ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Πτυχιακή εργασία

Συγκριτική αξιολόγηση NoSQL και παραδοσιακών βάσεων δεδομένων

Βασίλειος Ευσταθίου
2022201500041

Δημήτριος Παλαιολόγος
2022201500116

Επιβλέπουσα:

Παρασκευή Ραυτοπούλου
Ε.ΔΙ.Π.

Τρίπολη, Μάιος 2022

Περίληψη

Σε αυτήν την πτυχιακή εργασία θα εξετάσουμε και θα αξιολογήσουμε διάφορους τύπους NoSQL βάσεων δεδομένων καθώς επίσης και θα τις συγκρίνουμε με τις πιο παραδοσιακές σχεσιακές βάσεις. Ποια είναι τα κριτήρια επιλογής του καλύτερου συστήματος διαχείρισης βάσης δεδομένων; Στην βιβλιογραφική ανασκόπηση θα γίνει μία σύντομη παρουσίαση των παραδοσιακών συστημάτων διαχείρισης βάσεων δεδομένων καθώς και των NoSQL συστημάτων (Document databases, Key-value stores, Column-oriented databases, Graph databases). Επίσης θα εξετάσουμε τα βασικά θεωρήματα CAP και ACID που διέπουν τις βάσεις δεδομένων. Τελικά στην παρούσα πτυχιακή εργασία θα συγκρίνουμε τους 4 τύπους NoSQL βάσεων με μία παραδοσιακή βάση SQL. Θα αναλυθούν λεπτομερώς τα χαρακτηριστικά των 4 NoSQL βάσεων που εμείς επιλέξαμε και θα δημιουργηθούν 4 διαφορετικά συστήματα διαφορετικής κατηγορίας καθώς και ένα παραδοσιακό σύστημα SQL. Θα συζητηθούν με λεπτομέρεια τα κριτήρια επιλογής του κάθε συστήματος καθώς και τα κριτήρια σύγκρισής τους. Χαρακτηριστικά τέτοια θα μπορούσαν να είναι η αρχιτεκτονική του συστήματος, η επεκτασιμότητα του, η ταχύτητα αποτίμησης μιας σύνθετης ερώτησης, η ασφάλειά του κλπ. Τέλος θα συμπεράνουμε αν η μετάβαση απο τα παραδοσιακά SQL συστήματα στα νέα NoSQL είναι σημαντική και αξίζει να δοκιμαστεί.

Abstract

In this dissertation we will examine and evaluate different types of NoSQL databases as well as we will compare them with the more traditional relational databases. What are the criteria for selecting the best database management system? In the literature review there will be a brief presentation of traditional database management systems as well as NoSQL systems (Document databases, Key-value stores, Column-oriented databases, Graph databases). We will also look at the basic CAP and ACID theorems that govern databases. Finally, in this dissertation we will compare the 4 types of NoSQL databases with a traditional SQL database. The characteristics of the 4 NoSQL databases we have chosen will be analyzed in detail and 4 different systems of different categories will be created as well as a traditional SQL system. The selection criteria of each system as well as their comparison criteria will be discussed in detail. Features such as its system architecture, its scalability, the speed of evaluating a complex query, its security, etc. Finally we will conclude whether the transition from traditional SQL systems to the new NoSQL ones is important and worth trying.

Αφιερώνεται σε όσους μας βοήθησαν να βγάλουμε ένα πανεπιστήμιο σε αυτούς τους δύσκολους καιρούς.

Contents

Preface

Database means a collection of systematically formatted related data in which it is possible to retrieve data through on-demand search. Database systems are now a very important part of many businesses. They are used to maintain internal documents, present data to clients and users of the World Wide Web and support many other professional activities. The term ”databases” refers to organized, distinct collections of related data stored electronically and digitally, the software that considers such collections (Database Management System or DBMS) and the field that studies it. A database is essentially a collection of data properly stored on a computer system so that it can be also accessed and updated by the person who manages it. Databases as we will also see below are managed by different types of systems.

Traditionally structured relational databases are directly linked to the SQL (Structured Query Language) programming language. This language is used to access structured relational databases. Some of the most famous systems of 2021 that support SQL are MySQL, PostgreSQL and Microsoft SQL Server. Such systems support applications such as Netflix, Facebook, Instagram, Airbnb, Amazon, as well as banking and analytics companies. We will examine in depth the relational data model in the literature review.

For decades, the relational data model has been the dominant data model used for application development. Relational databases such as Oracle, DB2, SQL Server, MySQL and PostgreSQL remain up to this day dominant systems in data management. The development of newer, revolutionary systems began in the mid-2000s. To distinguish these new categories of databases and database models, the term ”NoSQL” was coined. The terms ”NoSQL” and ”non-relational” are often used interchangeably. NoSQL databases are widely recognized for their ease of development, functionality and performance for Big Data. Relational databases find it difficult to support the creation of modern applications as they greatly limit the design of a database due to the complexity and limited scalability of database objects and relationships. Some of the most popular NoSQL databases in 2021 are MongoDB, Redis, Apache HBase, DynamoDB and Cassandra. Also many NoSQL systems have been used in the design of famous applications such as Instagram, Facebook, Amazon, Twitter, Google applications and LinkedIn.

It is well known that many of the popular applications we know and use, often combine different types of databases according to their requirements. For example, Instagram mainly uses two backend database systems, PostgreSQL and Cassandra. Both PostgreSQL and Cassandra have mature replication frameworks that work well as a globally consistent data store.

But what are the characteristics of selecting the most appropriate database management system in a modern application? How are traditional systems different from modern NoSQL ones? Is it worth switching and upgrading from old relational systems to new NoSQL ones? In this dissertation, we will compare the different types of data storage systems (SQL and NoSQL systems) and we will examine in depth their advantages and disadvantages.

Chapter 1

Introduction

In this chapter we will give a brief overview of the basic theory of traditional SQL data management systems as well as NoSQL technologies and systems.

1.1 Database Management System (DBMS)

Formally, a *database* refers to a set of related data and the way it is organized. Access to this data is usually provided by a *database management system* (DBMS) consisting of an integrated set of computer software that allows users to interact with one or more databases and provides access to all of the data contained in the database (although restrictions may exist that limit access to particular data). The DBMS provides various functions that allow entry, storage and retrieval of large quantities of information and provides ways to manage how that information is organized.[1]

Existing DBMSs which provide various functions that allow management of a database and its data which can be classified into four main functional groups as follows:

- Data definition: Creation, modification and removal of definitions that define the organization of the data.
- Update: Insertion, modification and deletion of the actual data.[2]
- Retrieval: Providing information in a form directly usable or for further processing by other applications. The retrieved data may be made available in a form basically the same as it is stored in the database or in a new form obtained by altering or combining existing data from the database. [3]
- Administration: Registering and monitoring users, enforcing data security, monitoring performance, maintaining data integrity, dealing with concurrency control, and recovering information that has been corrupted by some event such as an unexpected system failure. [4]

Both a database and its DBMS conform to the principles of a particula database model.[5] *Database management system* refers collectively to the database model, database management system, and database. [6]

Physically, database servers are dedicated computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers, with generous memory and RAID disk arrays used for stable storage. Hardware database accelerators, connected to one or more servers via a high-speed channel, are also used in large volume transaction

processing environments. DBMSs are found at the heart of most database applications. DBMSs may be built around a custom multitasking kernel with built-in networking support, but modern DBMSs typically rely on a standard operating system to provide these functions.

Since DBMSs comprise a significant market, computer and storage vendors often take into account DBMS requirements in their own development plans.[7]

Databases and DBMSs can be categorized according to the database model(s) that they support (such as relational or XML), the type(s) of computer they run on (from a server cluster to a mobile phone), the query language(s) used to access the database (such as SQL or XQuery), and their internal engineering, which affects performance, scalability, resilience, and security.

1.2 Relational Databases

The relational model (RM) for database management is an approach to managing data using a structure and language consistent with first-order predicate logic, [8][9], where all data is represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database.

Most relational databases use the SQL data definition and query language, these systems implement what can be regarded as an engineering approximation to the relational model. A table in an SQL database schema corresponds to a predicate variable, the contents of a table to a relation, key constraints, other constraints, and SQL queries correspond to predicates.[10]

The relational model's central idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values. The content of the database at any given time is a finite (logical) model of the database, i.e. a set of relations, one per predicate variable, such that all predicates are satisfied. A request for information from the database (a database query) is also a predicate.[11]

A data type as used in a typical relational database might be the set of integers, the set of character strings, the set of dates, or the two boolean values true and false, and so on. Nowadays provisions are expected to be available for user-defined types in addition to the built-in ones provided by the system.

Attribute is the term used in the theory for what is commonly referred to as a column. Similarly, table is commonly used in place of the theoretical term. A table data structure is specified as a list of column definitions, each of which specifies a unique column name and the type of the values that are permitted for that column.

A tuple is basically the same thing as a row, except in an SQL DBMS, where the column values in a row are ordered. Tuples are not ordered. Instead, each attribute value is identified solely by the attribute name and never by its ordinal position within the tuple.

A relation is a table structure definition (a set of column definitions) along with the data appearing in that structure as we can see in Figure 1.1. A relationship between two database tables presupposes that one of them has a foreign key that references to the primary key of another table. We can see the function of database relationship in Figure 1.2. The structure definition is the heading and the data appearing in it is the body, a set of rows. A database relvar (relation variable) is commonly known as a base table. The heading and body of the table resulting from evaluation of some query are determined by the definitions of the operators used in the expression of that query. Note that in SQL the heading is not always a set of column definitions as described above, because it is possible for a column to have no name and also for two or more columns to have the same

name. Also, the body is not always a set of rows because in SQL it is possible for the same row to appear more than once in the same body.[12]

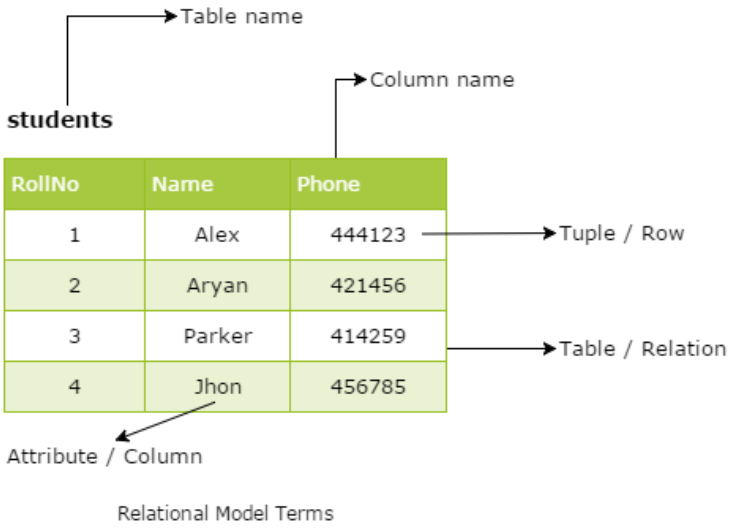


Figure 1.1: Example of a table (relation).

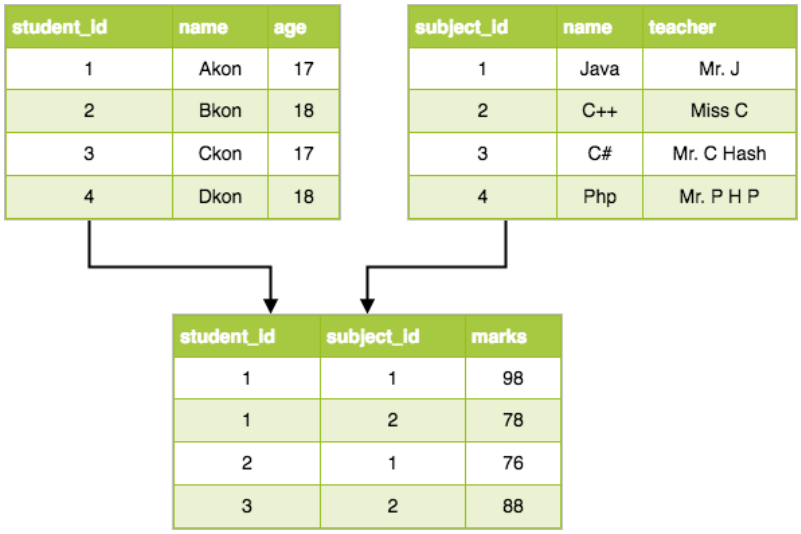


Figure 1.2: The relationship between the *students* table and the *subjects* table is the `student_id` column and the `subject_id` column.

1.3 ACID

In computer science, 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 (Figure 1.3). In the context of databases, a sequence of database operations that satisfies the ACID properties (which can be perceived as a single logical operation on the data) is called

a transaction. For example, a transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction.[13]

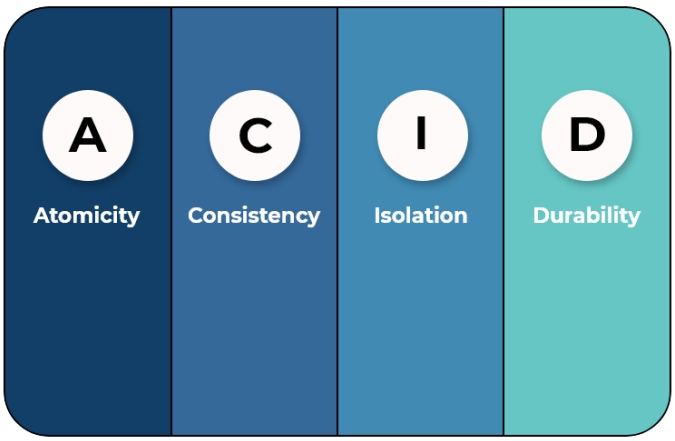


Figure 1.3: ACID properties.

