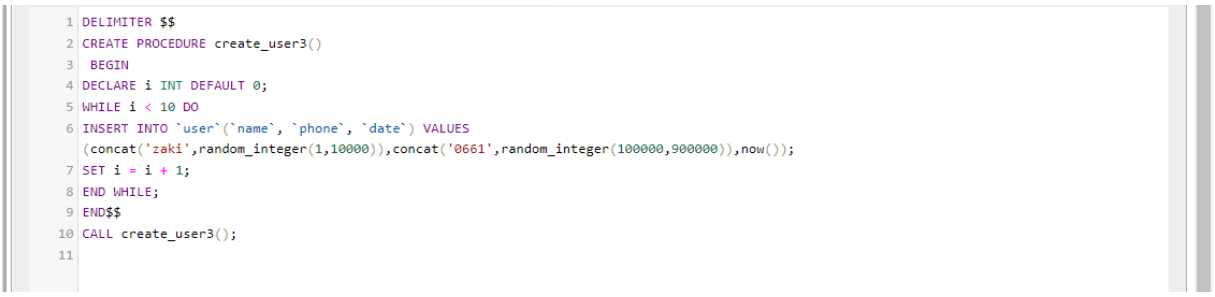
```
CREATE FUNCTION random_integer(value_minimum INT, value_maximum INT) RETURNS INT COMMENT 'Gets a random integer between value_minimum and
value_maximum, bounds included' RETURN FLOOR(value_minimum + RAND() * (value_maximum - value_minimum + 1));
```

- Generating / inserting random 1 , 10,100 and 1000 users



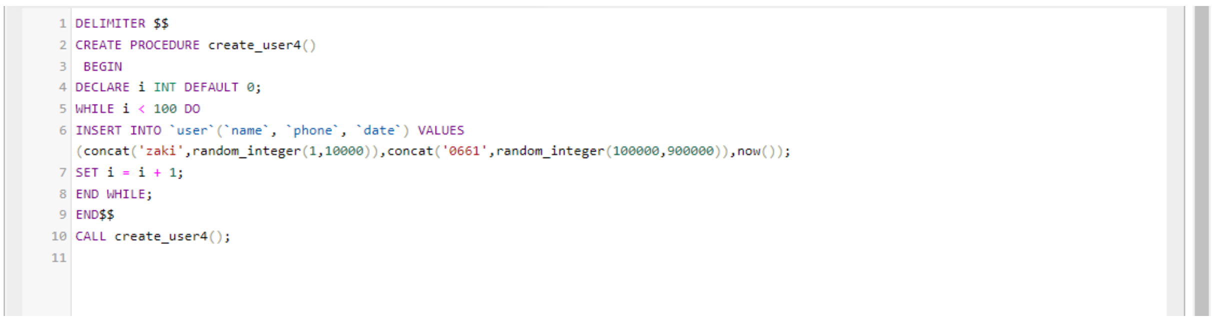
```
INSERT INTO `userr`(`name`, `phone`, `date`) VALUES
(concat('zaki',random_integer(1,10000)),concat('0661',random_integer(100000,900000)),now())
```

Figure 6 : inserting 1 user (Mysql)



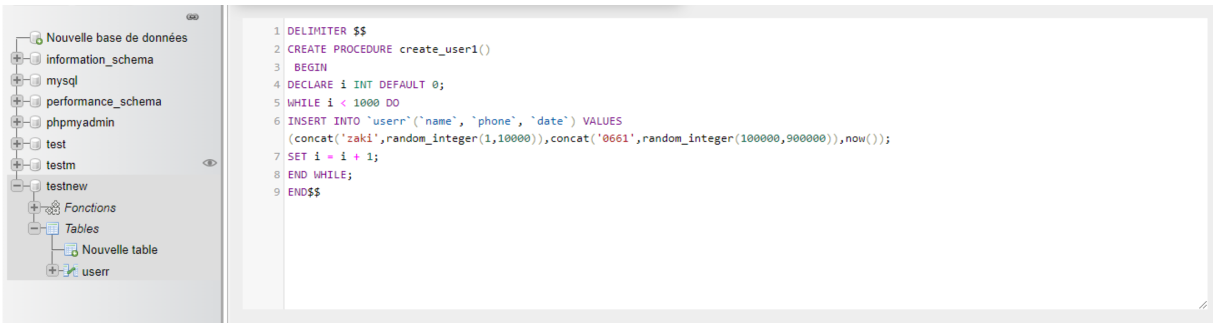
```
DELIMITER $$
CREATE PROCEDURE create_user3()
 BEGIN
DECLARE i INT DEFAULT 0;
WHILE i < 10 DO
INSERT INTO `user`(`name`, `phone`, `date`) VALUES
(concat('zaki',random_integer(1,10000)),concat('0661',random_integer(100000,900000)),now());
SET i = i + 1;
END WHILE;
END$$
CALL create_user3();
```

Figure 7 : Procedure Generate 10 user (Mysql)



```
DELIMITER $$
CREATE PROCEDURE create_user4()
 BEGIN
DECLARE i INT DEFAULT 0;
WHILE i < 100 DO
INSERT INTO `user`(`name`, `phone`, `date`) VALUES
(concat('zaki',random_integer(1,10000)),concat('0661',random_integer(100000,900000)),now());
SET i = i + 1;
END WHILE;
END$$
CALL create_user4();
```

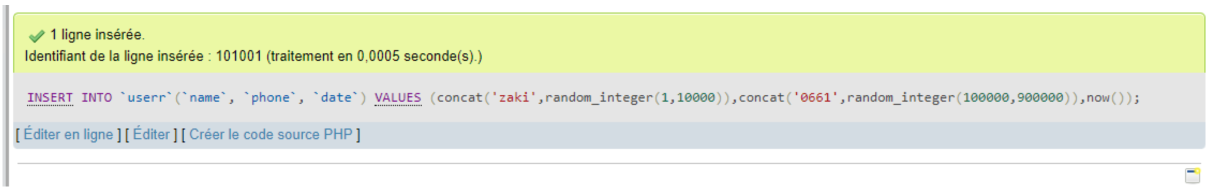
Figure 8 : Procedure Generate 100 user (Mysql)



```
DELIMITER $$
CREATE PROCEDURE create_user1()
 BEGIN
DECLARE i INT DEFAULT 0;
WHILE i < 1000 DO
INSERT INTO `userr`(`name`, `phone`, `date`) VALUES
(concat('zaki',random_integer(1,10000)),concat('0661',random_integer(100000,900000)),now());
SET i = i + 1;
END WHILE;
END$$
```

Figure 9 : Procedure Generate 1000 user (Mysql)

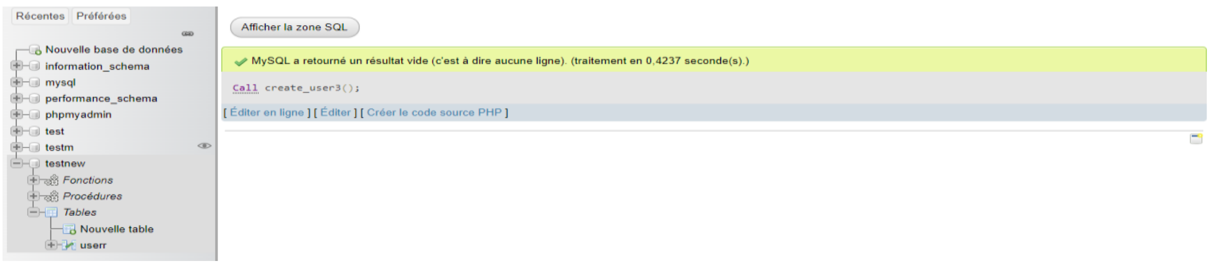
- The result of test is as follows:



```
1 ligne insérée.
Identifiant de la ligne insérée : 101001 (traitement en 0,0005 seconde(s).)

INSERT INTO `userr`(`name`, `phone`, `date`) VALUES (concat('zaki',random_integer(1,10000)),concat('0661',random_integer(100000,900000)),now());
```

[Éditer en ligne] [Éditer] [Créer le code source PHP]

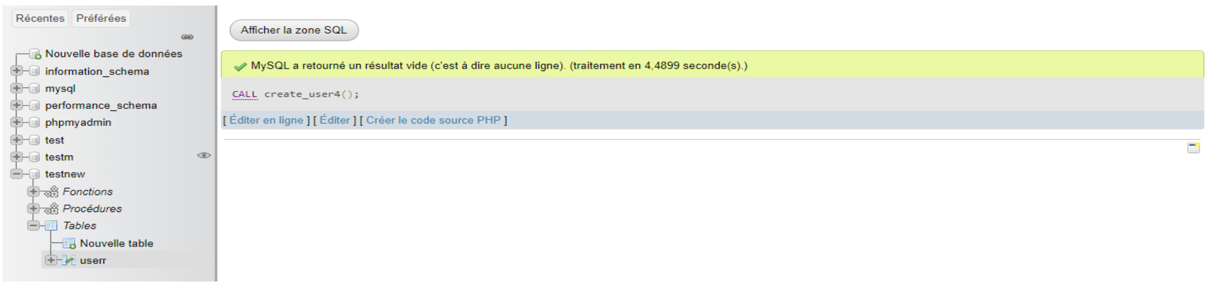
Figure 10 : execution time of inserting 1 user (Mysql)



```
MySQL a retourné un résultat vide (c'est à dire aucune ligne). (traitement en 0,4237 seconde(s).)

call create_user3();
```

[Éditer en ligne] [Éditer] [Créer le code source PHP]

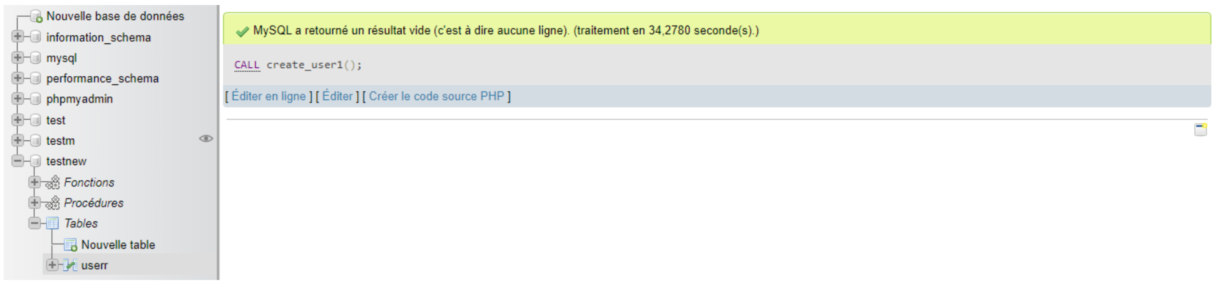
Figure 11 : execution time of inserting 10 user (Mysql)



```
MySQL a retourné un résultat vide (c'est à dire aucune ligne). (traitement en 4,4899 seconde(s).)

CALL create_user4();
```

[Éditer en ligne] [Éditer] [Créer le code source PHP]

Figure 12 : execution time of inserting 100 user (Mysql)



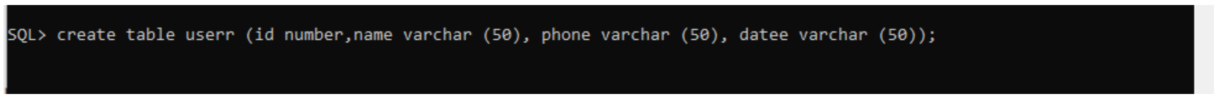
MySQL a retourné un résultat vide (c'est à dire aucune ligne). (traitement en 34,2780 seconde(s).)

```
CALL create_user1();
```

Figure 13 : execution time of inserting 1000 user (Mysql)

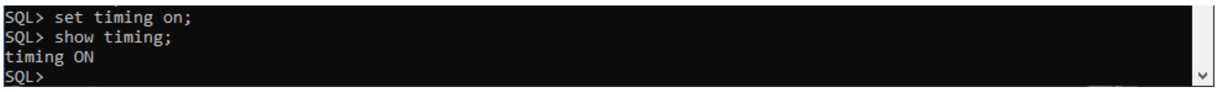
➢ Oracle Test :

- Create table userr



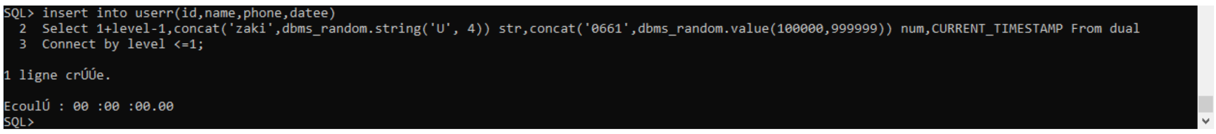
```
SQL> create table userr (id number,name varchar (50), phone varchar (50), datee varchar (50));
```

- Set Timing on :



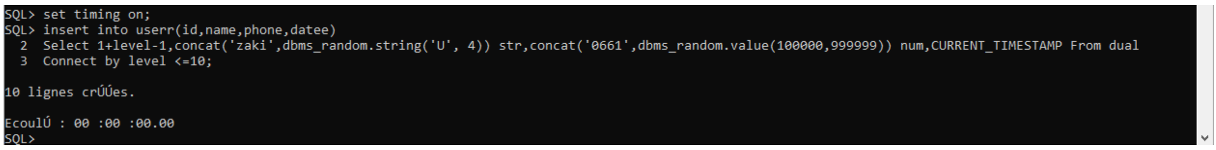
```
SQL> set timing on;
SQL> show timing;
timing ON
SQL>
```

- Generating / inserting 1,10,100,100 users and test result



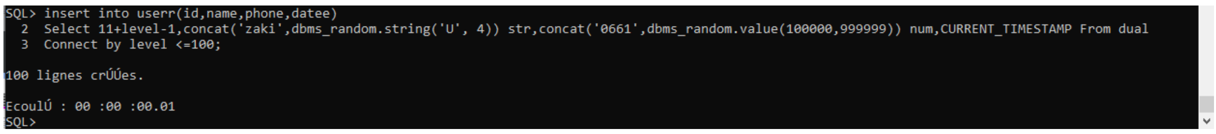
```
SQL> insert into userr(id,name,phone,datee)
  2  Select 1+level-1,concat('zaki',dbms_random.string('U', 4)) str,concat('0661',dbms_random.value(100000,999999)) num,CURRENT_TIMESTAMP From dual
  3  Connect by level <=1;

1 ligne crÚÚe.

EcoulÚ : 00 :00 :00.00
SQL>
```

Figure 14 : insert 1 user (Oracle)



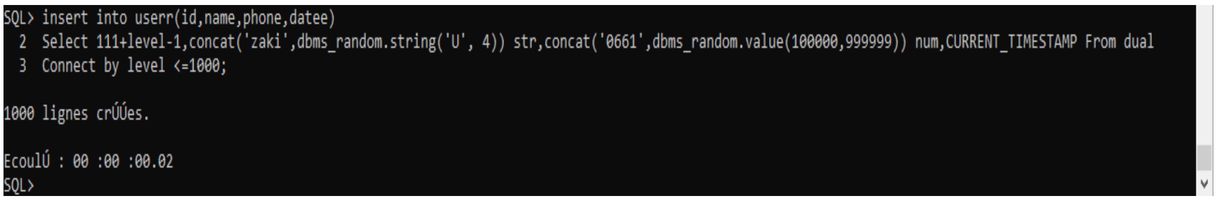
```
SQL> set timing on;
SQL> insert into userr(id,name,phone,datee)
  2  Select 1+level-1,concat('zaki',dbms_random.string('U', 4)) str,concat('0661',dbms_random.value(100000,999999)) num,CURRENT_TIMESTAMP From dual
  3  Connect by level <=10;

10 lignes crÚÚes.

EcoulÚ : 00 :00 :00.00
SQL>
```

Figure 15 : insert 10 userr (Oracle)



```
SQL> insert into userr(id,name,phone,datee)
  2  Select 11+level-1,concat('zaki',dbms_random.string('U', 4)) str,concat('0661',dbms_random.value(100000,999999)) num,CURRENT_TIMESTAMP From dual
  3  Connect by level <=100;

100 lignes crÚÚes.

EcoulÚ : 00 :00 :00.01
SQL>
```

Figure 16 : insert 100 user (Oracle)

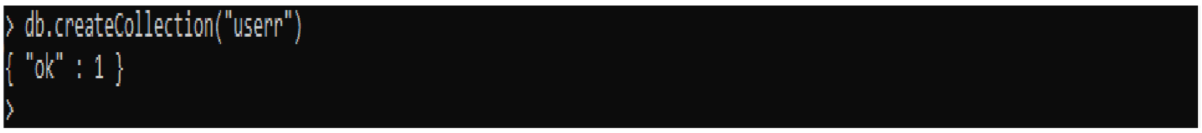


```
SQL> insert into userr(id,name,phone,datee)
  2  Select 111+level-1,concat('zaki',dbms_random.string('U', 4)) str,concat('0661',dbms_random.value(100000,999999)) num,CURRENT_TIMESTAMP From dual
  3  Connect by level <=1000;

1000 lignes crÚÚes.

EcoulÚ : 00 :00 :00.02
SQL>
```

Figure 17 : insert 1000 user (Oracle)

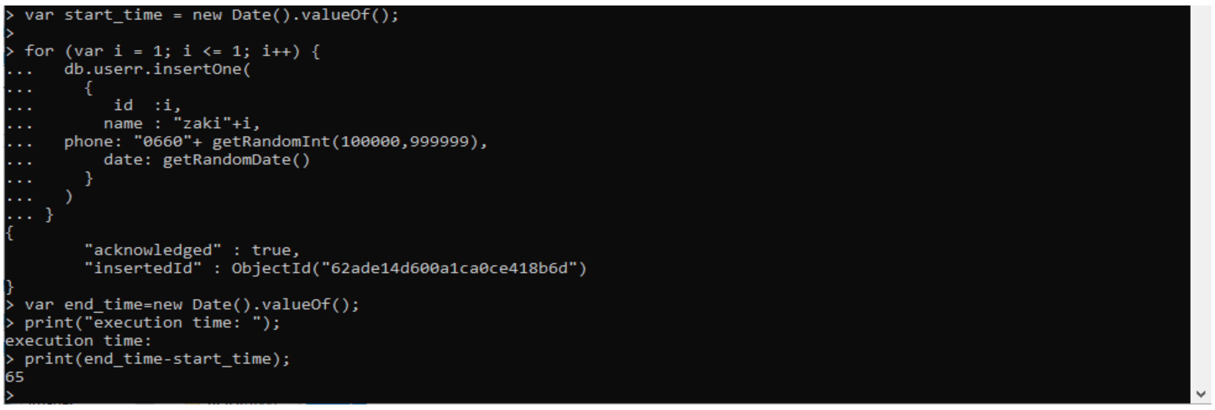
- **mongodb Test :**

- Create Collection users



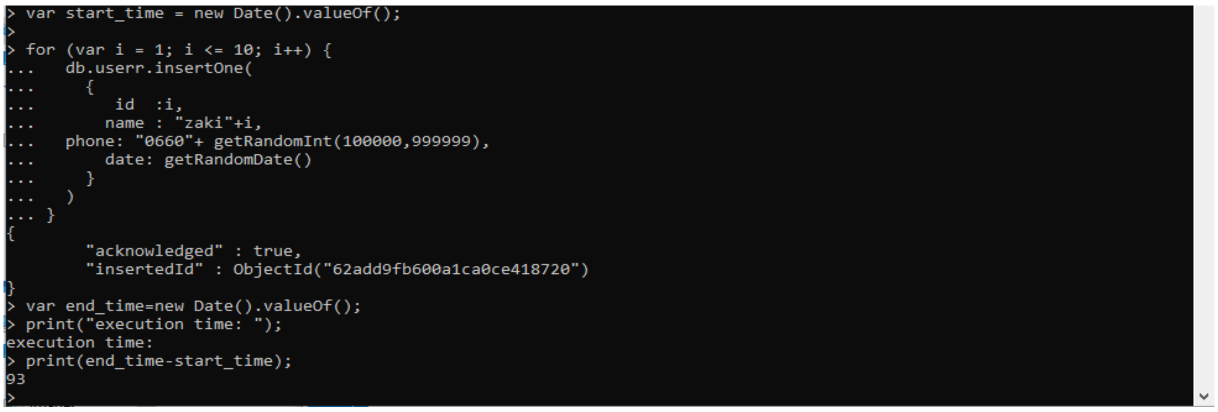
```
> db.createCollection("userr")
{ "ok" : 1 }
>
```

- Generating / inserting 1,10,100,1000 users and test result



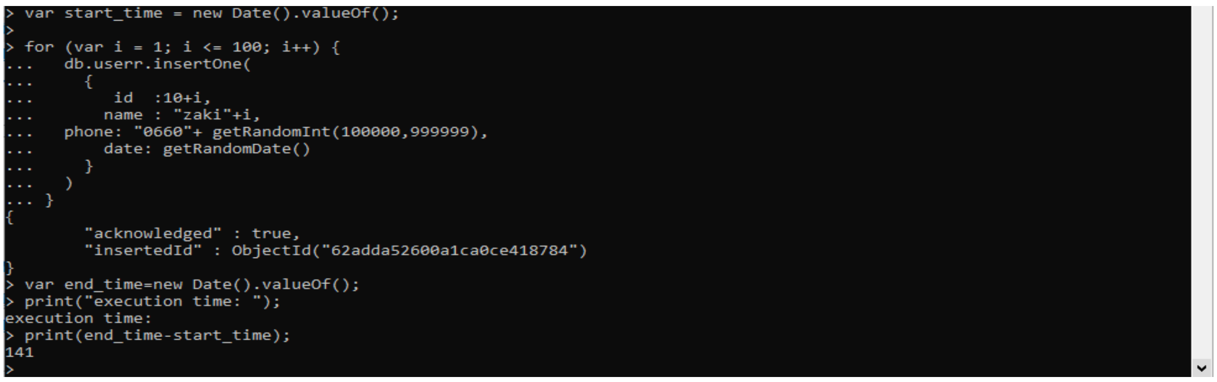
```
> var start_time = new Date().valueOf();
>
> for (var i = 1; i <= 1; i++) {
...    db.userr.insertOne(
...      {
...         id  :i,
...        name : "zaki"+i,
...    phone: "0660"+ getRandomInt(100000,999999),
...        date: getRandomDate()
...      }
...    )
... }
{
        "acknowledged" : true,
        "insertedId" : ObjectId("62ade14d600a1ca0ce418b6d")
}
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
65
>
```

Figure 18 : insert 1 user (Mongodb)



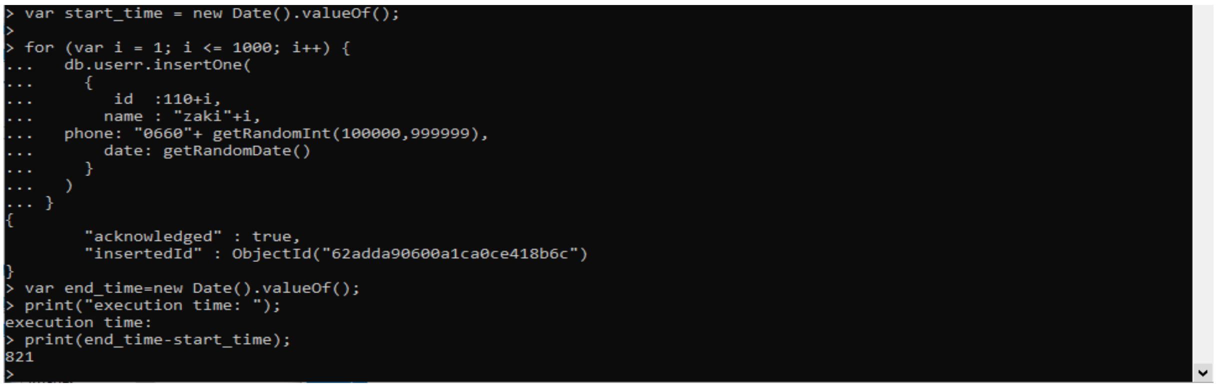
```
> var start_time = new Date().valueOf();
>
> for (var i = 1; i <= 10; i++) {
...    db.userr.insertOne(
...      {
...         id  :i,
...        name : "zaki"+i,
...    phone: "0660"+ getRandomInt(100000,999999),
...        date: getRandomDate()
...      }
...    )
... }
{
        "acknowledged" : true,
        "insertedId" : ObjectId("62add9fb600a1ca0ce418720")
}
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
93
>
```

Figure 19 : insert 10 user (Mongodb)



```
> var start_time = new Date().valueOf();
>
> for (var i = 1; i <= 100; i++) {
...    db.userr.insertOne(
...      {
...         id  :10+i,
...        name : "zaki"+i,
...    phone: "0660"+ getRandomInt(100000,999999),
...        date: getRandomDate()
...      }
...    )
... }
{
        "acknowledged" : true,
        "insertedId" : ObjectId("62adda52600a1ca0ce418784")
}
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
141
>
```

Figure 20 : : insert 100 user (Mongodb)

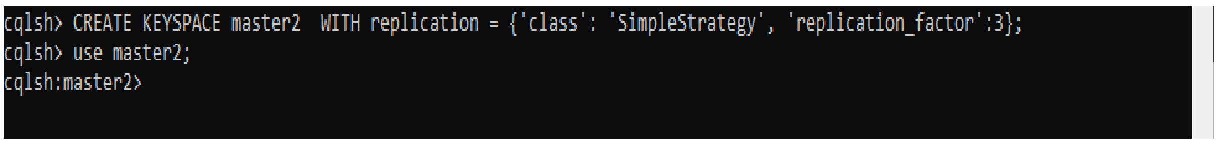


```
> var start_time = new Date().valueOf();
>
> for (var i = 1; i <= 1000; i++) {
...   db.userr.insertOne(
...     {
...        id  :110+i,
...       name : "zaki"+i,
...   phone: "0660"+ getRandomInt(100000,999999),
...       date: getRandomDate()
...     }
...   )
... }
{
        "acknowledged" : true,
        "insertedId" : ObjectId("62adda90600a1ca0ce418b6c")
}
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
821
>
```