- Atomicity: Transactions are often composed of multiple statements. Atomicity guarantees that each transaction is treated as a single *unit*, which either succeeds completely, or fails completely. An atomic system must guarantee atomicity in each and every situation, including power failures, errors and crashes.[14] A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright. As a consequence, the transaction cannot be observed to be in progress by another database client.[15]

- Consistency: Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants: any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This prevents database corruption by an illegal transaction, but does not guarantee that a transaction is correct. Referential integrity guarantees the primary key – foreign key relationship.[16]

- Isolation: Transactions are often executed concurrently (e.g. multiple transactions reading and writing to a table at the same time). Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. Isolation is the main goal of concurrency control; depending on the method used, the effects of an incomplete transaction might not even be visible to other transactions.[17]

- Durability: After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure. For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.[18]

1.4 Non Relational Databases

Non relational database is a class of systems which manage databases and it broadly differs from the relational systems in many significant ways, most importantly by the fact that it doesn't use relations (tables) as its storage structure. A NoSQL (originally referring to *non-SQL* or *non-relational*)[19] database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Such databases have existed since the late 1960s, but the name *NoSQL* was only coined in the early 21st century,[20] triggered by the needs of Web 2.0 companies.[21] NoSQL databases are increasingly used in big data and real-time web applications.[22] NoSQL systems are also sometimes called *Not only SQL* to emphasize that they may support SQL-like query languages or sit alongside SQL databases in polyglot-persistent architectures.[23]

Motivations for this approach include simplicity of design, simpler *horizontal* scaling to clusters of machines (which is a problem for relational databases.[24] The data structures used by NoSQL databases (key–value pair, wide column, graph or document as we can see in figure 1.4) are different from those used by default in relational databases, making some operations faster in NoSQL. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as *more flexible* than relational database tables.[25] Non relational databases may primarily be classified on the basis of way of organizing data as follows on the next subchapters.

1.4.1 Document Databases

Document Store, also commonly known as *Document Oriented Database*, is basically a computer program used for storing, retrieving, updating data stored in database. The underlying storage structure used in such databases is a 'document'. Each Document Store differs in its implementation of data. However each of it assumes that data is enclosed and encoded in some standard format which may be XML, BSON, PDF or Microsoft office. Each document is represented by a unique key which is a string (URI or path). An API or a query language is provided for fast retrieval of documents on the basis of its content. For example a query that retrieves all the documents in which certain field is set to some particular value.

Different implementations offer different ways of organizing and/or grouping documents:

- Collections
- Tags
- Non-visible metadata
- Directory hierarchies

Compared to relational databases, collections could be considered analogous to tables and documents analogous to records. But they are different in the sense that every record in a table has the same sequence of fields, while documents in a collection may have fields that are completely different.

1.4.2 Graph Databases

Graph databases are schema-less databases which use graph data structures along with nodes, edges and certain properties to represent data. Nodes may represent entities like people, business or

any other item similar to what objects represent in any programming language. Properties designate any pertinent information related to nodes. On the other hand, edges relate a node to other node or a node to some property. One can obtain some meaningful pattern or behavior after studying the interconnection between all nodes, properties and edges. Graph databases portray the data as it is viewed conceptually. This is accomplished by transferring the data into nodes and its relationships into edges. It consists of a set of objects, which can be a node or an edge.

- Nodes represent entities or instances such as people, businesses, accounts, or any other item to be tracked. They are roughly the equivalent of a record, relation, or row in a relational database, or a document in a document-store database.
- Edges, also termed graphs or relationships, are the lines that connect nodes to other nodes, representing the relationship between them. Meaningful patterns emerge when examining the connections and interconnections of nodes, properties and edges. The edges can either be directed or undirected. In an undirected graph, an edge connecting two nodes has a single meaning. In a directed graph, the edges connecting two different nodes have different meanings, depending on their direction. Edges are the key concept in graph databases, representing an abstraction that is not directly implemented in a relational model or a document-store model.
- Properties are information associated to nodes.

1.4.3 Key-Value Stores

Key value stores allow the application developer to store schema-less data. This data consists of a key which is represented by a string and the actual data which is the value in key-value pair. The data can be any primitive of programming language, which may be a string, an integer or an array, or it can be an object. Thus a key value store loosens the requirement of formatted data for storage, eliminating the need for a fixed data model.

1.4.4 Column-Oriented Databases

Column Store Databases, unlike Row Databases, store their data in the form of columns. It serializes all the values of one column together and so on. Column-oriented databases are comparatively efficient than row oriented one's when new values for a column are entered for all rows at once as column data can be written efficiently and replace old data without altering any other columns for the rows.

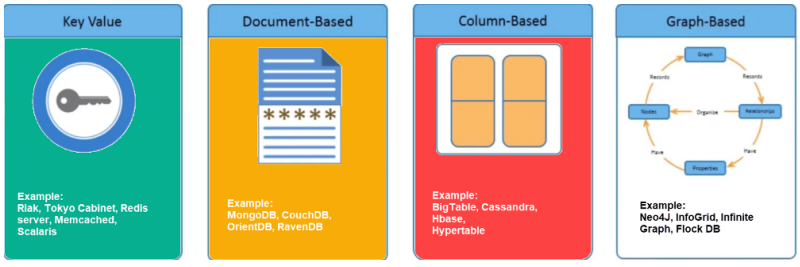


Figure 1.4: Some examples of the most famous NoSQL databases per NoSQL database system category.

1.5 CAP Theorem

A distributed database has three very desirable properties:

- Partition Tolerance
- Consistency
- Availability

The CAP theorem [26] states that one can have at most two of these properties for any shared-data system (Figure 1.5). Theoretically there are the three options that follow.

- Forfeit Partition Tolerance: The system does not have a defined behavior in case of a network partition. Brewer names 2-Phase-Commit (2PC) as a trait of this option, although 2PC supports temporarily partitions (node crashes, lost messages) by waiting until all messages are received.
- Forfeit Consistency: In case of partition data can still be used, but since the nodes cannot communicate with each other there is no guarantee that the data is consistent. It implies optimistic locking and inconsistency resolving protocols.
- Forfeit Availability: Data can only be used if its consistency is guaranteed. This implies pessimistic locking, since we need to lock any updated object until the update has been propagated to all nodes. In case of a network partition it might take quite long until the database is in a consistent state again, thus we cannot guarantee high availability anymore.

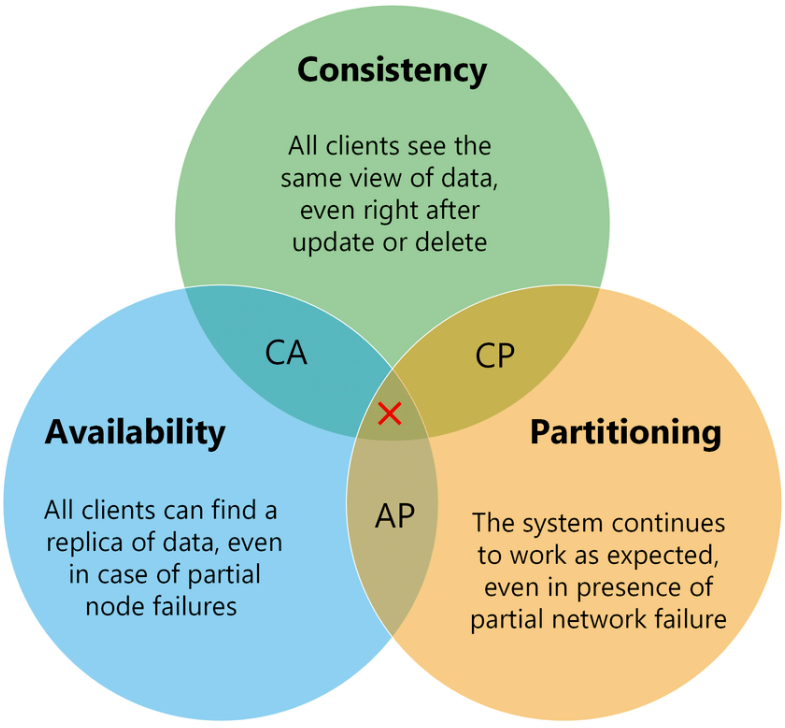


Figure 1.5: Visualization of CAP theorem.

With this definition, the theorem was proven by contradiction. Assume all three criteria (atomicity, availability and partition tolerance) are fulfilled. Since any network with at least two nodes can be divided into two disjoint, non-empty sets G1,G2, we define our network as such. An atomic object o has the initial value n0. We define a1 as part of an execution consisting of a single write on the atomic object to a value n1≠n0 in G1. Assume a1 is the only client request during that time. Further, assume that no messages from G1 are received in G2, and vice versa. Because of the availability requirement we know that a1 will complete, meaning that the object o now has value n1 in G1. Similarly a2 is part of an execution consisting of a single read of o in G2. During a2 again no messages from G2 are received in G1 and vice versa. Due to the availability requirement we know that a2 will complete. If we start an execution consisting of a1 and a2, G2 only sees a2 (since it does not receive any messages or requests concerning a1. Therefore the read request from a2 still must return the value n0. But since the read request starts only after the write request ended, the atomicity requirement is violated, which proves that we cannot guarantee all three requirements at the same time.[27]

1.6 BASE Model

The meaning of the BASE model is Basically Available (1.6), which mainly includes the following three aspects:

- Basically Available
- Soft-State
- Eventual Consistency

Figure 1.6: BASE properties.

BASE model, which emphasizes greater scalability and provides a flexible schema, offers better performance, mostly open source, cheap but, lacks a standard query language and does not provide adequate security mechanisms. Both databases will continue to exist alongside each other with none being better than the other. The choice of the database to use will depend on the nature of the application being developed.[28]

In practical applications, the user usually has high expectations on transactions. For example, the data cannot be lost, the transaction cannot be inconsistent, and the system must be available. But the system that can meet all user requirements completely does not exist. The user has to grasp an understanding of the most essential system requirements before choosing a proper database.[29]

Chapter 2

Relational vs. NoSQL Databases: Literature Review

In this chapter we will get a taste of the basic features of database systems and we will explore the key differences between traditional SQL data management systems and NoSQL technologies and systems.

2.1 Development History

Relational databases accessed with SQL were developed in the 1970s with a focus on reducing data duplication as storage was much more costly than developer time.[30] NoSQL databases were developed in the late 2000s with a focus on scaling, fast queries, allowing for frequent application changes, and making programming simpler for developers.

2.2 Data Storage Model

Concerning the data storage model, relational databases organize data as a series of two-dimensional tables with rows and columns. Most vendors provide a dialect of the Structured Query Language (SQL) for retrieving and managing data. An RDBMS typically implements a transactionally consistent mechanism that conforms to the ACID (Atomic, Consistent, Isolated, Durable) model for updating information.[30]

On the other hand, NoSQL databases have the following properties when comes to the data storage.[31]

- A document database stores a collection of documents, where each document consists of named fields and data. The data can be simple values or complex elements such as lists and child collections. Documents are retrieved by unique keys. Typically, a document contains the data for single entity, such as a customer or an order. A document may contain information that would be spread across several relational tables in an RDBMS. Documents do not need to have the same structure. Applications can store different data in documents as business requirements change.

- A graph database stores two types of information, nodes and edges. Edges specify relationships between nodes. Nodes and edges can have properties that provide information about

that node or edge, similar to columns in a table. Edges can also have a direction indicating the nature of the relationship.

- A key/value store associates each data value with a unique key. Most key/value stores only support simple query, insert, and delete operations. To modify a value (either partially or completely), an application must overwrite the existing data for the entire value. In most implementations, reading or writing a single value is an atomic operation.

- A column-family database organizes data into rows and columns. In its simplest form, a column-family database can appear very similar to a relational database, at least conceptually. The real power of a column-family database lies in its denormalized approach to structuring sparse data. We can think of a column-family database as holding tabular data with rows and columns, but the columns are divided into groups known as column families. Each column family holds a set of columns that are logically related together and are typically retrieved or manipulated as a unit. Other data that is accessed separately can be stored in separate column families. Within a column family, new columns can be added dynamically, and rows can be sparse.

2.3 Schema

A database schema is an abstract design that represents the storage of our data in a database system. It describes the organization of data and the relationships between tables in a given database. Developers plan a database schema in advance so they know what components are necessary and how they will connect to each other. A database schema is a blueprint or architecture of how our data will look.[32] Schema doesn't hold data itself, but instead describes the shape of the data and how it might relate to other tables or models. A database schema will include all important or relevant data, consistent formatting for all data entries and unique keys for all entries and database objects. Each column in a table has a name and data type.

Concerning SQL databases [33] they use structured query language and have a pre-defined schema for defining and manipulating data. SQL is one of the most versatile and widely used query languages available, making it a safe choice for many use cases. It's perfect for complex queries. However, SQL can be too restrictive. We have to use predefined schemas to determine our data structure before we can work with it. All of our data must follow the same structure. This process requires significant upfront preparation. If we ever wanted to change our data structure, it would be difficult and disruptive to our whole system.