Figure 21 : insert 1000 user (Mongodb)

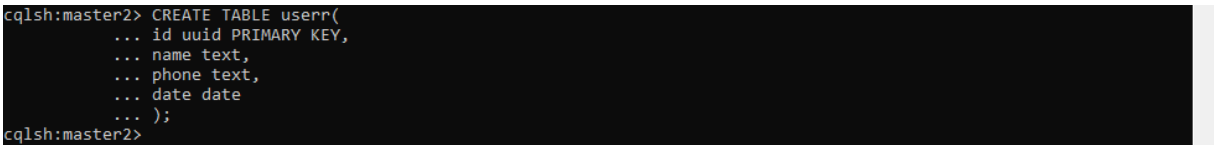
➢ **Cassandra Test :**

- Create KEY SPACE



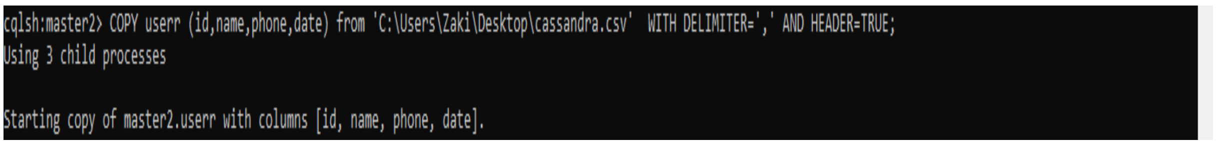
```
cqlsh> CREATE KEYSPACE master2  WITH replication = {'class': 'SimpleStrategy', 'replication_factor':3};
cqlsh> use master2;
cqlsh:master2>
```

- Create table users



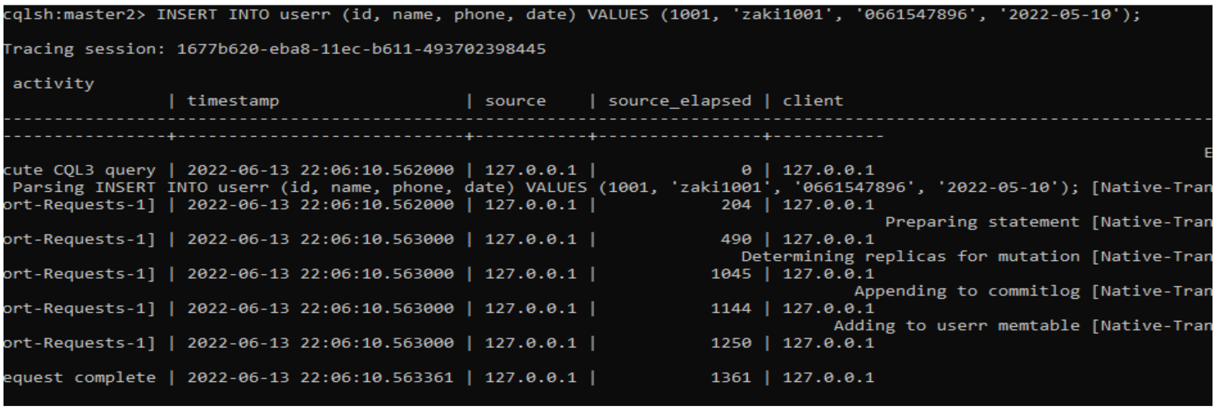
```
cqlsh:master2> CREATE TABLE userr(
           ... id uuid PRIMARY KEY,
           ... name text,
           ... phone text,
           ... date date
           ... );
cqlsh:master2>
```

- Import data base from cassandra.csv file



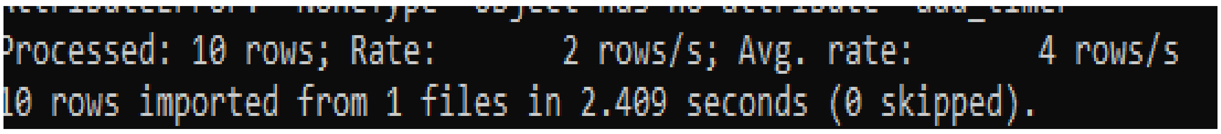
```
cqlsh:master2> COPY userr (id,name,phone,date) from 'C:\Users\Zaki\Desktop\cassandra.csv'  WITH DELIMITER=',' AND HEADER=TRUE;
Using 3 child processes

Starting copy of master2.userr with columns [id, name, phone, date].
```

- The result of test



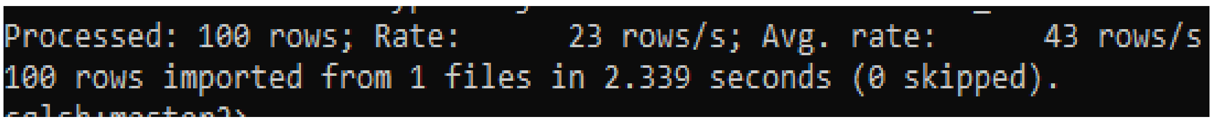
```
cqlsh:master2> INSERT INTO userr (id, name, phone, date) VALUES (1001, 'zaki1001', '0661547896', '2022-05-10');

Tracing session: 1677b620-eba8-11ec-b611-493702398445

 activity
                | timestamp                  | source    | source_elapsed | client
-----------------------------------------------------------------------------------------------------------------------
----------------+----------------------------+-----------+----------------+-----------
                                                                                                                      E
cute CQL3 query | 2022-06-13 22:06:10.562000 | 127.0.0.1 |              0 | 127.0.0.1
 Parsing INSERT INTO userr (id, name, phone, date) VALUES (1001, 'zaki1001', '0661547896', '2022-05-10'); [Native-Tran
ort-Requests-1] | 2022-06-13 22:06:10.562000 | 127.0.0.1 |            204 | 127.0.0.1
                                                                             Preparing statement [Native-Tran
ort-Requests-1] | 2022-06-13 22:06:10.563000 | 127.0.0.1 |            490 | 127.0.0.1
                                                                Determining replicas for mutation [Native-Tran
ort-Requests-1] | 2022-06-13 22:06:10.563000 | 127.0.0.1 |           1045 | 127.0.0.1
                                                                           Appending to commitlog [Native-Tran
ort-Requests-1] | 2022-06-13 22:06:10.563000 | 127.0.0.1 |           1144 | 127.0.0.1
                                                                         Adding to userr memtable [Native-Tran
ort-Requests-1] | 2022-06-13 22:06:10.563000 | 127.0.0.1 |           1250 | 127.0.0.1

equest complete | 2022-06-13 22:06:10.563361 | 127.0.0.1 |           1361 | 127.0.0.1
```

Figure 22 : insert 1 user (Cassandra)



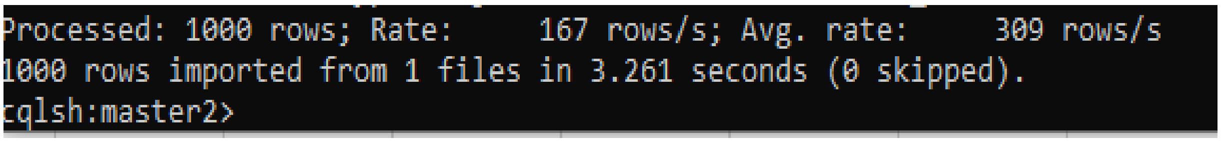
```
Processed: 10 rows; Rate:       2 rows/s; Avg. rate:       4 rows/s
10 rows imported from 1 files in 2.409 seconds (0 skipped).
```

Figure 23 : insert 10 user (Cassandra)



```
Processed: 100 rows; Rate:      23 rows/s; Avg. rate:      43 rows/s
100 rows imported from 1 files in 2.339 seconds (0 skipped).
```

Figure 24 : insert 100 user (Cassandra)



```
Processed: 1000 rows; Rate:     167 rows/s; Avg. rate:     309 rows/s
1000 rows imported from 1 files in 3.261 seconds (0 skipped).
cqlsh:master2>
```

Figure 25 : insert 1000 user (Cassandra)

➢ **NEWSQL Test :**

During our work, we encountered many problems in installing NewSql system such as: Voltdb, Nuodb. For this reason, we relied in our project on the results obtained in the work [21] to complete our test.

The results of inserting 1/10/100/1000 users are summarized in the following table:

	Databases	Execution time (sec) 1 user	Execution time (sec) 10 user	Execution time (sec) 100 user	Execution time (sec) 1000 user
Sql	Mysql	0.0005 s	0.4237	4.4899	34.278
	Oracle	0.0001 s	00.001	00.01	00.02
Nosql	Mongodb	0.065	0.093	0.141	0.821
	Cassandra	1.361	2.409	2.339	3.261
NewSql	Voltdb	-	0.369	2.656	28.666
	Nuodb	-	-	-	-

Tableau 9 : Performance test results (insert 1/10/100/1000 users)

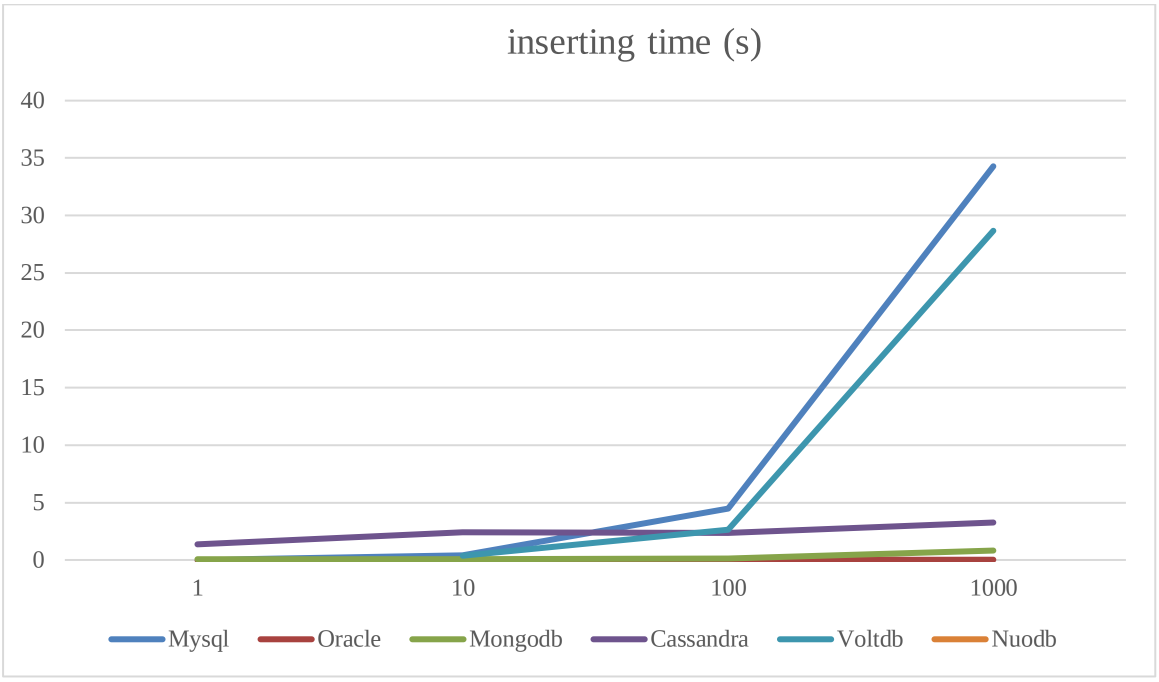
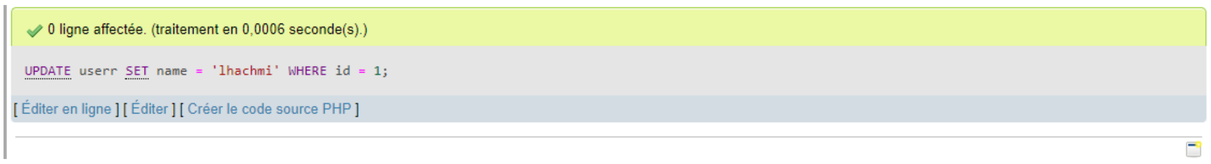


Figure 26 : curve represent the execution time of inserting 1/10/100/1000 users

In Figure 6, experiments performed on all six systems with different records 1/10/100/1000 of insert rates, proved that Oracle was slightly performant compared to the others, with an overall runtime 00.02 sec, against 3.261 sec presented by Cassandra, Mongodb with 0.821 sec and Voltdb with 28.866sec . We can observe that Mysql performs the worst overall runtime with 34.278 sec.

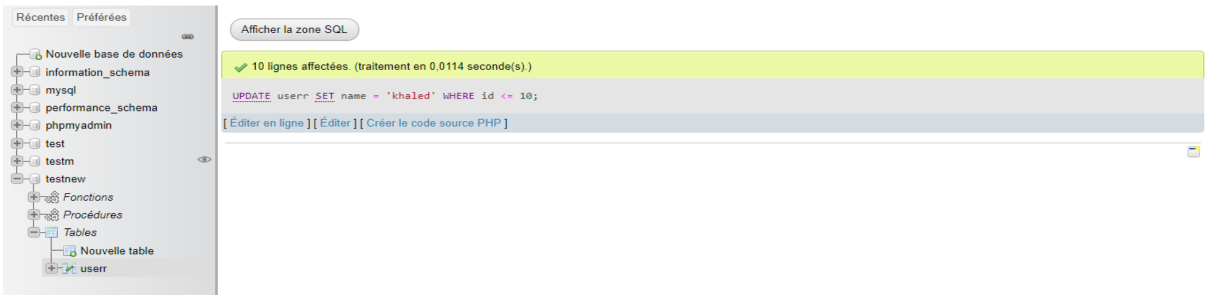
2. update of 1 ,10,100 and 1000 users

➢ Mysql(exampp) test:



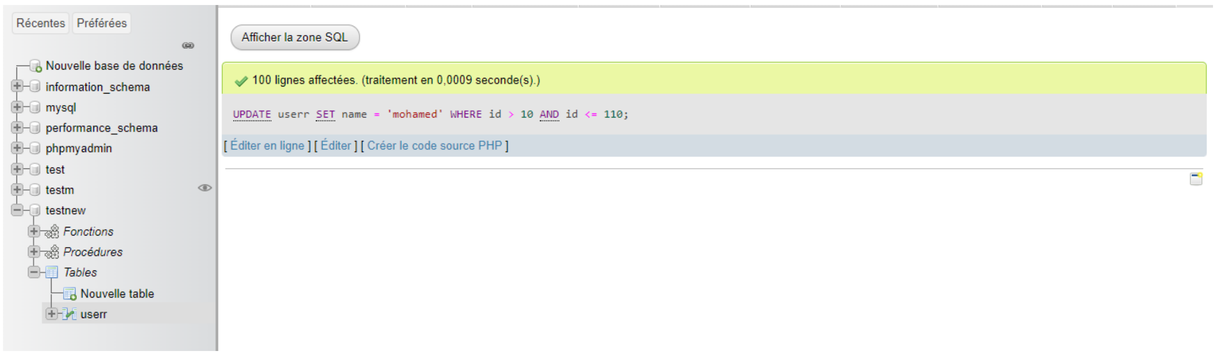
```
✓ 0 ligne affectée. (traitement en 0,0006 seconde(s).)

UPDATE userr SET name = 'lhachmi' WHERE id = 1;

[ Éditer en ligne ] [ Éditer ] [ Créer le code source PHP ]
```

Figure 27: Update 1 user (Mysql)



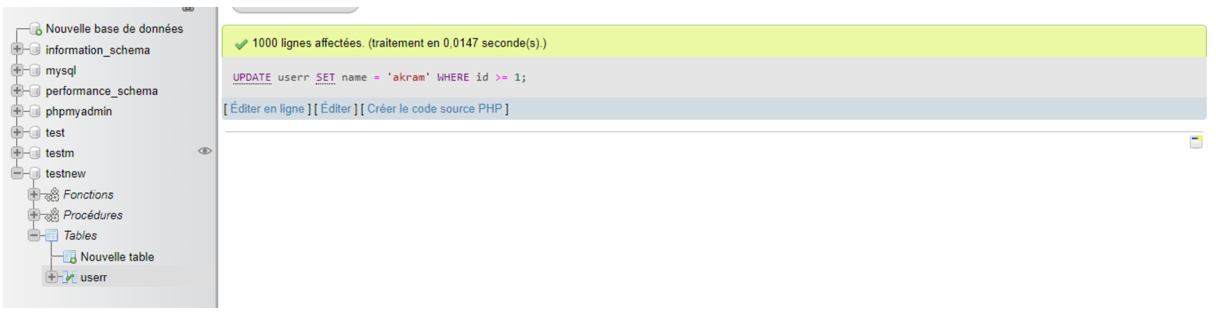
```
✓ 10 lignes affectées. (traitement en 0,0114 seconde(s).)

UPDATE userr SET name = 'khaled' WHERE id <= 10;

[ Éditer en ligne ] [ Éditer ] [ Créer le code source PHP ]
```

Figure 28: Update 10 user (Mysql)



```
✓ 100 lignes affectées. (traitement en 0,0009 seconde(s).)
UPDATE userr SET name = 'mohamed' WHERE id > 10 AND id <= 110;
[ Éditer en ligne ] [ Éditer ] [ Créer le code source PHP ]
```

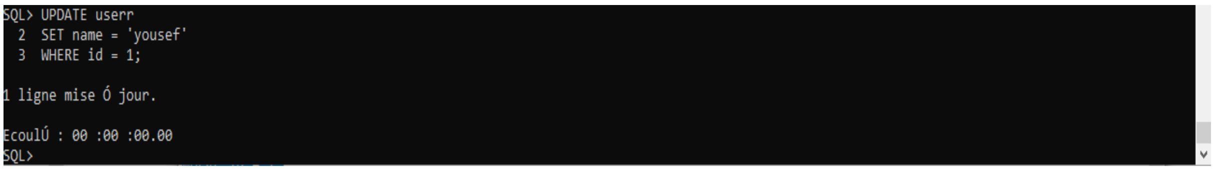
Figure 29: Update 100 user (Mysql)



```
✓ 1000 lignes affectées. (traitement en 0,0147 seconde(s).)
UPDATE userr SET name = 'akram' WHERE id >= 1;
[ Éditer en ligne ] [ Éditer ] [ Créer le code source PHP ]
```

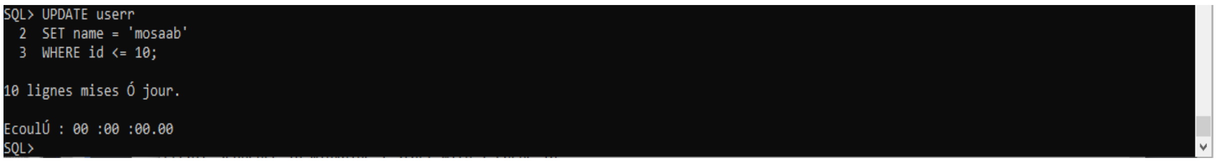
Figure 30: Update 1000 user (Mysql)

➢ Oracle Test :



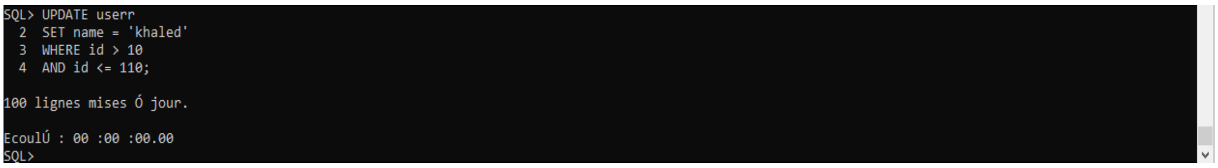
```
SQL> UPDATE userr
  2  SET name = 'yousef'
  3  WHERE id = 1;

1 ligne mise Ó jour.

EcoulÚ : 00 :00 :00.00
SQL>
```

Figure 31: Update 1 user (Oracle)



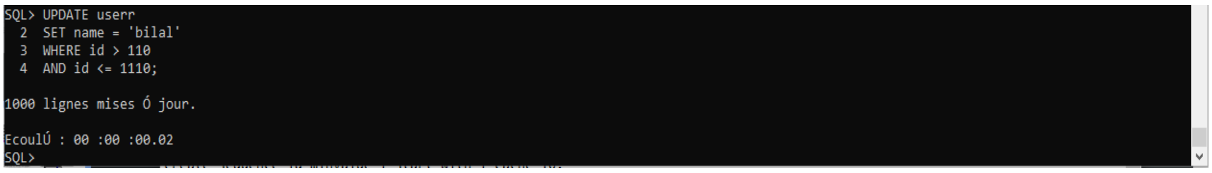
```
SQL> UPDATE userr
  2  SET name = 'mosaab'
  3  WHERE id <= 10;

10 lignes mises Ó jour.

EcoulÚ : 00 :00 :00.00
SQL>
```

Figure 32: Update 10 user (Oracle)



```
SQL> UPDATE userr
  2  SET name = 'khaled'
  3  WHERE id > 10
  4  AND id <= 110;

100 lignes mises Ó jour.

EcoulÚ : 00 :00 :00.00
SQL>
```

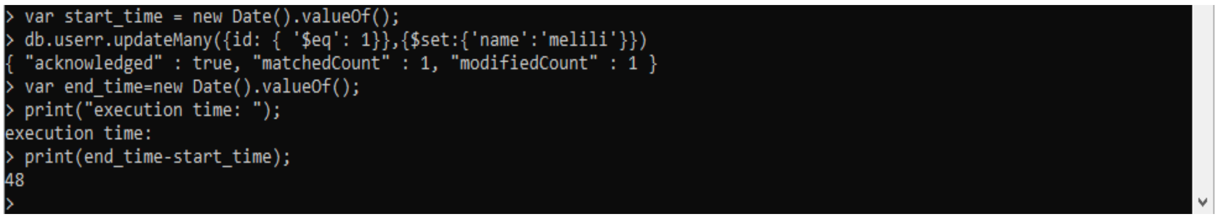
Figure 33: Update 100 user (Oracle)



```
SQL> UPDATE userr
  2  SET name = 'bilal'
  3  WHERE id > 110
  4  AND id <= 1110;

1000 lignes mises Ó jour.

EcoulÚ : 00 :00 :00.02
SQL>
```

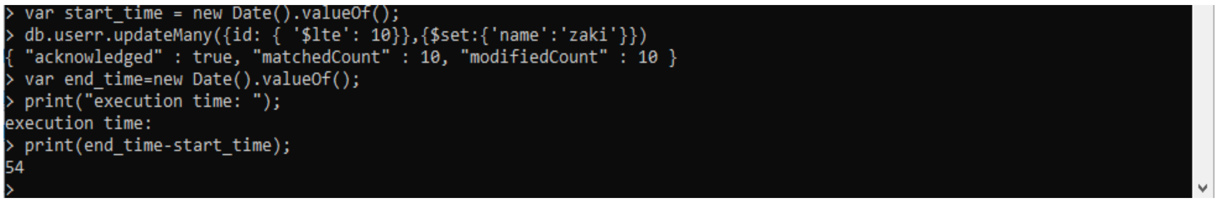
Figure 34: Update 1000 user (Oracle)

➢ Mongodb Test :



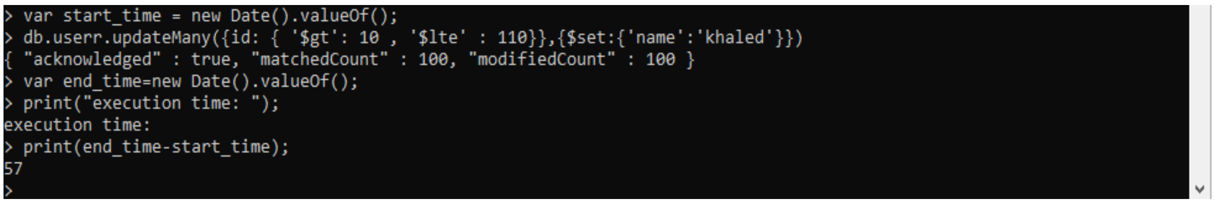
```
> var start_time = new Date().valueOf();
> db.userr.updateMany({id: { '$eq': 1}},{$set:{'name':'melili'}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
48
>
```

Figure 35: Update 1 user (Mongodb)



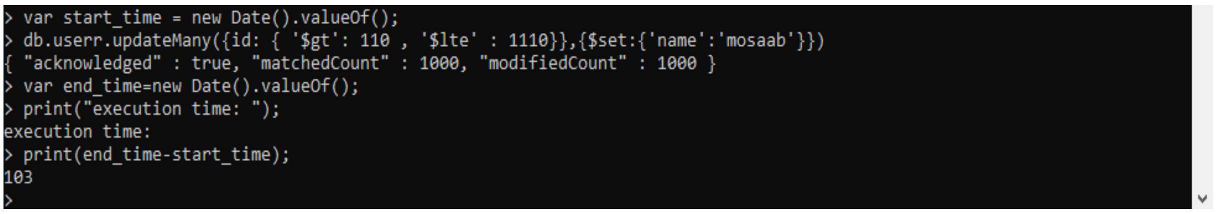
```
> var start_time = new Date().valueOf();
> db.userr.updateMany({id: { '$lte': 10}},{$set:{'name':'zaki'}})
{ "acknowledged" : true, "matchedCount" : 10, "modifiedCount" : 10 }
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
54
>
```