On the other hand the schema of NoSQL is not fixed, but it is flexible.[34] It uses varied interfaces to store and analyse sheer volume of usergenerated content, personal data and spatial data being generated by modern applications, cloud computing and smart devices. In this context, NoSQL database presents a preferred solution than SQL database primarily for its ability to cater to the horizontal partitioning of data, flexible data processing and improved performance. Different data can be stored together as required. It also allows modification of the schema freely with no or minimum downtime.

2.4 Scaling

Scalability is the ability to expand or contract the capacity of system resources in order to support the changing usage of our application.[35] This can refer both to increasing and decreasing usage of the application. Increased usage of our application brings three main challenges to our database server.

- The CPU and/or memory becomes overloaded, and the database server either cannot respond to all the request throughput or do so in a reasonable amount of time.
- Our database server runs out of storage, and thus cannot store all the data.
- Our network interface is overloaded, so it cannot support all the network traffic received.

However, there comes a point when system resource limits are reached. At this point, we want to consider scaling our database vertically, horizontally, or both (Figure 2.1).

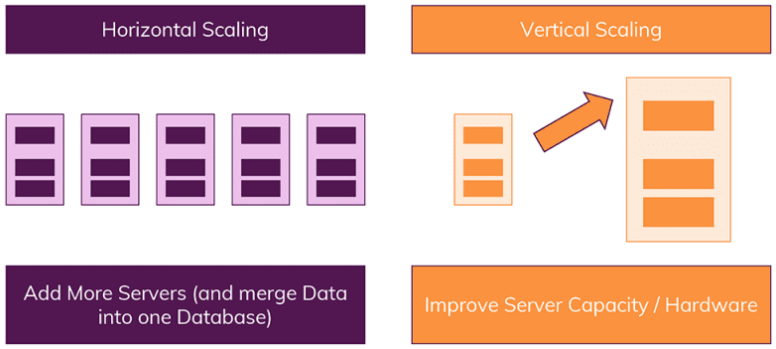


Figure 2.1: Horizontal and Vertical Scaling.

Most SQL databases are vertically scalable. *Vertical scaling* refers to increasing the processing power of a single server or cluster. Both relational and non-relational databases can scale up, but eventually, there will be a limit in terms of maximum processing power and throughput. Additionally, there are increased costs with high-performance hardware, as costs do not scale linearly. The advantages and the disadvantages of the vertical scaling are presented below.

- Advantages: The main benefit of vertical scaling is that nothing changes about our database infrastructure other than the hardware specifications of the machine running the database. As such, it's transparent to the application. The only difference is that we have more CPUs, memory, and/or storage space.Vertical scaling is a good option to try first if massive storage and processing are not required.
- Disadvantages: The downside of scaling up is that servers with more storage and processing power can be a lot more expensive.There is also a physical limit on the amount of CPUs, memory, network interfaces, and hard-drives that can be used on a single machine. For those scaling up using a cloud platform provider, we will eventually reach the highest tier of machine available. If scaling vertically requires a migration between hardwares, it could result in downtime or service disruption. When cost and/or machine limitations become an issue, be sure to consider horizontal scaling.

Horizontal scaling, used mainly by NoSQL databases, also known as scale-out, refers to bringing on additional nodes to share the load. This is difficult with relational databases due to the difficulty in spreading out related data across nodes. With non-relational databases, this is made simpler since collections are self-contained and not coupled relationally. Horizontal scaling allows NoSQL databases to be distributed across nodes more simply, as queries do not have to *join* them together across nodes.[36] In what follows, we give the main advantages and disadvantages of this approach.

- Advantages: The main benefit of horizontal scaling is that the database uses an automated server increase to match usage. Furthermore, these database systems have low to zero down-time needed for server upgrades, while they are resilient to random hardware failures.
- Disadvantages: A big disadvantage of these kind of database systems is that data consistency can be challenging across multiple machines (joins require cross-server communication). That means that costs may be higher, and more coding processes may be required. In addition, servers may still encounter hardware limit issues if machines are small-sized.

2.5 ACID and BASE Model

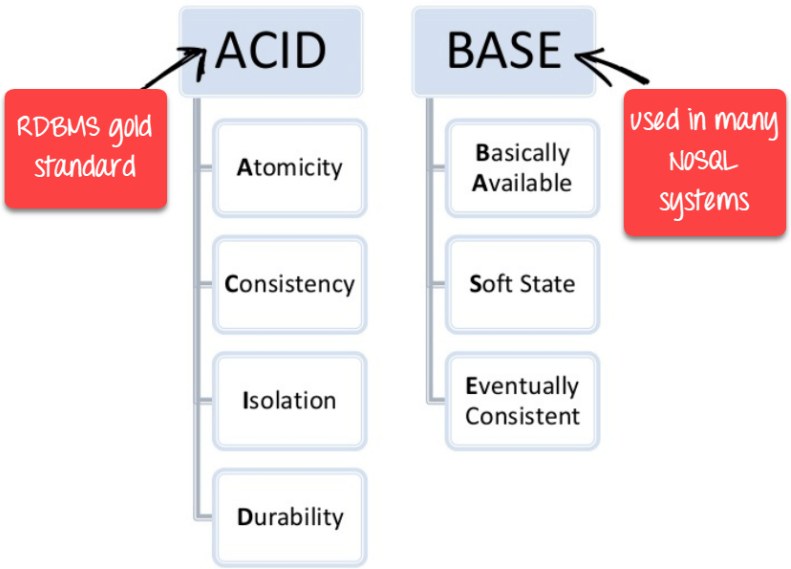


Figure 2.2: ACID and BASE properties

The presence of four properties ACID properties (atomicity, consistency, isolation and durability) can ensure that a database transaction is completed in a timely manner.[37] Most modern SQL DBs use transactional standards like ACID to ensure data integrity and keep our users from seeing wrong or stale data, and this post explores how they work.[38]

Base (Basically Available, Soft state, Eventually Consistent) is a model of many NoSQL systems. ACIDity is not always necessary, there are many scenarios where it is sufficient to promise a weaker set of guarantees.[39] These are known by the amusing backronym BASE, or *Basically Available, Soft State, Eventual Consistency* as we saw earlier. While these are alternatives to ACID, the words *available* and *consistency* refer to the same properties as the CAP theorem, which lets we know these guarantees apply specifically to distributed databases (2.2).

2.6 CAP on SQL and NoSQL

CAP theorem or Eric Brewers theorem states that we can only achieve at most two out of three guarantees for a database: Consistency, Availability and Partition Tolerance.[40] Here Consistency means that all nodes in the network see the same data at the same time.

A relational database can have downtime or be unavailable but it is always CAP-Available. We should note that consistency as defined in the CAP theorem is quite different from the consistency guaranteed in ACID database transactions.

Today, NoSQL databases are classified based on the two CAP characteristics they support. A CA (Consistency and Availability) database delivers consistency and availability across all nodes. It can't do this if there is a partition between any two nodes in the system, however, and therefore can't deliver fault tolerance. In a distributed system, partitions can not be avoided. So, while we can discuss a CA distributed database in theory, for all practical purposes, a CA distributed database can't exist. However, this doesn not mean we can not have a CA database for our distributed application if we need one. Many relational databases, such as PostgreSQL, deliver consistency and availability and can be deployed to multiple nodes using replication.[41]

2.7 SQL and NoSQL Performance

SQL relational database management systems are designed for fast transactions updating multiple rows across tables with complex integrity constraints.[42] SQL queries are expressive and declarative. We can focus on what a transaction should accomplish. RDBMS will figure out how to do it, since it will optimize our query using relational algebra and find the best execution plan.

NoSQL datastores are designed for efficiently handling a lot more data than RDBMS. There are no relational constraints on the data, and it does not need to be even tabular. NoSQL offers performance at a higher scale by typically giving up strong consistency. Data access is mostly processed through REST APIs (architectural styles for application program interfaces that use HTTP requests to access and use data). Although we should notice that NoSQL query languages (such as GraphQL) are not yet as mature as SQL in design and optimizations.

2.8 Famous SQL and NoSQL Databases

In this part we will present to we some of the most famous SQL and NoSQL database management systems according to statistic websites. We have not been able to find official data and sources based on the popularity of usage of SQL and NoSQL database management systems. The following databases are good approaches of usage popularity per database category according to the most statistic sites on web.

These are the most used SQL database management systems.[43]

- MySQL

Figure 2.3: MySQL logo.

MySQL is a free and open-source relational database management system (Figure 2.3). It is an extremely established database with a huge community, extensive testing, lots of stability and it is available for all major platforms Furthermore, replication and sharding (Sharding is a method for distributing a single dataset across multiple databases, which can then be stored on multiple machines) are available and it covers a wide range of use cases.

- Oracle

Figure 2.4: Oracle logo.

Oracle is a commercial database with frequent updates, professional management, and excellent customer support (Figure 2.4). It uses PL/SQL (procedural language/SQL) as SQL dialect. It is also one of the most expensive database solutions, while it works with huge databases. Furthermore Oracle uses simple upgrades, transaction control, it is compatible with all operating systems and it is suitable for enterprises and organizations with demanding workloads.

- Microsoft SQL Server

Figure 2.5: Microsoft SQL Server logo.

Microsoft SQL Server is a Commercial database developed and managed by Microsoft (Figure 2.5). It uses transact SQL, or T-SQL as its SQL dialect. Microsoft SQL Server works only with Windows and Linux and it makes it difficult to make adjustments in mid-process when errors are accured. Furthermore, Microsoft SQL Server provides excellent documentation. It also works well for small to medium-sized organizations that want a commercial database solution without the cost of Oracle.

- PostgreSQL

Figure 2.6: PostgreSQL logo.

PostgreSQL is an object-oriented database management system, which means it is a hybrid SQL/NoSQL database solution (Figure 2.6). It is basically free and open-source, while it delivers compatibility with a wide range of operating systems. It also has active community and many third-party service providers. PostgreSQL has high ACID compliance and it uses pure SQL. It works best for use cases where our data does not fit in with a relational model. It also works well for extra-large databases and running complicated queries.

As for the NoSQL database management systems we managed to include a single famous NoSQL system per category (NoSQL database systems categories are provided in Section 1.4).[44]

- Neo4J (Graph database)

Figure 2.7: Neo4j logo.

Neo4j is a graph NoSQL database system which supports ACID transactions and it is suitable when there is a need to store and query data about connections between related data, such as social network contexts (Figure 2.7). Neo4j is fast for simple queries, but also efficient when answering questions about connections between data. It also excels when profiling data and determining clusters of related objects.

- Apache HBase (Wide column store)

Figure 2.8: Apache HBase logo.

Apache HBase is a column-oriented NoSQL database system which guarantees atomicity at the row level but it does not support ACID transactions (Figure 2.8). It can be used for extremely large sets of data, where querying patterns are predictable, often for supporting aggregation and analytics. Apache Hbase is also excellent when aggregating values on particular columns, with strong compression and in-memory support.

- MongoDB (Document database)

Figure 2.9: MongoDB logo.

MongoDB is a document database system which supports ACID transactions (Figure 2.9). It is best used when data is modeled by a set of interrelated objects, with its flexibility toward data structure making it a good general purpose database because documents can contain nested structures for capturing complex data. MongoDB also provides fast retrieval of individual documents, much like key-value stores, but can also support complex queries with aggregation pipelines.

- Redis (Key-value store)

Figure 2.10: Redis logo.

Redis is a key-value store type database system which supports ACID transactions (Figure 2.10). It is excellent for frequent high-speed access to the same chunks of data, even if those chunks of data are large. Redis is also great for instantaneous retrieval when queries are not complex, useful for caching use cases.

2.9 SQL and NoSQL Pros and Cons

In this section we we present some of the pros and cons of using SQL or NoSQL according to our needs of using data and the suitability for a specific purpose.[30][45]

In what follows, we present the advantages and the disadvantages of SQL databases.

- Pros
  - Flexible queries: Enables support for diverse workloads, abstracts data over underlying implementations, and allows engines to optimize queries to fit on-disk representations.

  - Reduced data storage footprint: Due to normalization and other optimization opportunities, a reduced footprint maximizes database performance and resource usage.
  - Strong and well-understood data integrity semantics: Atomicity, consistency, isolation and durability, (i.e. ACID properties) are database properties that guarantee valid transactions.

- Cons

  - Rigid data models: Requires careful up-front design to ensure adequate performance and resistance to evolution. SQL has a predefined schema, so changing it often includes downtime.
  - Limited horizontal scalability: It is either completely unsupported, supported in an ad-hoc way or only supported on relatively immature technologies.
  - Single point of failure: Non-distributed engines are mitigated by replication and failover techniques.

NoSQL databases on the other hand, have the following advantages and disadvantages.

- Pros

  - Scalable and highly available: Many NoSQL databases are designed to support seamless, online horizontal scalability without significant single points of failure.
  - Flexible data models: Most non-relational systems do not require developers to make up-front commitments to data models. Existing schemas are dynamic, so they can often be changed *on the fly*.
  - Dynamic schema for unstructured data: Documents can be created without a defined structure first, which enables each to have its own unique structure. Syntax varies per database and fields can be added as we build the document.
  - High performance: A limited database functionality range (e.g., by relaxing durability guarantees) enables high performance amongst many NoSQL databases.
  - High-level data abstractions: Beyond the *value in a cell* data model, NoSQL systems provide high-level APIs for powerful data structures. For example, Redis includes a native-sorted set abstraction.
  - Easy for developers: Some NoSQL databases like MongoDB map their data structures to those of popular programming languages. This mapping allows developers to store their data in the same way that they use it in their application code. While it may seem like a trivial advantage, this mapping can allow developers to write less code, leading to faster development time and fewer bugs.