Figure 36: Update 10 user (Mongodb)



```
> var start_time = new Date().valueOf();
> db.userr.updateMany({id: { '$gt': 10 , '$lte' : 110}},{$set:{'name':'khaled'}})
{ "acknowledged" : true, "matchedCount" : 100, "modifiedCount" : 100 }
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
57
>
```

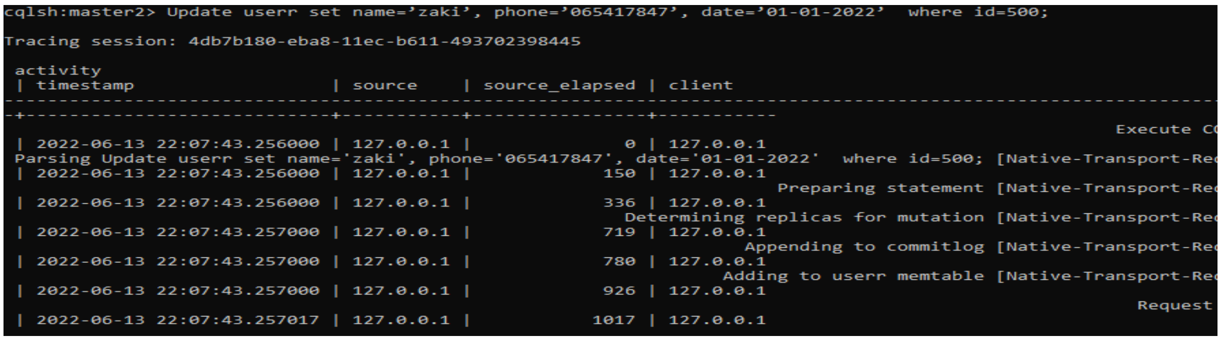
Figure 37: Update 100 user (Mongodb)



```
> var start_time = new Date().valueOf();
> db.userr.updateMany({id: { '$gt': 110 , '$lte' : 1110}},{$set:{'name':'mosaab'}})
{ "acknowledged" : true, "matchedCount" : 1000, "modifiedCount" : 1000 }
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
103
>
```

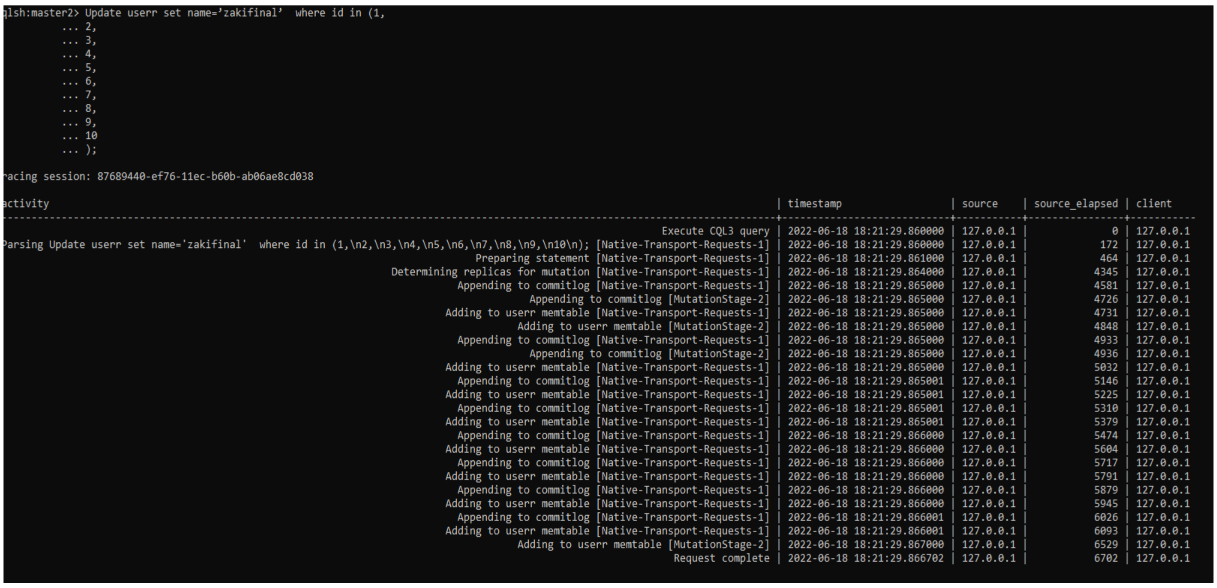
Figure 38: Update 1000 user (Mongodb)

➢ Cassandra Test :



```
cqlsh:master2> Update userr set name=’zaki’, phone=’065417847’, date=’01-01-2022’  where id=500;

Tracing session: 4db7b180-eba8-11ec-b611-493702398445

 activity
 | timestamp                  | source    | source_elapsed | client
------------------------------------------------------------------------------------------------------------
-+----------------------------+-----------+----------------+-----------
                                                                                                 Execute C
 | 2022-06-13 22:07:43.256000 | 127.0.0.1 |              0 | 127.0.0.1
 Parsing Update userr set name='zaki', phone='065417847', date='01-01-2022'  where id=500; [Native-Transport-Re
 | 2022-06-13 22:07:43.256000 | 127.0.0.1 |            150 | 127.0.0.1
                                                                      Preparing statement [Native-Transport-Re
 | 2022-06-13 22:07:43.256000 | 127.0.0.1 |            336 | 127.0.0.1
                                                         Determining replicas for mutation [Native-Transport-Re
 | 2022-06-13 22:07:43.257000 | 127.0.0.1 |            719 | 127.0.0.1
                                                                    Appending to commitlog [Native-Transport-Re
 | 2022-06-13 22:07:43.257000 | 127.0.0.1 |            780 | 127.0.0.1
                                                                  Adding to userr memtable [Native-Transport-Re
 | 2022-06-13 22:07:43.257000 | 127.0.0.1 |            926 | 127.0.0.1
                                                                                                       Request
 | 2022-06-13 22:07:43.257017 | 127.0.0.1 |           1017 | 127.0.0.1
```

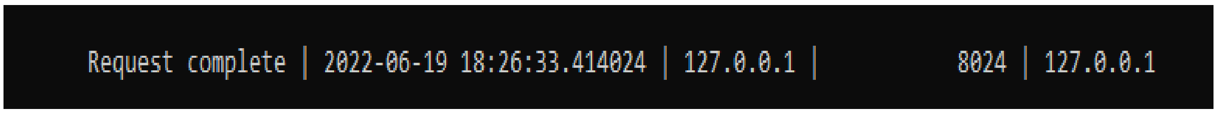
Figure 39: Update 1 user (Cassandra)



```
qlsh:master2> Update userr set name='zakifinal'  where id in (1,
          ... 2,
          ... 3,
          ... 4,
          ... 5,
          ... 6,
          ... 7,
          ... 8,
          ... 9,
          ... 10
          ... );

racing session: 87689440-ef76-11ec-b60b-ab06ae8cd038

activity                                                                                                                         | timestamp                  | source    | source_elapsed | client
---------------------------------------------------------------------------------------------------------------------------------+----------------------------+-----------+----------------+-----------
                                                                                                              Execute CQL3 query | 2022-06-18 18:21:29.860000 | 127.0.0.1 |              0 | 127.0.0.1
Parsing Update userr set name='zakifinal'  where id in (1,\n2,\n3,\n4,\n5,\n6,\n7,\n8,\n9,\n10\n); [Native-Transport-Requests-1] | 2022-06-18 18:21:29.860000 | 127.0.0.1 |            172 | 127.0.0.1
                                                               Preparing statement [Native-Transport-Requests-1] | 2022-06-18 18:21:29.861000 | 127.0.0.1 |            464 | 127.0.0.1
                                                 Determining replicas for mutation [Native-Transport-Requests-1] | 2022-06-18 18:21:29.864000 | 127.0.0.1 |           4345 | 127.0.0.1
                                                            Appending to commitlog [Native-Transport-Requests-1] | 2022-06-18 18:21:29.865000 | 127.0.0.1 |           4581 | 127.0.0.1
                                                                        Appending to commitlog [MutationStage-2] | 2022-06-18 18:21:29.865000 | 127.0.0.1 |           4726 | 127.0.0.1
                                                          Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-18 18:21:29.865000 | 127.0.0.1 |           4731 | 127.0.0.1
                                                                      Adding to userr memtable [MutationStage-2] | 2022-06-18 18:21:29.865000 | 127.0.0.1 |           4848 | 127.0.0.1
                                                            Appending to commitlog [Native-Transport-Requests-1] | 2022-06-18 18:21:29.865000 | 127.0.0.1 |           4933 | 127.0.0.1
                                                                        Appending to commitlog [MutationStage-2] | 2022-06-18 18:21:29.865000 | 127.0.0.1 |           4936 | 127.0.0.1
                                                          Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-18 18:21:29.865000 | 127.0.0.1 |           5032 | 127.0.0.1
                                                            Appending to commitlog [Native-Transport-Requests-1] | 2022-06-18 18:21:29.865001 | 127.0.0.1 |           5146 | 127.0.0.1
                                                          Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-18 18:21:29.865001 | 127.0.0.1 |           5225 | 127.0.0.1
                                                            Appending to commitlog [Native-Transport-Requests-1] | 2022-06-18 18:21:29.865001 | 127.0.0.1 |           5310 | 127.0.0.1
                                                          Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-18 18:21:29.865001 | 127.0.0.1 |           5379 | 127.0.0.1
                                                            Appending to commitlog [Native-Transport-Requests-1] | 2022-06-18 18:21:29.866000 | 127.0.0.1 |           5474 | 127.0.0.1
                                                          Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-18 18:21:29.866000 | 127.0.0.1 |           5604 | 127.0.0.1
                                                            Appending to commitlog [Native-Transport-Requests-1] | 2022-06-18 18:21:29.866000 | 127.0.0.1 |           5717 | 127.0.0.1
                                                          Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-18 18:21:29.866000 | 127.0.0.1 |           5791 | 127.0.0.1
                                                            Appending to commitlog [Native-Transport-Requests-1] | 2022-06-18 18:21:29.866000 | 127.0.0.1 |           5879 | 127.0.0.1
                                                          Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-18 18:21:29.866000 | 127.0.0.1 |           5945 | 127.0.0.1
                                                            Appending to commitlog [Native-Transport-Requests-1] | 2022-06-18 18:21:29.866001 | 127.0.0.1 |           6026 | 127.0.0.1
                                                          Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-18 18:21:29.866001 | 127.0.0.1 |           6093 | 127.0.0.1
                                                                      Adding to userr memtable [MutationStage-2] | 2022-06-18 18:21:29.867000 | 127.0.0.1 |           6529 | 127.0.0.1
                                                                                                Request complete | 2022-06-18 18:21:29.866702 | 127.0.0.1 |           6702 | 127.0.0.1
```

Figure 40: Update 10 user (Cassandra)

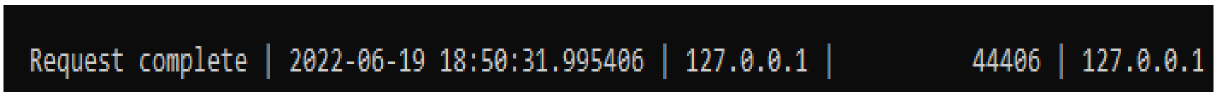
For updating 100 users Type : update ‘ userr‘ set name = ‘ zakifinal‘ where id in (1,2,3,4…… 100) ;



```
    Request complete | 2022-06-19 18:26:33.414024 | 127.0.0.1 |           8024 | 127.0.0.1
```

Figure 41: Update 100 user (Cassandra)

For updating 1000 user type : update ‘ userr‘ set name = ‘ zakifinal‘ where id in (1,2,3,4…… 100……..1000) ;



```
  Request complete | 2022-06-19 18:50:31.995406 | 127.0.0.1 |          44406 | 127.0.0.1
```

Figure 42: Update 1000 user (Cassandra)

The results of updating 1/10/100/1000 users are summarized in the following table:

	Databases	Execution time (sec) 1 user	Execution time (sec) 10 user	Execution time (sec) 100 user	Execution time (sec) 1000 user
Sql	Mysql	0.0005 s	0.0114	0.0009	0.0147
	Oracle	00.001 s	00.001	00.001	00.02

Nosql	Mongodb	0.048	0.054	0.057	0.103
	Cassandra	1.017 s	6.702	8.024	44.406 s
Newsql	Voltdb	-	-	-	0.026
	Nuodb	-	-	-	-

Tableau 10 : Performance test results (update 1,10,100,1000 users)

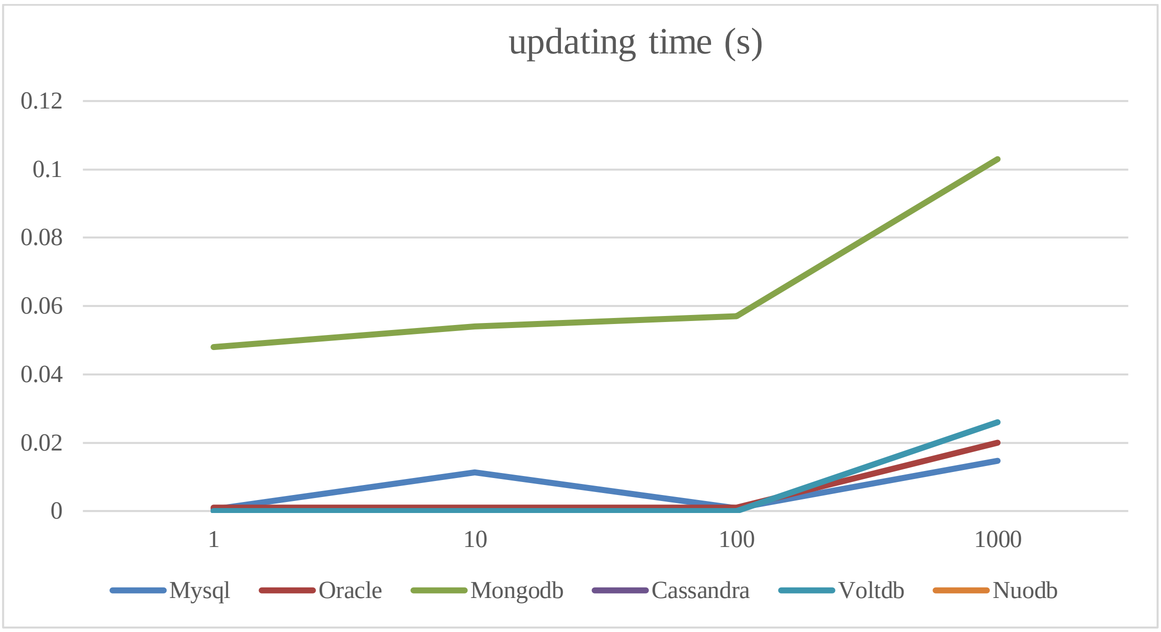


Figure 43 : curve represent the execution time of updating 1,10,100,1000 users(without Cassandra because she effects on the result in the curve)

Figure 7 shows the different records 1/10/100/1000 of updating rates, proved that Oracle , Mysql and Voltdb was nearly the same best performant compared to the rest of them , with an overall runtime 00.02 sec for oracle , against 0.0147 sec presented by Mysql , and Voltdb with 0.026 sec Mongodb with 0.103 sec.

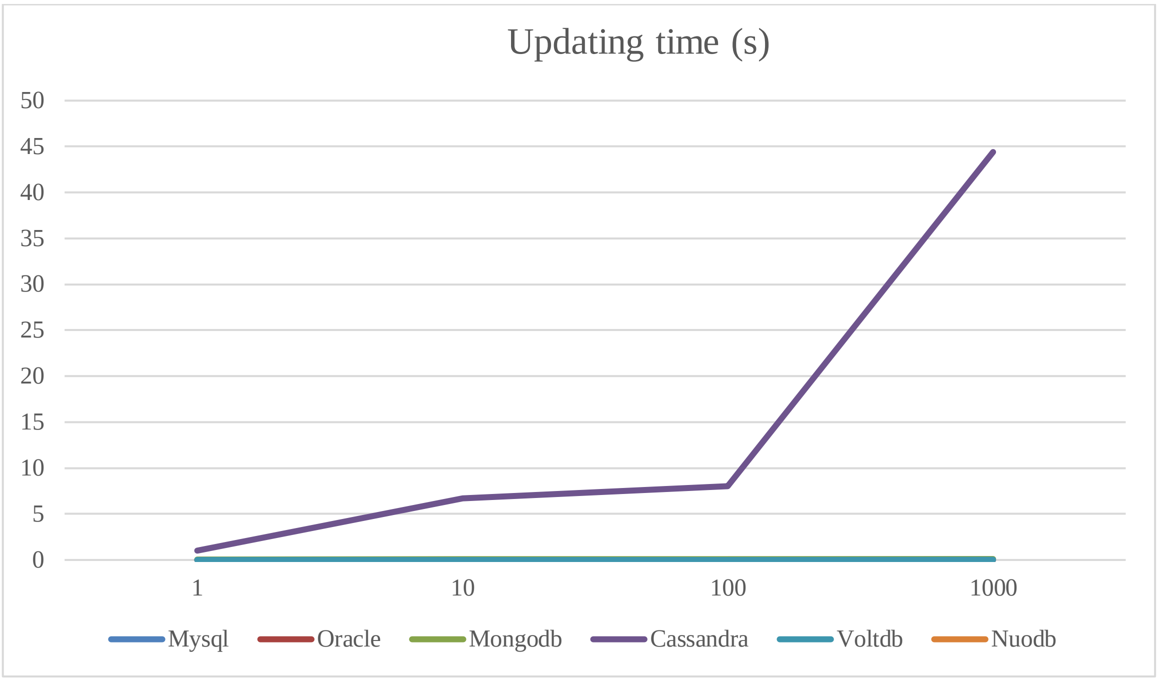
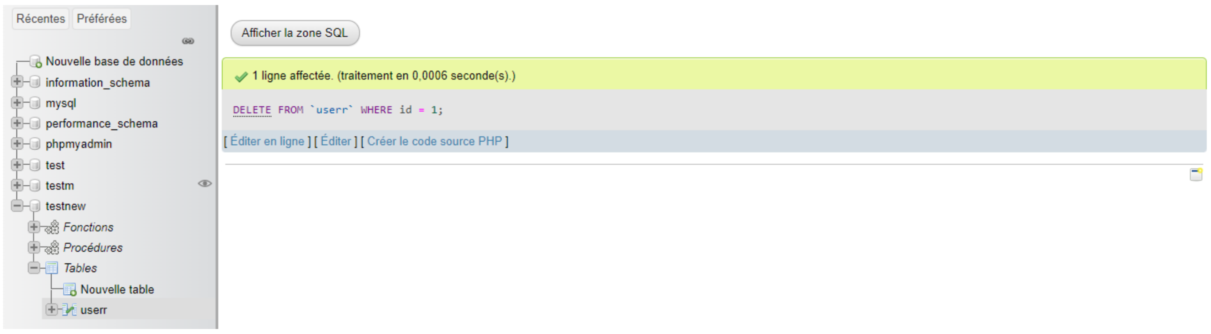


Figure 44 : curve represent the execution time of updating 1,10,100,1000 users (with Cassandra because she effects on the test curve result)

From figure 8, we can observe that Cassandra performs the worst overall runtime with 44.406 sec.

3. delete 1,10,100,1000 users :

➢ Mysql(exampp) test:

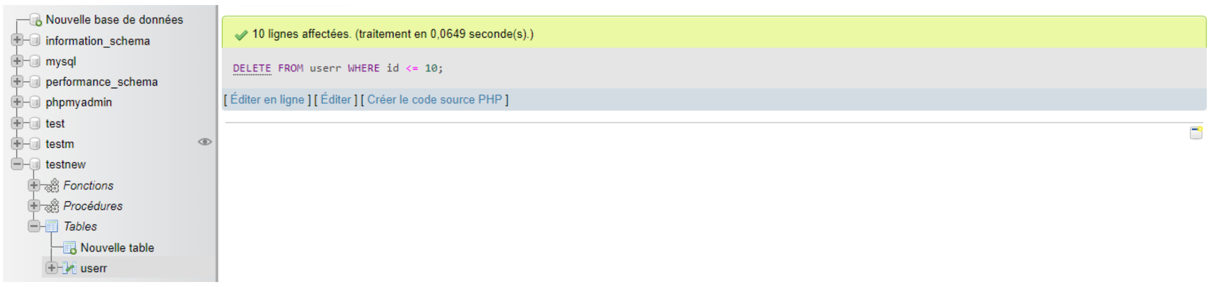


1 ligne affectée. (traitement en 0,0006 seconde(s).)

```
DELETE FROM `userr` WHERE id = 1;
```

[Éditer en ligne] [Éditer] [Créer le code source PHP]

Figure 45:Delete 1 user (Mysql)

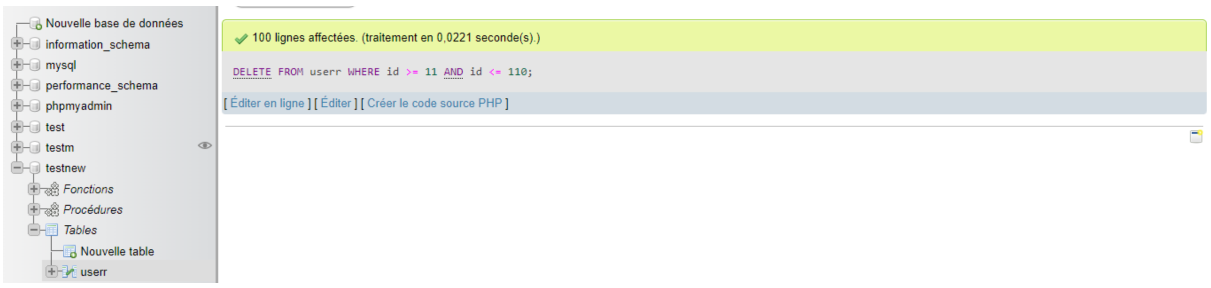


10 lignes affectées. (traitement en 0,0649 seconde(s).)

```
DELETE FROM userr WHERE id <= 10;
```

[Éditer en ligne] [Éditer] [Créer le code source PHP]

Figure 46:Delete 10 user (Mysql)

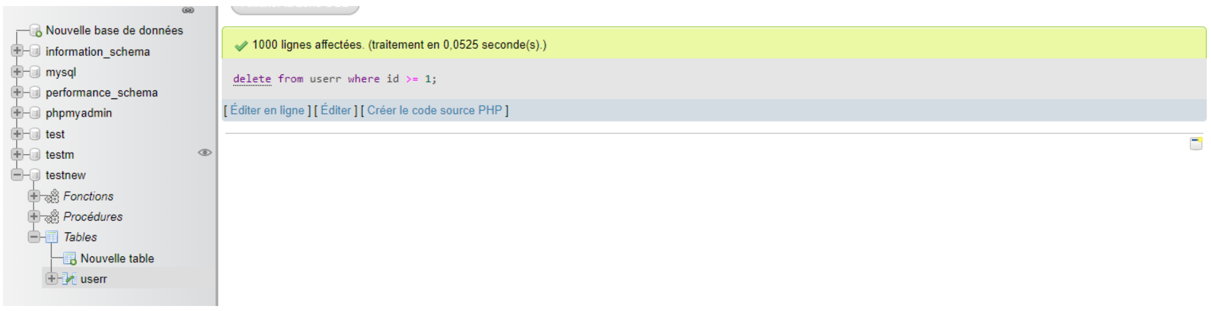


100 lignes affectées. (traitement en 0,0221 seconde(s).)

```
DELETE FROM userr WHERE id >= 11 AND id <= 110;
```

[Éditer en ligne] [Éditer] [Créer le code source PHP]

Figure 47:Delete 100 user (Mysql)



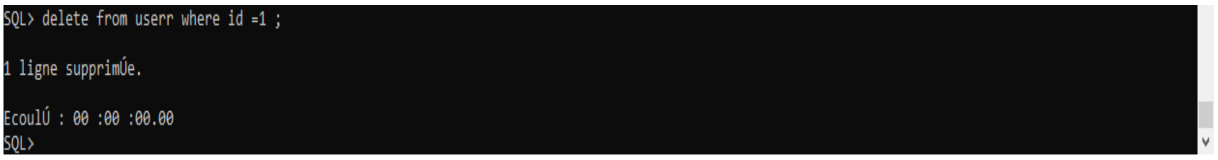
1000 lignes affectées. (traitement en 0,0525 seconde(s).)

```
delete from userr where id >= 1;
```

[Éditer en ligne] [Éditer] [Créer le code source PHP]

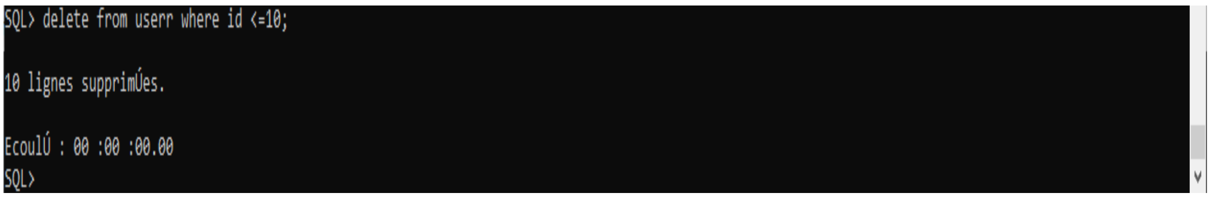
Figure 48:Delete 1000 user (Mysql)

➢ Oracle Test :