- Cons

  - Vague interpretations of ACID constraints: Despite the widespread belief that it supports NoSQL systems, ACID interpretations can be too broad to make clear determinations about database semantics.
  - Distributed systems have distributed systems problems: Though not specific to NoSQL systems, encountering such problems is common amongst NoSQL developers and may require SME troubleshooting.

– Lack of flexibility in access patterns: Without the abstraction found in relational databases, the on-disk representation of data leaks into the application's queries and leaves no room for NoSQL engines to optimize queries

The following Table 2.1 summarizes the main differences between SQL and NoSQL database management systems as we have recently seen.

	SQL Databases	NoSQL Databases
Data Storage Model	Tables with fixed rows and columns	JSON documents, key-value pairs, tables with rows and dynamic columns, nodes and edges
Schema	Rigid	Flexible
Scaling	Vertical (scale-up with a larger server)	Horizontal (scale-out across commodity servers)
ACID and BASE Model	ACID (Atomicity, Consistency, Isolation and Durability) is a standard for RDBMS	Flexible BASE (Basically Available, Soft State, Eventually Consistent) is a model of many NoSQL systems
CAP	Always CAP-Available	CA distributed database can not exist
Famous Databases	MySQL, Oracle, Microsoft SQL Server, PostgreSQL	MongoDB, Redis, Apache HBase, Neo4j, Cassandra

Table 2.1: Summary of the main differences between SQL and NoSQL DBMS.

Chapter 3

Systems Selection and Evaluation Criteria

In this chapter, we will present the database management systems we are going to test and compare, explaining also the reasons that pushed us towards these specific systems. In addition, we wil discuss the comparison criteria according to which we will evaluate the aforementioned database management systems.

3.1 Selecting Suitable Database Systems

For comparing the different options available in database management systems, we select popular systems, one for each different kind of data store. In this section we point out the reasons why we selected these systems and we present the main characteristics of each system. *Please notice that all systems we selected are free-of-charge.*

3.1.1 SQL Database

Figure 3.1: MySQL logo.

Considering SQL databases we selected to work with MySQL since it is the world's most popular open source database and it provides comprehensive support for every application development need. Many of the world's largest and fastest-growing organizations including Facebook,

Twitter, Booking.com and Verizon rely on MySQL to save time and money powering their high-volume Web sites, business-critical systems and packaged software. Most importantly MySQL is an open-source relational database management system which represents all the possibilities of a traditional relational database. MySQL also uses the SQL language to query the database. In Figure 3.1 we can find the logo of MySQL relational database management system.

3.1.2 NoSQL Databases

Figure 3.2: Neo4j logo.

Considering NoSQL databases we selected to work with Neo4j as our default graph database management system, since it is the first and dominant mover in the graph market. The company's goal is to bring graph technology into the mainstream by connecting the community, customers, partners and even competitors as they adopt graph best practices everywhere. It is easy to learn due to its mature UI with intuitive interaction and built-in learning, time-tested training ecosystem and expert-authored books for in-depth learning. Neo4j also uses Cypher, the world's most powerful and productive graph query language, or the native Java API for writing custom special-purpose extensions. In Figure 3.2 we can find the logo of Neo4j NoSQL database management system.

Figure 3.3: Casandra logo.

For our default column-oriented database management system, we selected to work with Cassandra, since it is free to use and it is one of the most efficient and widely used NoSQL database systems. Instead of Apache HBase, we found Cassandra easier to use. One of the key benefits of this system is that it offers highly-available service and no single point of failure. This is key for businesses that can afford to have their system go down or to lose data. It also uses CQL, as a querry language which we found pretty simple to understand. But most importantly, this system meets all the requirements of a column-oriented database. In Figure 3.3 we can find the logo of Cassandra NoSQL database management system.

Figure 3.4: MongoDB logo.

We chose MongoDB as our default document database management system, since it is free to use and it is the most commonly-used document-based database. MongoDB is built on a scale-out architecture that has become popular with developers of all kinds for developing scalable applications with evolving data schemas. As a document database, MongoDB makes it easy for developers to store structured or unstructured data. It uses a JSON-like format to store documents. This format directly maps to native objects in most modern programming languages, making it a natural choice for developers, as they don't need to think about normalizing data. MongoDB can also handle high volume and can scale both vertically or horizontally to accommodate large data loads. In Figure 3.4 we can find the logo of MongoDB NoSQL database management system.

Figure 3.5: Redis logo.

For testing and studying a key-value store database management system, we selected to work with Redis, since it is a great choice for implementing a highly available in-memory cache to decrease data access latency and increase throughput. Redis popularized the idea of a system that can be considered at the same time a store and a cache, using a design where data is always modified and read from the main computer memory, but also stored on disk in a format that is unsuitable for random access of data. At the same time, Redis provides a data model that is very unusual compared to a relational database management system (RDBMS). 5846 companies reportedly use Redis in their tech stacks, including Uber, Airbnb, and Pinterest. We have also considered to use other key-value stores like Amazon DynamoDB and Oracle NoSQL Database, but they were not available for free. So Redis was the only key-value store that satisfied our material abilities. In Figure 3.5 we can find the logo of Redis NoSQL database management system.

3.2 Evaluation Criteria

In this section we will point out the comparison criteria according to which we will evaluate and compare the systems we have chosen earlier.

- Database: A brief description of the historical status and type of database is presented.

- Architecture: A Database Architecture is a representation of DBMS design. It helps to design, develop, implement, and maintain the database management system. A DBMS architecture allows dividing the database system into individual components that can be independently modified, changed, replaced, and altered.
- Scalability: Database scalability is the ability of a database to handle changing demands by adding/removing resources. Vertical database scaling implies that the database system can fully exploit maximally configured systems, including typically multiprocessors with large memories and vast storage capacity. Such systems are relatively simple to administer, but may offer reduced availability. However, any single computer has a maximum configuration. If workloads expand beyond that limit, the choices are either to migrate to a different, still larger system, or to rearchitect the system to achieve horizontal scalability. Horizontal database scaling involves adding more servers to work on a single workload. Most horizontally scalable systems come with functionality compromises. If an application requires more functionality, migration to a vertically scaled system may be preferable.
- Replication (Nodes, Internode Communication): Database replication is the frequent electronic copying of data from a database in one computer or server to a database in another, so that all users share the same level of information. A database cluster is a collection of databases that is managed by a single instance of a running database server. Database nodes are storage nodes that connect to databases and perform different operations on them, such as update, insert, delete, and select. Internode TLS secures communication between nodes within a cluster. It is important to secure communications between nodes if we do not trust the network between the nodes.
- Data Models: A database model is a type of data model that determines the logical structure of a database. It fundamentally determines in which manner data can be stored, organized and manipulated.
- Query Language: Query languages, data query languages or database query languages (DQLs) are computer languages used to make queries in databases and information systems.
- Support: Database Services ensures that customer databases are protected and monitored by establishing backup and recovery procedures, providing a secure database environment, and monitoring database performance.
- Security: Database security refers to the range of tools, controls, and measures designed to establish and preserve database confidentiality, integrity, and availability.
- Transactions: A transactional database is a DBMS that provides the ACID properties for a bracketed set of database operations.
- Documentation: Official documentation given from the database's online site.
- Data Visualization: More information will be provided in Subsection 3.2.1.
- Performance – Read & Write Capability: More information will be provided in Subsection 3.2.2.
- Other Features: More information will be provided in Subsection 3.2.3.

3.2.1 Data Visualization

It concerns the graphical representation of information and data, where using visual elements like charts, graphs and maps, it may provide an accessible way to see and understand trends, outliers, and patterns in data. Some databases are using GUIs(Graphical User Interfaces), a user interface is the view of a database interface that is seen by the user. User interfaces are often graphical or at least partly graphical constructed and offer tools which make the interaction with the database easier.

3.2.2 Performance – Read & Write Capability

At a high level, database performance can be defined as the rate at which a database management system (DBMS) supplies information to users.[46] The performance of accessing and modifying data in the database can be improved by the proper allocation and application of resources. Optimization speeds up query performance. For performance in databases, the *workload*, *throughput*, *data* and *system resources* definitions are integral parts. What follows is an overview of these factors. We are also going to present our datasheet, in which our systems we chose will be tested.

Workload

The workload equals the total demand from the DBMS, and it varies over time.[47] The total workload is a combination of user queries, applications, batch jobs, transactions, and system commands directed through the DBMS at any given time. For example, it can increase when month-end reports are run or decrease on weekends when most users are out of the office. Workload strongly influences database performance. Knowing our workload and peak demand times helps us plan for the most efficient use of our system resources and enables processing the largest possible workload. For our tests we are using open source versions of the dbms we discused earlier which include free licenses for development, startups and academic-educational uses. These dbms are using the default clustering and replication factors, no configurations are made. Furthermore, we are using single database connection, which means that we are using only one shared connection to execute all the queries triggered by us, so workload is something that doesn't consern us.[48]

Throughput

A system's throughput defines its overall capability to process data.[47] DBMS throughput is measured in queries per second, transactions per second, or average response times. DBMS throughput is closely related to the processing capacity of the underlying systems (disk I/O, CPU speed, memory bandwidth, and so on), so it is important to know the throughput capacity of our hardware when setting DBMS throughput goals. For throughput we are going to use the average read/write speed on our dbms for queries-per-second and transactions-per-second measurments. Database write performance will be tested by the average time of uploading large file amounts to each of our database systems, while read performance will be tested by the average query time of simple and more demanding queries to our database systems. These queries are going to be the same in every database performance test. It is important to point out that every single query is being tested 10 times, so we can get an average execution time. We execute 10 tests for each query to have as little deviation as possible in the query times estimates in case of large fluctuations.

After that measurment, we calculate the average time of all the average query time tests, which is the average query time per database. Queries of different kinds will be included, like *searching a maximum value*, *using foreign keys*, *average value calculation*, *counting values*, *sorting*, etc. The queries we are going to use will be presented in Chapter 5. As for the upload performance, every file upload time is being tested 10 times before we reach the final result. Examples of shell notifications about file imports and queries will be provided as well, which are estimates. The actual import and query execution times comes as a result of 10 tests as we discused. The files that are used, are being presented in the next Subchapter (3.2.2).

Datasheet

In this entry, we will consider two general types [49] of databases:

- Flat File databases (which store denormalized data) consist of formats including single or multiple record types, and come in flavors of fixed-length definitions and delimited. The simplest form of flat file is a standard text file and consists of a single record definition. The record or row (as commonly referred to) repeats from one to many times, with each successive row representing a common definition. Every row is made up of a horizontal list of fields and the same definition of the row can be applied to every row in the file.
- Relational databases (which store normalized data) take on the challenge of storing data of differing definitions or formats separately in what is referred to most often as tables. Each table will consist of two groups of columns identified as key values and stored values. The key values make it possible to relate records in one table to another in a parent-child relationship or dependency. A special example of these two general types of databases is given in Figure 3.6.

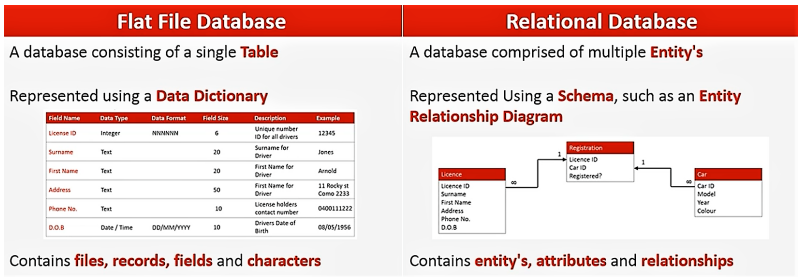


Figure 3.6: Example of a Flat File Database and a Relational one.

The files that we are going to use for testing the different database management systems are CSV format files of thousands of rows. We used a generator for the records. Some of these records will be imported as documents while others as numerical data type records.

- Flat table file: This table will be used from all of our database systems. It consists of 200.000 rows and 1.800.000 records (15,5 MB of data). The visualization of this file is provided in table 3.1.
  - Name of the data table: people
  - Data types per column:

        * id: int
        * firstname: varchar
        * lastname: varchar
        * birthyear: int
        * country: varchar
        * profession: varchar
        * weight: int
        * height: float
        * email: varchar

id	firstname	lastname	birthyear	country	profession	weight	height	email
1	Amara	Vale	1975	Bosnia	Dentist	45	1,7	amaravale@gmail.com
2	Mangalo	Mendez	1954	Panama	Farmer	74	1,82	mangalomendez@gmail
...	...	...	...	...	...	...	...	...
200.000	Nicolas	Felizio	1990	Italy	Waiter	71	1,75	nicolasfelizio@gmail.com

Table 3.1: Summary of the attributes and their values from our flat table file.