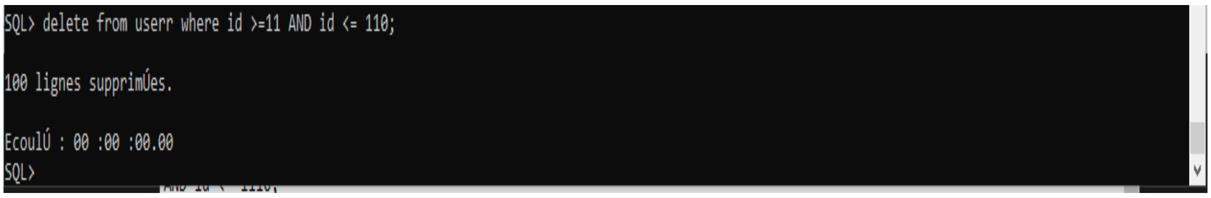
```
SQL> delete from userr where id =1 ;

1 ligne supprimÚe.

EcoulÚ : 00 :00 :00.00
SQL>
```

Figure 49:Delete 1 user (Oracle)



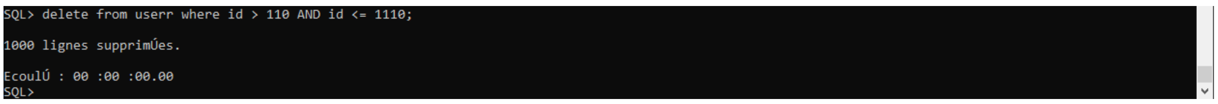
```
SQL> delete from userr where id <=10;

10 lignes supprimÚes.

EcoulÚ : 00 :00 :00.00
SQL>
```

Figure 50:Delete 10 user (Oracle)



```
SQL> delete from userr where id >=11 AND id <= 110;

100 lignes supprimÚes.

EcoulÚ : 00 :00 :00.00
SQL>
```

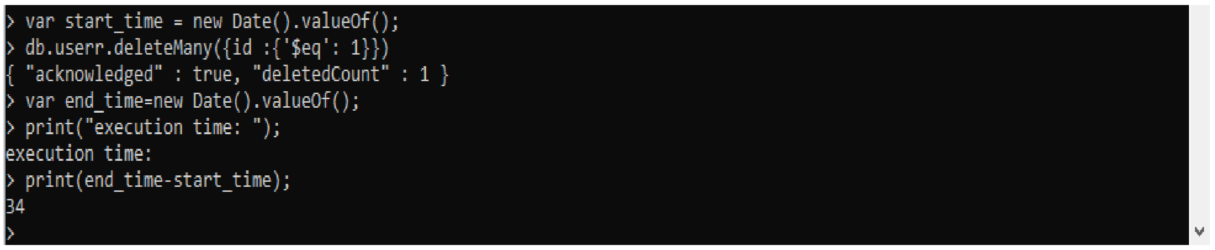
Figure 51:Delete 100 user (Oracle)



```
SQL> delete from userr where id > 110 AND id <= 1110;

1000 lignes supprimÚes.

EcoulÚ : 00 :00 :00.00
SQL>
```

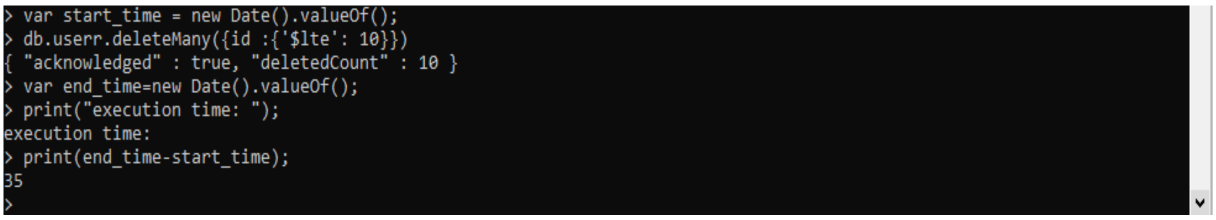
Figure 52:Delete 1000 user (Oracle)

➢ Mongodb Test :



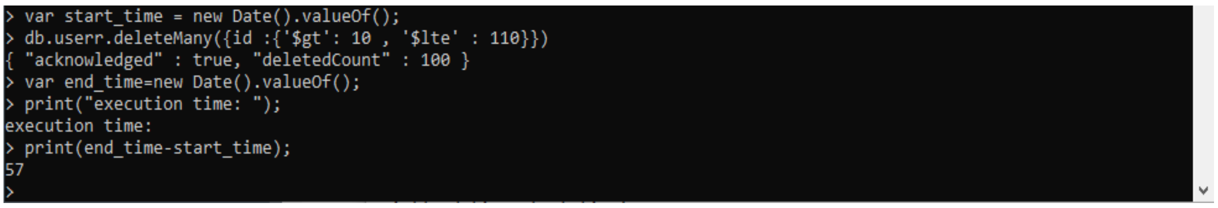
```
> var start_time = new Date().valueOf();
> db.userr.deleteMany({id :{'$eq': 1}})
{ "acknowledged" : true, "deletedCount" : 1 }
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
34
>
```

Figure 53:Delete 1 user (Mongodb)



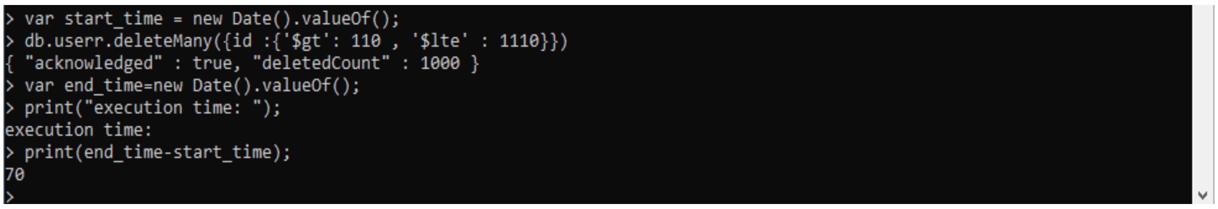
```
> var start_time = new Date().valueOf();
> db.userr.deleteMany({id :{'$lte': 10}})
{ "acknowledged" : true, "deletedCount" : 10 }
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
35
>
```

Figure 54:Delete 10 user (Mongodb)



```
> var start_time = new Date().valueOf();
> db.userr.deleteMany({id :{'$gt': 10 , '$lte' : 110}})
{ "acknowledged" : true, "deletedCount" : 100 }
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
57
>
```

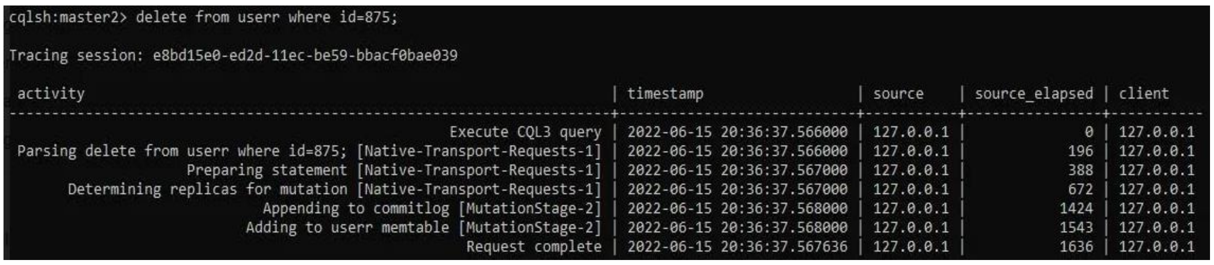
Figure 55:Delete 100 user (Mongodb)



```
> var start_time = new Date().valueOf();
> db.userr.deleteMany({id :{'$gt': 110 , '$lte' : 1110}})
{ "acknowledged" : true, "deletedCount" : 1000 }
> var end_time=new Date().valueOf();
> print("execution time: ");
execution time:
> print(end_time-start_time);
70
>
```

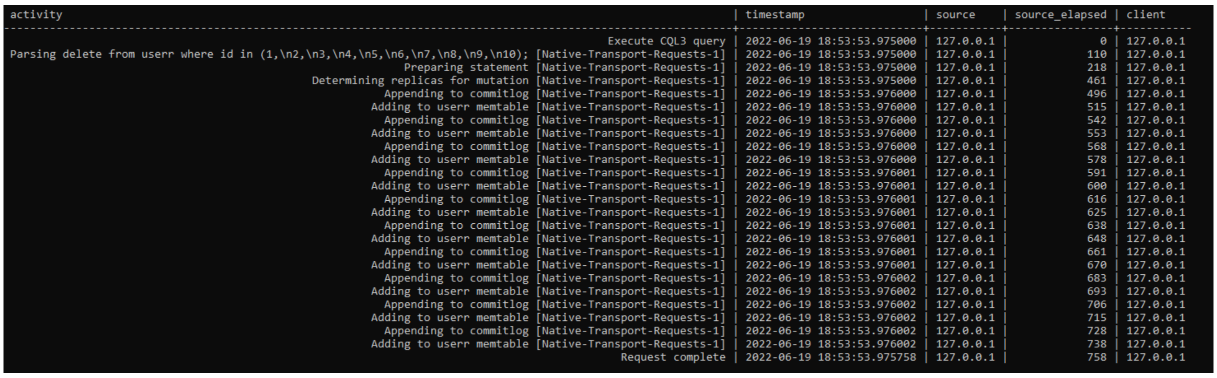
Figure 56:Delete 1000 user (Mongodb)

➢ Cassandra test :



```
cqlsh:master2> delete from userr where id=875;

Tracing session: e8bd15e0-ed2d-11ec-be59-bbacf0bae039

 activity                                                                 | timestamp                  | source    | source_elapsed | client
--------------------------------------------------------------------------+----------------------------+-----------+----------------+-----------
                                                       Execute CQL3 query | 2022-06-15 20:36:37.566000 | 127.0.0.1 |              0 | 127.0.0.1
 Parsing delete from userr where id=875; [Native-Transport-Requests-1] | 2022-06-15 20:36:37.566000 | 127.0.0.1 |            196 | 127.0.0.1
                       Preparing statement [Native-Transport-Requests-1] | 2022-06-15 20:36:37.567000 | 127.0.0.1 |            388 | 127.0.0.1
          Determining replicas for mutation [Native-Transport-Requests-1] | 2022-06-15 20:36:37.567000 | 127.0.0.1 |            672 | 127.0.0.1
                            Appending to commitlog [MutationStage-2] | 2022-06-15 20:36:37.568000 | 127.0.0.1 |           1424 | 127.0.0.1
                          Adding to userr memtable [MutationStage-2] | 2022-06-15 20:36:37.568000 | 127.0.0.1 |           1543 | 127.0.0.1
                                                         Request complete | 2022-06-15 20:36:37.567636 | 127.0.0.1 |           1636 | 127.0.0.1
```

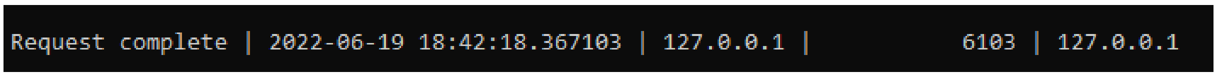
Figure 57 :Delete 1 user (Cassandra)



```
activity                                                                                                      | timestamp                  | source    | source_elapsed | client
--------------------------------------------------------------------------------------------------------------+----------------------------+-----------+----------------+-----------
                                                                                           Execute CQL3 query | 2022-06-19 18:53:53.975000 | 127.0.0.1 |              0 | 127.0.0.1
Parsing delete from userr where id in (1,\n2,\n3,\n4,\n5,\n6,\n7,\n8,\n9,\n10); [Native-Transport-Requests-1] | 2022-06-19 18:53:53.975000 | 127.0.0.1 |            110 | 127.0.0.1
                                                            Preparing statement [Native-Transport-Requests-1] | 2022-06-19 18:53:53.975000 | 127.0.0.1 |            218 | 127.0.0.1
                                              Determining replicas for mutation [Native-Transport-Requests-1] | 2022-06-19 18:53:53.975000 | 127.0.0.1 |            461 | 127.0.0.1
                                                         Appending to commitlog [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976000 | 127.0.0.1 |            496 | 127.0.0.1
                                                       Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976000 | 127.0.0.1 |            515 | 127.0.0.1
                                                         Appending to commitlog [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976000 | 127.0.0.1 |            542 | 127.0.0.1
                                                       Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976000 | 127.0.0.1 |            553 | 127.0.0.1
                                                         Appending to commitlog [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976000 | 127.0.0.1 |            568 | 127.0.0.1
                                                       Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976000 | 127.0.0.1 |            578 | 127.0.0.1
                                                         Appending to commitlog [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976001 | 127.0.0.1 |            591 | 127.0.0.1
                                                       Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976001 | 127.0.0.1 |            600 | 127.0.0.1
                                                         Appending to commitlog [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976001 | 127.0.0.1 |            616 | 127.0.0.1
                                                       Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976001 | 127.0.0.1 |            625 | 127.0.0.1
                                                         Appending to commitlog [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976001 | 127.0.0.1 |            638 | 127.0.0.1
                                                       Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976001 | 127.0.0.1 |            648 | 127.0.0.1
                                                         Appending to commitlog [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976001 | 127.0.0.1 |            661 | 127.0.0.1
                                                       Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976001 | 127.0.0.1 |            670 | 127.0.0.1
                                                         Appending to commitlog [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976002 | 127.0.0.1 |            683 | 127.0.0.1
                                                       Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976002 | 127.0.0.1 |            693 | 127.0.0.1
                                                         Appending to commitlog [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976002 | 127.0.0.1 |            706 | 127.0.0.1
                                                       Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976002 | 127.0.0.1 |            715 | 127.0.0.1
                                                         Appending to commitlog [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976002 | 127.0.0.1 |            728 | 127.0.0.1
                                                       Adding to userr memtable [Native-Transport-Requests-1] | 2022-06-19 18:53:53.976002 | 127.0.0.1 |            738 | 127.0.0.1
                                                                                             Request complete | 2022-06-19 18:53:53.975758 | 127.0.0.1 |            758 | 127.0.0.1
```