- Relational table files: For the relational data model we are going to use 5 CSV format files which relate to each other. Tables 3.5 and 3.6 are going to be used as relations. These tables consist of keys which are used to establish relationships between the different tables and columns of our relational data model. Individual values in a key are called key values. Tables 3.2, 3.3 and 3.4 are going to be used as data tables. Keys id, `country_id`, `profession_id` from tables 3.2, 3.3 and 3.4 are unique. In relation tables 3.5 and 3.6 all of the attributes are used as foreign keys. A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them. In what follows, we present our data and our relation tables.

    – File Data table 1: It consists of 200.000 rows and 1.400.000 records (11,3 MB of data). The visualization of this file is provided in table 3.2.

        * Name of the data table: `people_related`
        * Data types per column:
            · id: int
            · firstname: varchar
            · lastname: varchar
            · birthyear: int
            · weight: int
            · height: float
            · email: varchar

id	firstname	lastname	birthyear	weight	height	email
1	Amara	Vale	1975	45	1,7	amaravale@gmail.com
2	Mangalo	Mendez	1954	74	1,82	mangalomendez@gmail
...	...	...	...	...	...	...
200.000	Nicolas	Felizio	1990	71	1,75	nicolasfelizio@gmail.com

Table 3.2: Summary of the attributes and their values of the File Data table 1.

– File Data table 2: It consists of 244 rows and 488 records (4,8 Kb of data). The visualization of this file is provided in table 3.3.
  * Name of the data table: Country
  * Data types per column:
    · `country_id`: int
    · country: varchar

`country_id`	country
100010	Indonesia
100011	Cuba
...	...
100254	Switzerland

Table 3.3: Summary of the attributes and their values of the File Data table 2.

– File Data table 3: It consists of 56 rows and 112 records (1016 bytes of data). The visualization of this file is provided in table 3.4.
  * Name of the data table: Profession
  * Data types per column:
    · `profession_id`: int
    · profession: varchar

`profession_id`	profession
110000	Electrician
110001	Scientist
...	...
110056	Reporter

Table 3.4: Summary of the attributes and their values of the File Data table 3.

– File Data table 4: It consists of 200000 rows and 400000 records (2,75 MB of data). The visualization of this file is provided in table 3.5.
  * Name of the relation table: `Works_as`
  * Data types per column:
    · id : int

· `profession_id`: int

id	`profession_id`
5432	110322
43	110322
...	...
165323	110007

Table 3.5: Summary of the attributes and their values of the File Data table 4.

– File Data table 5: It consists of 200000 rows and 400000 records (2,75 MB of data). The visualization of this file is provided in table 3.6.
  * Name of the relation table: `Lives_in`
  * Data types per column:
    · id : int
    · `country_id`: int

id	`country_id`
5432	100126
43	100126
...	...
165323	100017

Table 3.6: Summary of the attributes and their values of the File Data table 5.

In these tests we pay attention to the querry complexity, the large file amounts, the cardinality of the database and how these database systems handle the flat type file and the relational type one. At this point, the validity of the files we have chosen is not something that concerns us. Some of the databases we have chosen to create can't support the relational data model or it is not recommended. So their read/write performance will be compared with our MySQL system only by the usage of a flat data type.

System Resources

Database performance relies heavily on disk I/O and memory usage.[47] To accurately set performance expectations, we need to know the baseline performance of the hardware on which our DBMS is deployed. Performance of hardware components such as CPUs, hard disks, disk controllers, RAM, and network interfaces will significantly affect how fast our database performs. At this point we are going to show you the operating system, hardware and connection speed that we are using:

- Operating System: Windows 10 Pro 64-bit
- CPU: AMD Ryzen 5, Raven Ridge 14nm Technology
- RAM: 14,0GB Dual-Channel, 266Mhz

- SSD: Kingston SA2000M8250G
- Motherboard: Gigabyte Technology B450M DS3H-CF(AM4)
- Graphics: 2048MB ATI AMD Radeon RX Vega 11 Graphics(ATI)
- Connection Speed:
  - Download Speed: 40Mbps
  - Upload Speed: 4Mbps

3.2.3 Other Features

Other features stand for the ability of each database to update its data and its capability to alter a table (schema manipulation), by changing an existing schema or creating a new one because new information does not fit in the schemas one database already has. With schema manipulation, we can update individual rows, a subset of all rows or even all the rows in a table.

Chapter 4

In Depth System Study

In this chapter, we are going to evaluate and examine all the database management systems we discused earlier, according to the certain evaluation criteria presented in Chapter 3.

4.1 MySQL

Database

MySQL is an open-source relational database management system (RDBMS).[18][50] Its name is a combination of *My*, the name of co-founder Michael Widenius's daughter,[51] and *SQL*, the abbreviation for Structured Query Language. The first version of MySQL appeared on 23 May 1995. A relational database organizes data into one or more data tables in which data types may be related to each other, these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups. MySQL has received positive reviews, and reviewers noticed it performs extremely well in the average case and that the developer interfaces are there, and the documentation (not to mention feedback in the real world via Web sites and the like) is very, very good. [52] It has also been tested to be *a fast, stable and true multi-user, multi-threaded SQL database server*.[53]

Architecture

The MySQL pluggable storage engine architecture enables a database professional to select a specialized storage engine for a particular application need while being completely shielded from the need to manage any specific application coding requirements.[54] The MySQL server architecture isolates the application programmer and DBA (database administrator) from all of the low-level implementation details at the storage level, providing a consistent and easy application model and API. Thus, although there are different capabilities across different storage engines, the application is shielded from these differences.

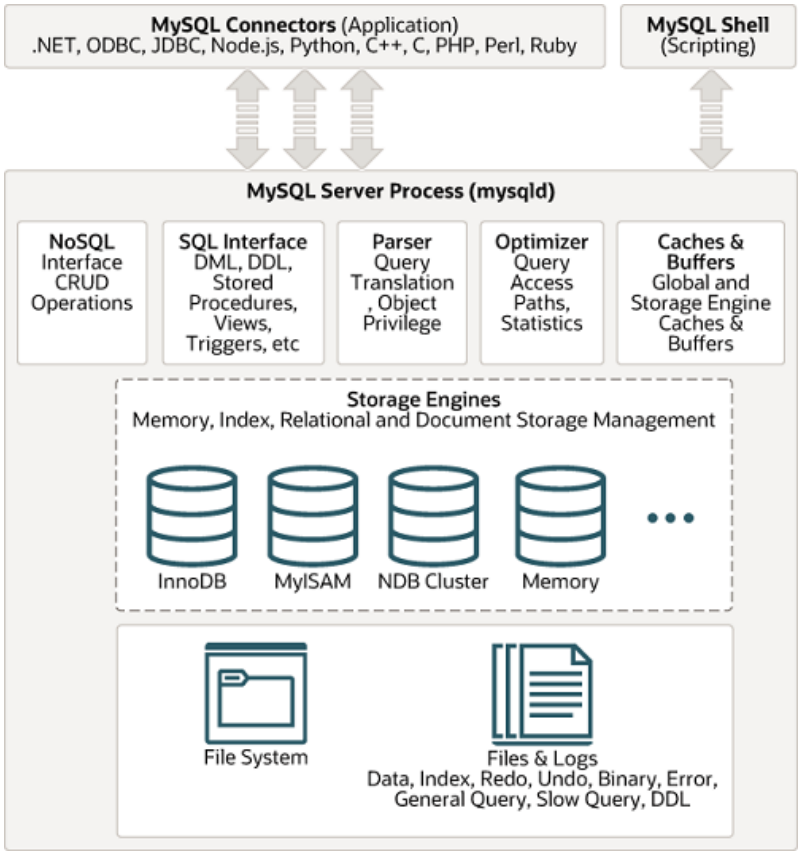


Figure 4.1: The MySQL pluggable storage engine architecture.

The pluggable storage engine architecture as we can see in Figure 4.1 provides a standard set of management and support services that are common among all underlying storage engines. The storage engines themselves are the components of the database server that actually perform actions on the underlying data that is maintained at the physical server level.

This efficient and modular architecture provides huge benefits for those wishing to specifically target a particular application need such as data warehousing, transaction processing, or high availability situations while enjoying the advantage of utilizing a set of interfaces and services that are independent of any one storage engine.

The application programmer and DBA interact with the MySQL database through Connector APIs and service layers that are above the storage engines. If application changes bring about requirements that demand the underlying storage engine change, or that one or more storage engines be added to support new needs, no significant coding or process changes are required to make things work. The MySQL server architecture shields the application from the underlying complexity of the storage engine by presenting a consistent and easy-to-use API that applies across storage engines.

Architecture of MySQL describes the relation among the different components of MySQL System. MySQL follow Client-Server Architecture. It is designed so that end user that is Clients can access the resources from Computer that is server using various networking services. The Architecture of MySQL contain the following major layer's:

- Client: This layer is the topmost layer in the above diagram presented in Figure 4.1. The

Client gives request instructions to the Server with the help of the Client Layer. Then, the Client makes request through Command Prompt or through GUI (graphical user interface) screen by using valid MySQL commands and expressions. If the expressions and commands are valid then the output is obtained on the screen.

- Server: The second layer of MySQL architecture is responsible for all logical functionalities of relational database management systems of MYSQL. This layer of MySQL System is also known as *Brain of MySQL Architecture*. When the Client gives request instructions to the Server, then the Server gives the output as soon as the the instruction is matched.
- Storage Layer: This storage engine layer of MySQL Architecture make this system unique and most preferable for developers. In MySQL Server, for different situations and requirements, different types of storage engines are used which are InnoDB ,MYiSAM, NDB, Memory etc.

Scalability

Although most SQL databases scale vertically, MySQL Cluster automatically shards (partitions) tables across nodes, enabling databases to scale horizontally on low cost, commodity hardware to serve read and write intensive workloads, accessed both from SQL and directly via NoSQL APIs (application programming interfaces).[54] What is Database Sharding? Sharding is a method for distributing a single dataset across multiple databases, which can then be stored on multiple machines as we can see in Figure 4.2. Sharding is entirely transparent to the application which is able to connect to any node in the cluster and have queries automatically access the correct shards. With its active/active, multi-master architecture, updates can be handled by any node, and are instantly available to all of the other clients accessing the cluster.

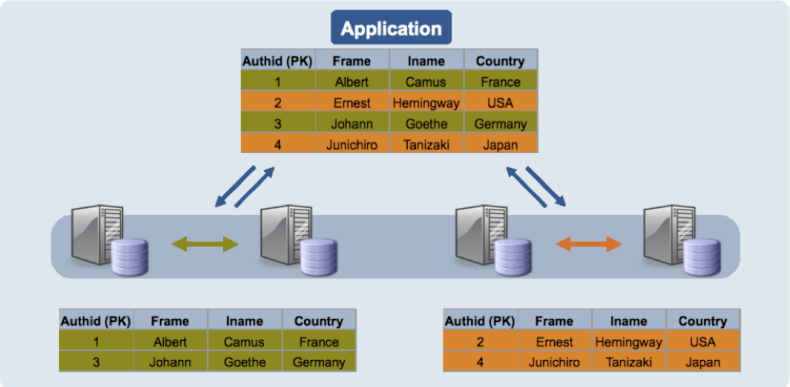


Figure 4.2: Representation of how MySQL sharding works.

Unlike other sharded databases, database users do not lose the ability to perform JOIN operations (SQL operations performed to establish connections between two or more database tables), sacrifice ACID-guarantees or referential integrity (Foreign Keys) when performing queries and transactions across shards.

MySQL Cluster also replicates across data centers for disaster recovery and global scalability. Using its conflict handling mechanisms, each cluster can be active, accepting updates while maintaining consistency across locations.

Replication(Nodes, Internode Communication)

Replication enables data from one MySQL database server (known as a source) to be copied to one or more MySQL database servers (known as replicas).[54] A simple replication example in MySQL is presented in Figure 4.3. Replication is asynchronous by default, replicas do not need to be connected permanently to receive updates from a source. Depending on the configuration, we can replicate all databases, selected databases, or even selected tables within a database.

Advantages of replication in MySQL include:

- Scale-out solutions: By spreading the load among multiple replicas to improve performance. In this environment, all writes and updates must take place on the source server. Reads, however, may take place on one or more replicas. This model can improve the performance of writes (since the source is dedicated to updates), while dramatically increasing read speed across an increasing number of replicas.
- Data security: because the replica can pause the replication process, it is possible to run backup services on the replica without corrupting the corresponding source data.
- Analytics: Live data can be created on the source, while the analysis of the information can take place on the replica without affecting the performance of the source.
- Long-distance data distribution: We can use replication to create a local copy of data for a remote site to use, without permanent access to the source.

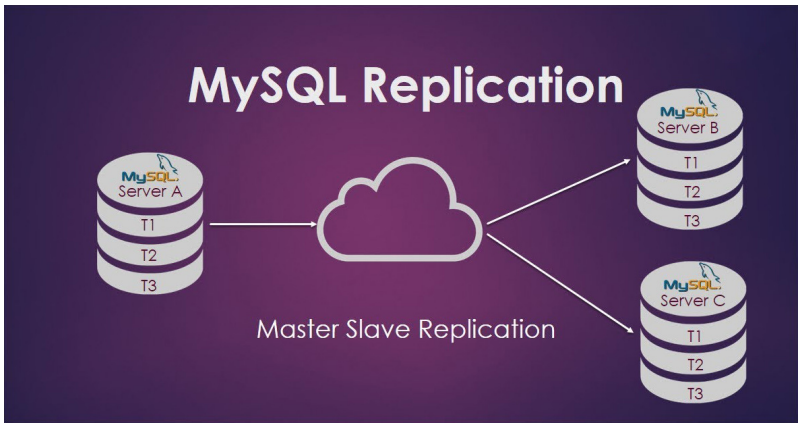


Figure 4.3: A simple Master-Slave replication example in MySQL.

Replication in MySQL supports different types of synchronization. The original type of synchronization is one-way, asynchronous replication, in which one server acts as the source, while one or more other servers act as replicas. This is in contrast to the synchronous replication which is a characteristic of *NDB Cluster*. NDB Cluster is the distributed database system underlying MySQL Cluster. It can be used independently of a MySQL Server with users accessing the Cluster via the NDB API (by using C++). *NDB* stands for Network Database. We can expore NDB cluster's components in Figure 4.5. In MySQL 8.0, semisynchronous replication is supported in addition to the built-in asynchronous replication. With semisynchronous replication, a commit performed on the source blocks before returning to the session that performed the transaction until at least one replica acknowledges that it has received and logged the events for the transaction.

MySQL Cluster is implemented as a strongly consistent, active/active, multi-master database ensuring updates can be made to any node and are instantly available to the rest of the cluster, without any replication lag. Tables are automatically sharded across a pool of low cost commodity data nodes, enabling the database to scale horizontally, accessed both from SQL and directly via NoSQL APIs. New nodes can be added on-line, instantly scaling database capacity and performance, even for the heaviest write loads. MySQL Cluster is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment as we can see in Figure 4.4.

There are three types of cluster nodes, and in a minimal NDB Cluster configuration, there are at least three nodes, one of each of the types that follow.

- Management node: The role of this type of node is to manage the other nodes within the NDB Cluster, performing such functions as providing configuration data, starting and stopping nodes, and running backups. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node.
- Data node: This type of node stores cluster data. There are as many data nodes as there are fragment replicas, times the number of fragments. NDB Cluster tables are normally stored completely in memory rather than on disk (this is why we refer to NDB Cluster as an in-memory database). However, some NDB Cluster data can be stored on disk.
- SQL node: This is a node that accesses the cluster data. In the case of NDB Cluster, an SQL node is a traditional MySQL server that uses the NDB cluster storage engine.

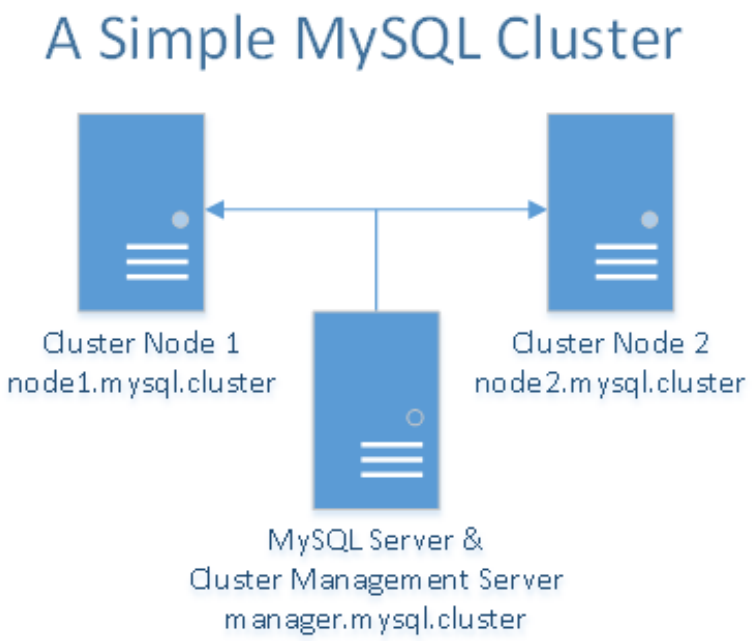


Figure 4.4: An example of a Multi-Node MySQL Cluster.

An SQL node is actually just a specialized type of API (application programming interface) node, which designates any application which accesses NDB Cluster data. Individual nodes can be stopped and restarted, and can then rejoin the system (cluster).

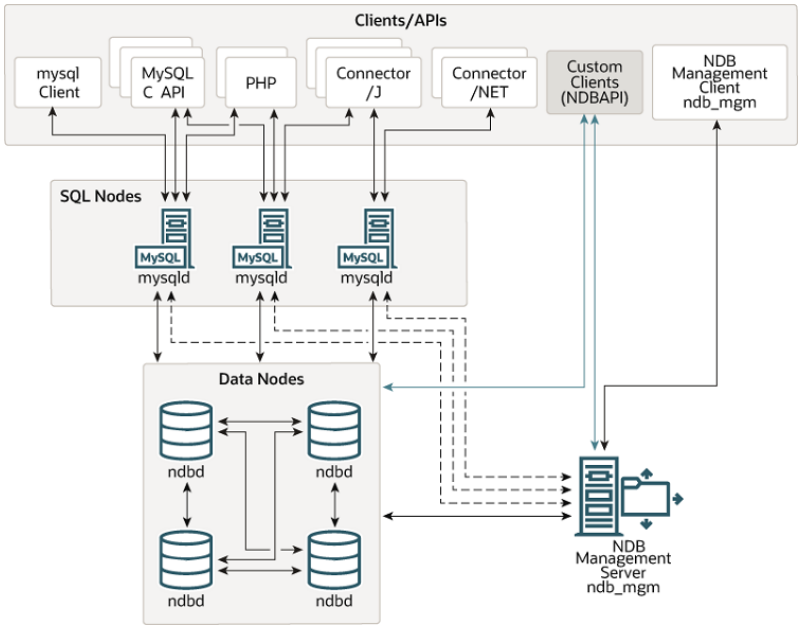


Figure 4.5: NDB Cluster Components.

NDB Cluster requires communication between data nodes and API nodes (including SQL nodes), as well as between data nodes and other data nodes, to execute queries and updates. Communication latency between these processes can directly affect the observed performance and latency of user queries. In addition, to maintain consistency and service despite the silent failure of nodes, NDB Cluster uses heartbeating and timeout mechanisms which treat an extended loss of communication from a node as node failure.

The failure of a data or API (application programming interface) node results in the abort of all uncommitted transactions involving the failed node. Data node recovery requires synchronization of the failed node's data from a surviving data node, and re-establishment of disk-based redo and checkpoint logs, before the data node returns to service. This recovery can take some time, during which the Cluster operates with reduced redundancy.

Heartbeating relies on timely generation of heartbeat signals by all nodes. This may not be possible if the node is overloaded, has insufficient machine CPU due to sharing with other programs, or is experiencing delays due to swapping. If heartbeat generation is sufficiently delayed, other nodes treat the node that is slow to respond as failed.[54]

Data Models

MySQL is relational in nature since all the data is stored in different tables and relations are established using primary keys or other keys known as foreign keys. A table is used to organize data in the form of rows and columns and used for both storing and displaying records in the structure format as we can see in Figure 4.6. It is similar to worksheets in the spreadsheet application. A table creation command requires the following three things.

- Name of the table
- Names of fields

- Definitions for each field

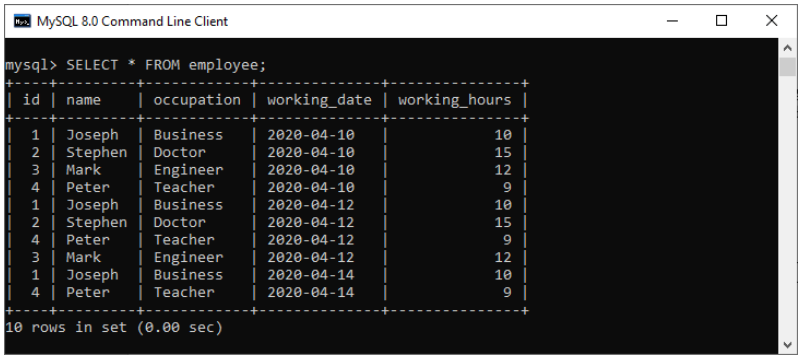


Figure 4.6: An example of a simple MySQL table.

Normalization is the procedure of professionally organizing data in a database. Normalization database schema design technique, by which an existing schema is modified to minimize redundancy and dependency of data, eliminating redundancy data means this goal will storing the same data in more than one table and the second one is ensuring data dependencies make sense means only storing related data in a table. In organizing data Normalization split a large table into smaller tables and it defines relationships among them to increases the simplicity. Since MySQL is a relational SQL database, we tend to use normalized data in it.[54]

Query Language

MySQL is developed in C and C++ programming languages with testing on a broad range of compilers and it uses the SQL language to query the database.[54] MySQL works on and supports different types of programming language platforms. It was designed to support multithreaded kernels with a multi-layered server design to use multiple CPUs. It able to perform joins very fast using optimization, and have separate storage for transactional and non-transactional. Internally uses hash tables which will be used as temporary tables.

MySQL supports functions and full operators in *SELECT* and *WHERE* clauses of a query. It also supports left outer joinss and right outer joins with basic syntax and ODBC (ODBC are actually drivers that provide access to a MySQL database management system) syntax. Furthermore, MySQL querry language supports curd operations like *Insert*, *Delete*, *Replace*, and *Update* statements which return the number of rows updated, inserted, and deletes the rows which match the condition.

Support

MySQL supports different types of backups.[54] Physical backups consist of raw copies of the directories and files that store database contents. This type of backup is suitable for large, important databases that need to be recovered quickly when problems occur. Logical backups save information represented as logical database structure and content (INSERT statements or delimited-text files). This type of backup is suitable for smaller amounts of data where we might edit the data values or table structure, or recreate the data on a different machine architecture. In what follows we explain in more detail, the twobackup options.

Physical backup methods have the following characteristics.

- The backup consists of exact copies of database directories and files. Typically this is a copy of all or part of the MySQL data directory.
- Physical backup methods are faster than logical because they involve only file copying without conversion.
- Output is more compact than for logical backup.
- Because backup speed and compactness are important for busy, important databases, the MySQL Enterprise Backup product performs physical backups.
- Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files. This may or may not provide for table-level granularity, depending on storage engine
- In addition to databases, the backup can include any related files such as log or configuration files.
- Backups are portable only to other machines that have identical or similar hardware characteristics.
- Backups can be performed while the MySQL server is not running. If the server is running, it is necessary to perform appropriate locking so that the server does not change database contents during the backup. MySQL Enterprise Backup does this locking automatically for tables that require it.

Logical backup methods have the following characteristics.

- The backup is done by querying the MySQL server to obtain database structure and content information.
- Backup is slower than physical methods because the server must access database information and convert it to logical format. If the output is written on the client side, the server must also send it to the backup program.
- Output is larger than for physical backup, particularly when saved in text format.
- Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This is true regardless of storage engine.
- The backup does not include log or configuration files, or other database-related files that are not part of databases.
- Backups stored in logical format are machine independent and highly portable.
- Logical backups are performed with the MySQL server running. The server is not taken offline.

Security

MySQL supports the usage of passwords in several cases.[54] Also with an unencrypted connection between the MySQL client and the server, someone with access to the network could watch all our traffic and inspect the data being sent or received between client and server. When we must move information over a network in a secure fashion, an unencrypted connection is unacceptable. However, to make any kind of data unreadable, we may use encryption. Encryption algorithms must include security elements to resist many kinds of known attacks such as changing the order of encrypted messages or replaying data twice. The supported encryption protocols are presented bellow.

- MySQL supports encrypted connections between clients and the server using the TLS (Transport Layer Security) protocol. TLS is sometimes referred to as SSL (Secure Sockets Layer) but MySQL does not actually use the SSL protocol for encrypted connections because its encryption is weak. TLS uses encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect data change, loss, or replay. TLS also incorporates algorithms that provide identity verification using the X.509 standard.

- X.509 makes it possible to identify someone on the Internet. In basic terms, there should be some entity called a *Certificate Authority* (or CA) that assigns electronic certificates to anyone who needs them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key and a secret key). A certificate owner can present the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted using this public key can be decrypted only using the corresponding secret key, which is held by the owner of the certificate.

Furthermore, MySQL Enterprise Edition supports an authentication method that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users as we can see in Figure 4.7. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as traditional Unix passwords or an LDAP directory (LDAP is an open and cross platform protocol used for directory services authentication).

Transactions

MySQL fully satisfies the ACID requirements for a transaction-safe relational database management system,[54] as follows.

- Atomicity is handled by storing the results of transactional statements (the modified rows) in a memory buffer and writing these results to disk and to the binary log from the buffer only once the transaction is committed. This ensures that the statements in a transaction operate as an indivisible unit and that their effects are seen collectively, or not at all.

- Consistency is primarily handled by MySQL's logging mechanisms, which record all changes to the database and provide an audit trail for transaction recovery. In addition to the logging process, MySQL provides locking mechanisms that ensure that all of the tables, rows, and indexes that make up the transaction are locked by the initiating process long enough to either commit the transaction or roll it back.