Figure 58 : Delete 10 user (Cassandra)

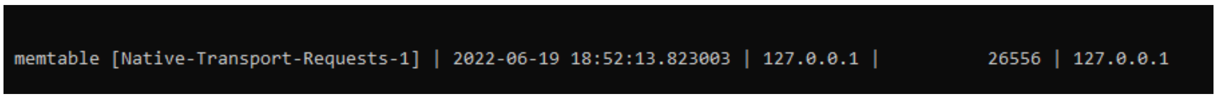
To delete 100 user type : TRUNcate userr ;



```
Request complete | 2022-06-19 18:42:18.367103 | 127.0.0.1 |           6103 | 127.0.0.1
```

Figure 59: Delete 100 user (Cassandra)

To delete 1000 userr type : Truncate userr ;



```
memtable [Native-Transport-Requests-1] | 2022-06-19 18:52:13.823003 | 127.0.0.1 |          26556 | 127.0.0.1
```

Figure 60: Delete 1000 user (Cassandra)

The results of deleting 1,10,100,1000 users are summarized in the following table:

	Databases	Execution time (sec) 1 user	Execution time (sec) 10 user	Execution time (sec) 100 user	Execution time (sec) 1000 user
Sql	Mysql	0.0005 s	0.0649	0.0221	0.0525
	Oracle	00.001 s	00.001	00.001	00.001
Nosql	Mongodb	0.034	0.035	0.057	0.07
	Cassandra	1.636 s	0.758	6.103	26.556
Newsql	Voltdb	-	0.020	0.020	0.022
	Nuodb	-	-	-	-

Tableau 11 : Performance test results (delete 1,10,100,1000 users)

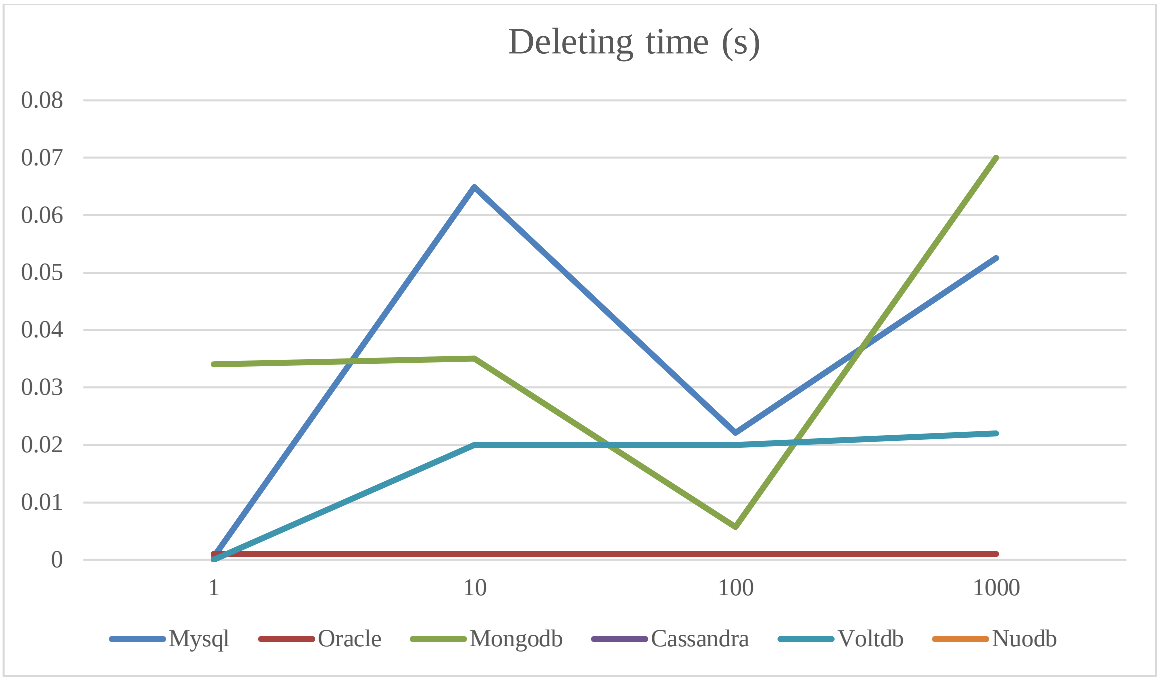


Figure 61 : curve represent the execution time of Deleting 1,10,100,1000 user(without Cassandra)

Figure 9 shows the different records 1/10/100/1000 of deleting rates, proved that Oracle , still the best performant compared to all of them , with an overall runtime 00.001 sec for oracle , against 0.022 sec presented by Voltdb , and Mysql with 0.0525 sec and Mongodb with 0.07 sec.

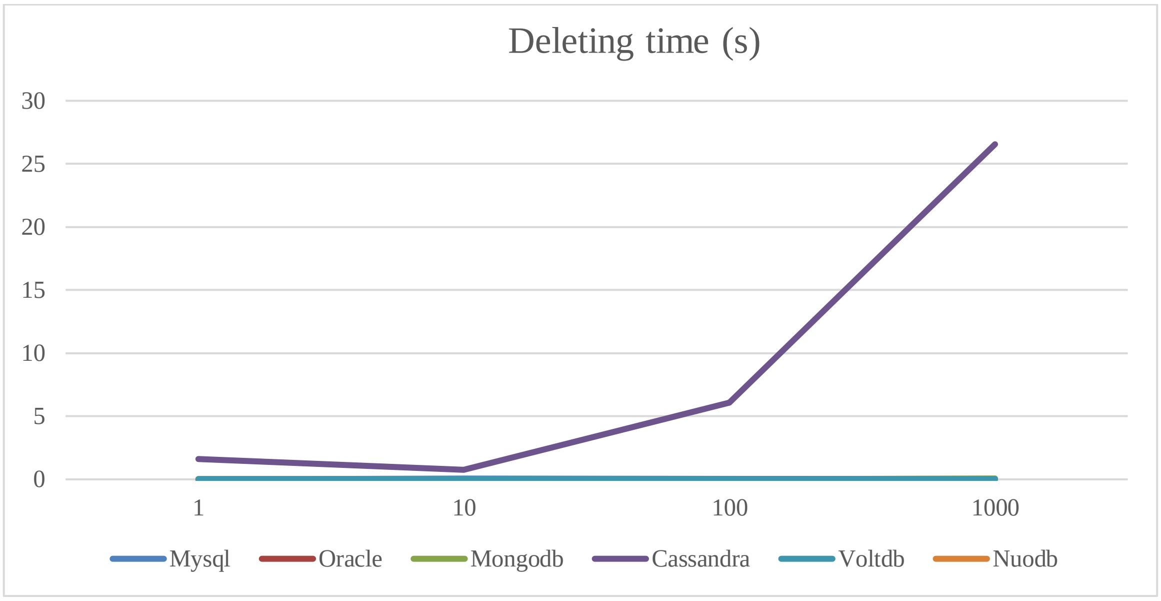


Figure 62 : curve represent the execution time of Deleting 1,10,100,1000 user(with Cassandra)

From figure 10, we can observe that Cassandra performs the worst overall runtime with 26.556 sec.

6. Conclusion & Overall evaluation :

After reading the results obtained by all this systems, sever-al lessons can be learned:

- Oracle showed his performance in insert, update and delete operations. Because he supports Data Partitioning and is best suited for large-scale data. It is also very flexible in both static and dynamic environments and executes queries at a faster rate in both environments
- MongoDB, use volatile memory, to store and retrieve data, allowing this way lower execution times
- Cassandra shows slower performance. The main reason for this difference is the way records are kept on disk, rather than on memory
- Cassandra presented more difficulty on update operations. This is due mainly to its lack of optimization to run this kind of operations.
- VoltDB showed his performance because he uses in-memory storage to maximize throughput, avoiding costly disk access. Further performance gains are achieved by serializing all data access, avoiding many of the time-consuming functions

General Conclusion

This end-of-studies project consists of studying, different databases SQL, NOsql and Newsql and makes a theoretical and practical comparative survey. It was an opportunity for me to develop and implement my personal skills and enrich my knowledge where I was able to discover new concepts in the field of databases.

These experimental results of the different tests that have been donning made it possible to compare the different types of databases Sql, Nosql, Newsql.

After doing several experiments, we can conclude that the results are different according to the work that needs to be done.

Therefore, the selection criteria must depend on the needs of the application and the nature of the operations carried out on the data.

As for the solutions that have been studied, there are good solutions to do the insert operations, updates and others to do the delete operations:

- For insertion operation : it is necessary to move towards oracle is the best followed by Mongodb and Cassandra.
- For update operation : it is better to choose one of this systems Oracle, Mongodb, Mysql, Voltdb.
- For delete operation : Oracle, Voltdb, Mongodb, Mysql. For Cassandra a lot of effort remains to be provided by designers to improve their performance.

As related future work research, we will pass to higher scales by increasing the number of operations performed to achieve million operations and the number of records inserted to have a big test database.

We also plan to extend the comparative study to other popular NoSQL and NewSQL systems as well as SQL relational systems.

REFERENCES

[1] Don Chamberlin : “SQL”, IBM Almaden Research Center, San Jose, CA September 2012

[2] Sneha Binani, Ajinkya Gutti, Shivam Upadhyay: “SQL vs. NoSQL vs. NewSQL- A Comparative Study, October 2016 – www.caeaccess.org

[3] https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable

[4] https://databasetown.com/relational-database-benefits-and-limitations/

[5] Li, Z. (2018). NoSQL Databases. The Geographic Information Science & Technology Body of Knowledge (2nd Quarter 2018 Edition), John P. Wilson (Ed).

[6] Leavitt, N. (2010). Will NoSQL databases live up to their promise?. Computer,43(2), 12-14.

[7] https://www.lifewire.com/abandoning-acid-in-favor-of-base-1019674

[8] https://technologypoint.in/advantages-and-disadvantages-of-nosql-databases/

[9] Khasawneh, Tariq N.; AL-Sahlee, Mahmoud H.; Safia, Ali A. (2020): “ - *SQL, NewSQL, and NOSQL Databases: A Comparative Survey. , [IEEE 2020 11th International Conference on Information and Communication Systems (ICICS) - Irbid, Jordan (2020.4.7-2020.4.9)] 2020 11th International Conference on Information and Communication Systems (ICICS) (), 013–021*

[10] A B M Moniruzzaman: "NewSQL: Towards NextGeneration Scalable RDBMS for Online Transaction Processing (OLTP) for Big Data Management", November 2014.

[11] Jenny Richards, Advantages and Disadvantages of NoSQL databases – what you should know, Hadoop360, September 24, 2015, http://www.hadoop360.com/blog/advantages-anddisadvantages-of-nosql-databases-what-you-should-k

[12] http://www.sneakerala.com/ldce/IT/IT1/5-Oracle.pdf

[13] "Cassandra is an Apache top level project" (http://www.mail-archive.com/cassandra-dev@incubator.apache.org/msg01518.html).Mail-archive.com.2010-02-18.Archived

[14] http://www.odbms.org/wp-content/uploads/2013/11/VoltDBTechnicalOverview.pdf

[15] https://blog.sqlauthority.com/2017/07/18/nuodb-3-important-features-elastic-sql-database-every-one-must-know/

[16] https://www.apachefriends.org/fr/download.html

[17] https://www.oracle.com/me/database/technologies/xe-downloads.html

[18] https://www.codeflow.site/fr/article/mongodb__how-to-install-mongodb-on-windows

[19] https://www.python.org/downloads/release/python-2715/

[20] http://cassandra.apache.org

[21] Ngoc Huyen Nguyen: "An application-oriented comparison of two NoSQL database systems: MongoDB and VoltDB"; thesis, Hamburg University of applied sciences; August 2016.
https://reposit.hawhamburg.de/bitstream/20.500.12738/7750/1/bachelor_thesis.pdf