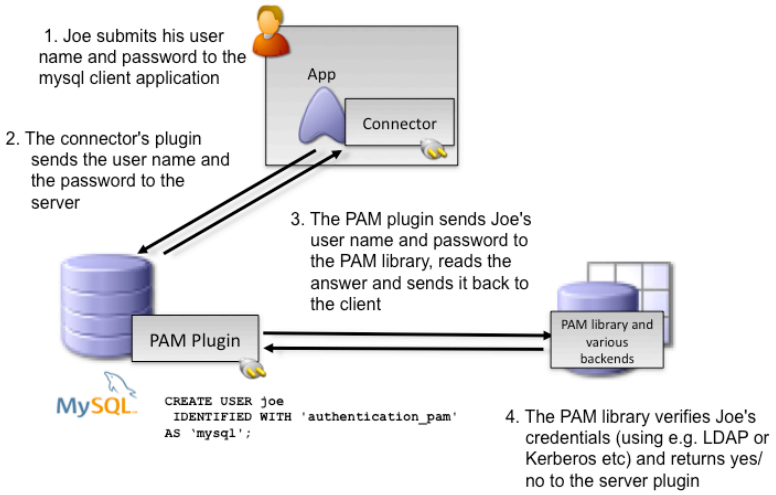


Figure 4.7: MySQL Enerprise Edition authentication.

- Server-side semaphore variables and locking mechanisms act as traffic managers to help programs manage their own isolation mechanisms. For example, MySQL's InnoDB engine uses fine-grained row-level locking for this purpose.

- MySQL implements durability by maintaining a binary transaction log file that tracks changes to the system during the course of a transaction. In the event of a hardware failure or abrupt system shutdown, recovering lost data is a relatively straightforward task by using the last backup in combination with the log when the system restarts. By default, InnoDB tables are 100 percent durable (in other words, all transactions committed to the system before the crash are liable to be rolled back during the recovery process), while MyISAM tables offer partial durability.

Documentation

We found MySQL's documentation the most complete of all. We managed to find everything we need. This is the link for the official documentation for MySQL: https://dev.mysql.com/doc

4.2 Neo4j

Database

Neo4j is a graph database management system developed by Neo4j, Inc. Version 1.0 was released in February 2010. Described by its developers as an ACID-compliant transactional database with native graph storage and processing,[55] Neo4j is available in a GPL3-licensed (GPL 3 is a

strong copyleft license), open-source *community edition*, with online backup and high availability extensions licensed under a closed-source commercial license.[56]

[57]Neo4j is implemented in Java and accessible from software written in other languages using the Cypher query language (which is the Neo4j query language) through a transactional HTTP endpoint, or through the binary *Bolt* protocol. We are going to discuss all these terms in the following subsections.

Neo4j enables organizations to unlock the business value of connections, influences and relationships in data through new applications that adapt to changing business needs, and by enabling existing applications to scale with the business.

Today, thousands of organizations from startups to Fortune 500 companies (a list of 500 of the largest companies in the United States compiled by Fortune magazine every year) are using Neo4j to build new and innovative applications that leverage connections in data such as recommendations, impact analysis for network and IT operations, real-time routing for logistics and the next generation business applications such as master data management, identity and access management, content management, fraud detection, portfolio and risk management.

Architecture

Neo4j's distributed high-performance architecture is fault-tolerant and guarantees data integrity (ACID compliant) in all topologies ensuring safe scalability without having to compromise performance.[56] The stateful, cluster-aware sessions with encrypted connections do not require load balancers to ensure all client requests are seamlessly redirected to the correct server.

The architecture is designed for optimal management, storage, and traversal of nodes and relationships. The graph database takes a property graph approach, which is beneficial for both traversal performance and operations runtime.

Neo4j's High Availability (HA) Architecture, which is shown in Figure 4.8, has a cluster of three instances and at any one point of time one of those instances is acting as the master, and the master is responsible for coordinating all of the writes to the system.

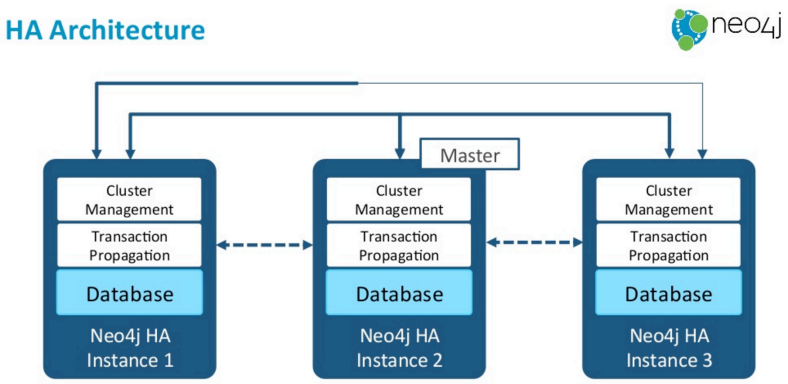


Figure 4.8: Representation of HA architecture.

Scalability

A large graph database may have a potentially unlimited number of nodes.[58] Neo4j uses *horizontal scaling* So the ability to divide the graph database across many servers is key to scala-

bility, as well as the ability to support use cases such as compliance with data privacy regulations. We can see how Neo4j scales in Figure 4.9.

Sharding is the division of a single logical database into as many physical machines as is required. Neo4j provides horizontal scalability via sharding for mission-critical applications with a minutes-to-milliseconds performance advantage.

Neo4j also uses *Fabric* which is basically a way to store and retrieve data in multiple databases. Fabric provides scalability by simplifying the data model to reduce complexity. With Fabric, we can execute queries in parallel on multiple databases, combining or aggregating results. It also allows us to chain queries together from multiple databases for sophisticated, real-time analyses. Fabric helps us achieve a number of functions, as follows.

- A unified view of local and distributed data, accessible via a single client connection and user session.
- Increased scalability for read/write operations, data volume, and concurrency.
- Predictable response time for queries executed during normal operations, a failover, or other infrastructure changes.
- High Availability and no single point of failure for large data volume.

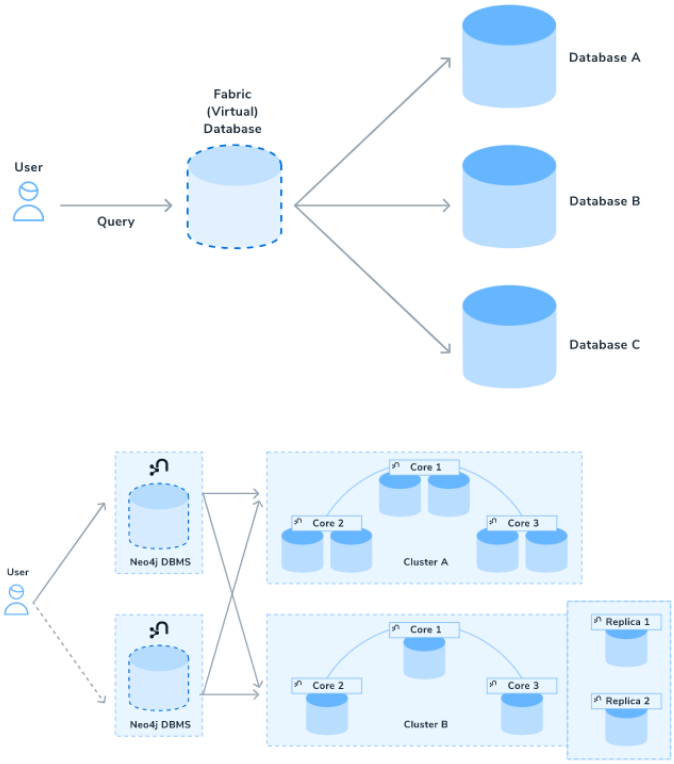


Figure 4.9: Representation of how Neo4j scales.

Replication(Nodes, Internode Communication)

Neo4j uses *Causal Clustering* which enables support for large clusters and different cluster topologies for data center and cloud.[58] We can see how causal clusering works in Figure 4.10. Causal clustering provides the following three main features.

- Safety: Primary Servers provide a fault tolerant platform for transaction processing which will remain available while a simple majority of those servers are functioning.

- Scale: Secondary Servers provide a scalable platform for graph queries that enables very large graph workloads to be executed in a distributed topology.

- Causal consistency: Through the use of bookmarks, a client application is guaranteed to read at least its own writes.

Together, this allows the end-user system to be fully functional and both read and write to the database in the event of multiple hardware and network failures and makes reasoning about database interactions straightforward.

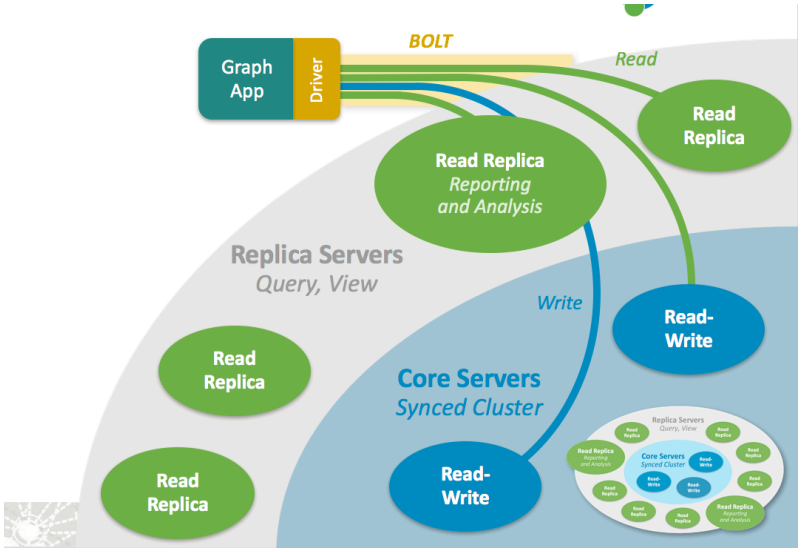


Figure 4.10: Representation of how Causal Clustering works.

As for replication, Neo4j uses *Causal Consistency* which makes it possible to write to Core Servers (where data is safe) and read those writes from a Read Replica (where graph operations are scaled out), as we can see in Figure 4.11. For example, causal consistency guarantees that the write which created a user account will be present when that same user subsequently attempts to log in.

Figure 4.11: Cluster setup with causal consistency via Neo4j drivers.

In Neo4j (v4.0+), we can create and use more than one active database at the same time. This works in standalone and causal cluster scenarios and allows us to maintain multiple, separate graphs in one installation. When we create a database, Neo4j will initially create a system database and a default database. The system database is named system and contains the overall information that applies across databases - managing administration of individual databases (stopping and starting) and maintaining user privileges (security roles and privileges). The default database is named neo4j (can be changed) and is where we can store and query data in a graph and integrate with other applications and tools. We can also create additional databases, as needed, to store other graphs and different data that may be unrelated to any of our other databases.

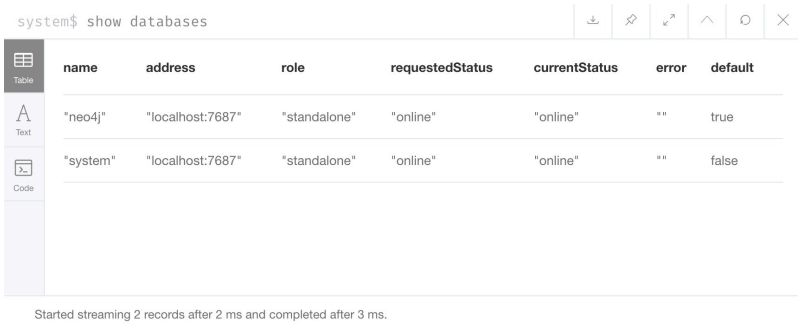


Figure 4.12: Representation of a neo4j instance.

An instance can contain many databases where emphnode or relational data is stored as we can see in Figure 4.12.

The Neo4j Browser and the official Neo4j Drivers use the Bolt database protocol to communicate with Neo4j. The Bolt protocol is implemented by connectors on either side of the communication. One connector sits in the driver itself, which acts on behalf of the application, and the other connector sits in the nearest server. These connectors send Cypher (querry language) messages to the server, which then sends streams of records (the results) back to the connectors.

Data Models

Neo4j uses a property graph database model.[58] A graph data structure consists of nodes (discrete objects) that can be connected by relationships. An example of 4 nodes is shown in Figure 4.13.

The Neo4j graph database model consists of the following properties.

- Nodes that describe entities (discrete objects) of a domain.
- Labels that define (classify) nodes i.e., describe what kind is the kind of each node. Nodes can have zero or more labels.
- Relationships that describe a connection between a source node and a target node. Relationships always have a direction (one direction). In addition, relationships must have a type (one type) to define (classify) what type of relationship they are.
- Nodes and relationships can have properties (key-value pairs), which further describe them.

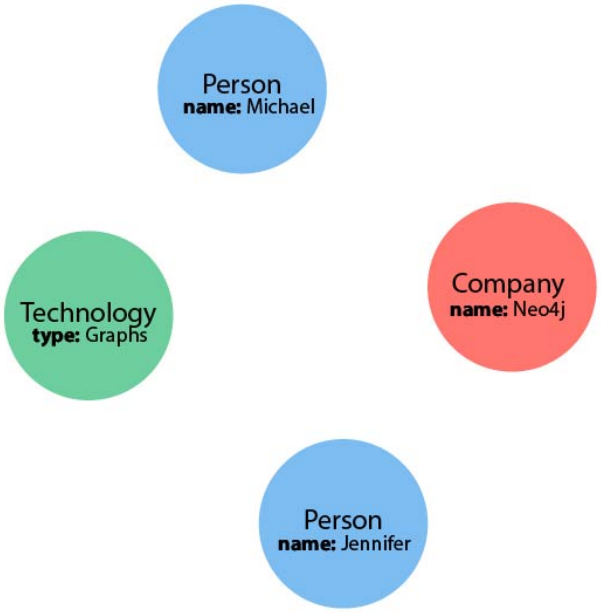


Figure 4.13: An example of 4 nodes with their properties.

A relationship describes how a connection between a source node and a target node are related. We can see a related graph concept in Figure 4.14. It is possible for a node to have a relationship to itself. A relationship includes the following functions.

- Connects a source node and a target node.
- Has a direction (one direction).
- Must have a type (one type) to define (classify) what type of relationship it is.
- Can have properties (key-value pairs), which further describe the relationship.
- Relationships organize nodes into structures, allowing a graph to resemble a list, a tree, a map, or a compound entity — any of which may be combined into yet more complex, richly inter-connected structures.

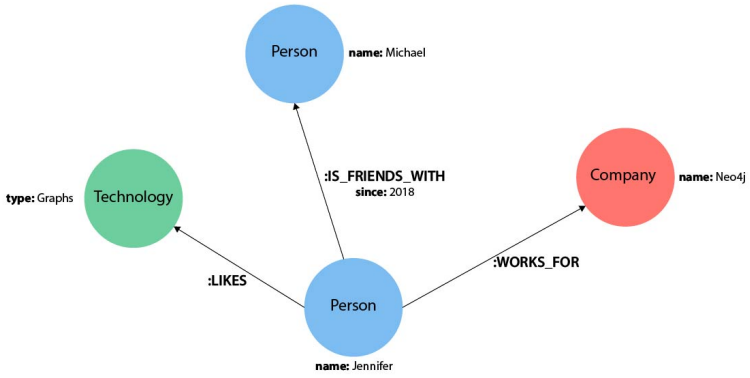


Figure 4.14: A concept of a graph structure.

Neo4j is often described as schema optional, meaning that it is not necessary to create indexes and constraints. We can create data nodes, relationships and properties without defining a schema up front. Indexes and constraints can be introduced when desired, in order to gain performance or modeling benefits.

Query Language

Cypher is Neo4j's graph query language that allows users to store and retrieve data from the graph database.[58] Neo4j wanted to make querying graph data easy to learn, understand, and use for everyone, but also incorporate the power and functionality of other standard data access languages. This is what Cypher aims to accomplish. Cypher's syntax provides a visual and logical way to match patterns of nodes and relationships in the graph. It is a declarative, SQL-inspired language for describing visual patterns in graphs. Cypher query language allows us to state what we want to select, insert, update, or delete from our graph data without a description of exactly how to do it. Through Cypher, users can construct expressive and efficient queries to handle needed create, read, update, and delete functionality. We already know that Neo4j's property graph model is composed of nodes and relationships, which may also have properties associated with them. However, nodes and relationships are the simple components that build the most valuable and powerful piece of the property graph model - the pattern. Patterns are comprised of node and relationship elements and can express simple or complex traversals and paths. With Cypher, we

can specify a relationship direction with the *greater than* > or *less than* < signs, as we can see in Figure 4.15. Pattern recognition is fundamental to the way that the brain works. Cypher is also heavily based on patterns and is designed to recognize various versions of these patterns in data, making it a simple and logical language for users to learn.

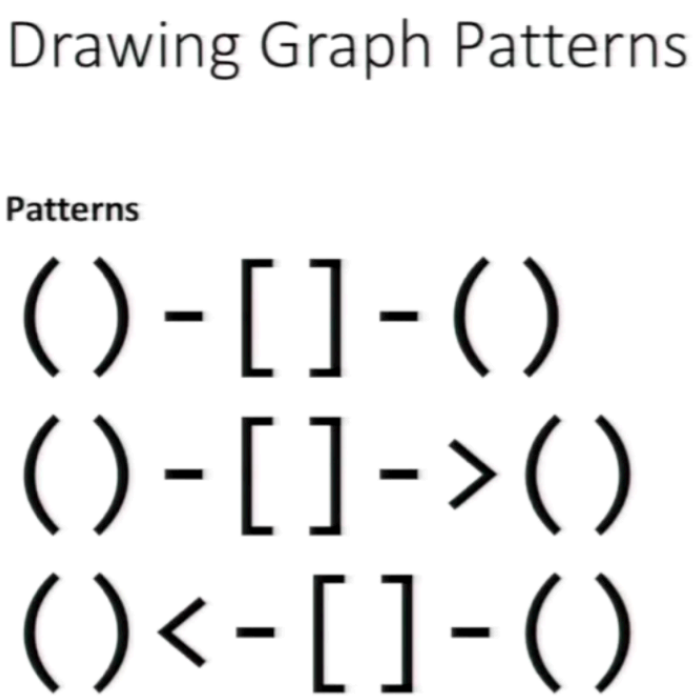


Figure 4.15: Cypher relation graph patterns.

Support

Neo4j supports backing up and restoring both online and offline databases.[58] It uses Neo4j Admin tool commands, which can be run from a live, as well as from an offline Neo4j DBMS. A Neo4j DBMS can host multiple databases. Both Neo4j Community and Enterprise Editions have a default user database, called neo4j, and a system database, which contains configurations, e.g., operational states of databases, security configuration, schema definitions, login credentials, and roles. In the Enterprise Edition, we can also create additional user databases. Each of these databases are backed up independently of one another. For any backup, it is important that we store our data separately from the production system, where there are no common dependencies, and preferably off-site. If we are running Neo4j in the cloud, we may use a different availability zone or even a separate cloud provider. Since backups are kept for a long time, the longevity of archival storage should be considered as part of backup planning.

Security

Neo4j uses a variety of security measures as follows.[58]

- Neo4j supports the securing of communication channels using standard SSL(Secure Sockets Layer)/TLS(Transport Layer Security) technology.
- Neo4j Browser has two mechanisms for avoiding users having to repeatedly enter their Neo4j credentials. First, while the Browser is open in a web browser tab, it ensures that the existing database session is kept alive. This is subject to a timeout. The timeout is reset whenever there is user interaction with the Browser.Second, the Browser can also cache the user's Neo4j credentials locally. When credentials are cached, they are stored unencrypted in the web browser's local storage
- It also uses subnets, firewalls and volume encryption.

Transactions

In order to fully maintain data integrity and ensure good transactional behavior, Neo4j supports all of the following ACID properties.[58]

- Atomicity, if any part of a transaction fails, the database state is left unchanged.
- Consistency, any transaction will leave the database in a consistent state.
- Isolation, during a transaction, modified data cannot be accessed by other operations.
- Durability, the DBMS can always recover the results of a committed transaction.

Specifically, all database operations that access the graph, indexes, or the schema must be performed in a transaction. The default isolation level is read-committed isolation level. Data retrieved by traversals is not protected from modification by other transactions. Non-repeatable reads may occur (i.e., only write locks are acquired and held until the end of the transaction).

Documentation

We found Neo4j's documentation a little hard to navigate, but it covered all the criteria we needed to find. This is the link for the official documentation for Neo4j: `https://neo4j.com/docs/`

4.3 Cassandra

Database

Apache Cassandra is a free and open-source, distributed, wide-column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.[59] Avinash Lakshman, one of the authors of Amazon's Dynamo, and Prashant Malik initially developed Cassandra at Facebook to power the Facebook inbox search feature. Facebook released Cassandra as an open-source project on Google code in July 2008. In March 2009 it became an Apache Incubator project.[60] On February 17, 2010 it graduated to a top-level project. Facebook developers named their database after the Trojan mythological prophet Cassandra, with classical allusions to a curse on an oracle.

Architecture

Cassandra is designed to handle big data workloads across multiple nodes with no single point of failure.[61] Its architecture is based on the understanding that system and hardware failures can and do occur. Cassandra addresses the problem of failures by employing a peer-to-peer distributed system across homogeneous nodes where data is distributed among all nodes in the cluster. Each node frequently exchanges state information about itself and other nodes across the cluster using peer-to-peer gossip communication protocol. A sequentially written commit log on each node captures write activity to ensure data durability. Data is then indexed and written to an in-memory structure, called a memtable, which resembles a write-back cache. All writes are automatically partitioned and replicated throughout the cluster. To ensure all data across the cluster stays consistent, various repair mechanisms are employed.

Cassandra is a partitioned row store database, where rows are organized into tables with a required primary key. Cassandra's architecture allows any authorized user to connect to any node in any datacenter and access data using the *CQL* language (Cassandra query language). For ease of use, CQL uses a similar syntax to SQL and works with table data. Developers can access CQL through cqlsh, DevCenter, and via drivers for application languages. Typically, a cluster has one keyspace per application composed of many different tables.

Client read or write requests can be sent to any node in the cluster. When a client connects to a node with a request, that node serves as the coordinator for that particular client operation. The coordinator acts as a proxy between the client application and the nodes that own the data being requested. The coordinator determines which nodes in the ring should get the request based on how the cluster is configured.

One important Cassandra attribute is that its databases are distributed. Cassandra databases easily scale when an application is under high stress, and the distribution also prevents data loss from any given datacenter's hardware failure. That yields both technical and business advantages. MySQL also uses a distributed database technology which is actually a single relational database which replicates data across multiple servers as we have recently seen in Section 4.1. A distributed architecture also brings technical power, for example, a developer can tweak the throughput of read queries or write queries in isolation.

Scalability

One reason for Cassandra's popularity is that it enables developers to scale their databases dynamically, using off-the-shelf hardware, with no downtime.[61] We can expand when we need to and also shrink, if the application requirements suggest that path.

We know that extending Oracle or MySQL databases to support more users or storage capacity requires to add more CPU power, RAM, or faster disks. Each of those costs a significant amount of money.

Cassandra makes it easy to increase the amount of data it can manage. Because it is based on nodes, Cassandra scales horizontally (aka scale-out), using lower commodity hardware as it is shown in Figure 4.16. To double capacity or double throughput, double the number of nodes. We also have the flexibility to scale back if we wish.

This linear scalability applies essentially indefinitely. This capability has become one of Cassandra's key strengths.

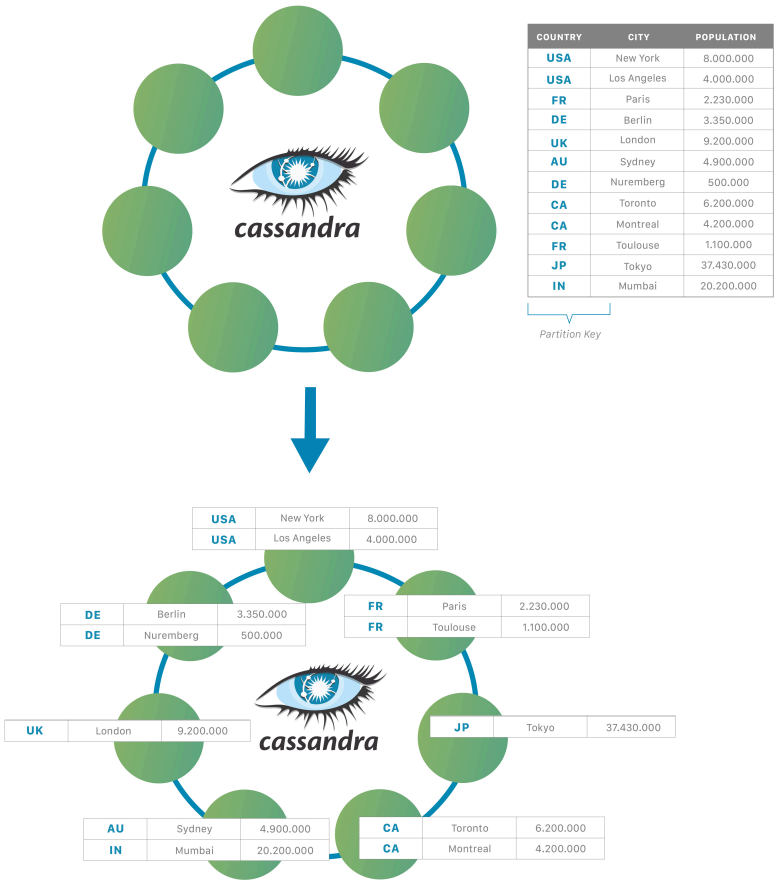


Figure 4.16: An example that shows how Cassandra scales.

Replication(Nodes, Internode Communication)

Cassandra stores replicas on multiple nodes to ensure reliability and fault tolerance.[61] A replication strategy determines the nodes where replicas are placed. The total number of replicas across the cluster is referred to as the replication factor. A replication factor of 1 means that there is only one copy of each row in the cluster. If the node containing the row goes down, the row cannot be retrieved. A replication factor of 2 means two copies of each row, where each copy is on a different node. All replicas are equally important, there is no primary or master replica. As a general rule, the replication factor should not exceed the number of nodes in the cluster. However, we can increase the replication factor and then add the desired number of nodes later.

Two replication strategies are available:

- SimpleStrategy: Which is mainly used only for a single datacenter and one rack. If we ever want more than one datacenter, we should consider to use the following NetworkTopologyStrategy.

- NetworkTopologyStrategy: This strategy is highly recommended for most deployments because it is a strategy in which we can actually store multiple copies of data on different data centers as per need.

The Cassandra consistency level is defined as the minimum number of Cassandra nodes that must acknowledge a read or write operation before the operation can be considered successful. We can see how consistency levels work in a Cassandra cluster in Figure 4.17. Different consistency levels can be assigned to different Edge keyspaces.

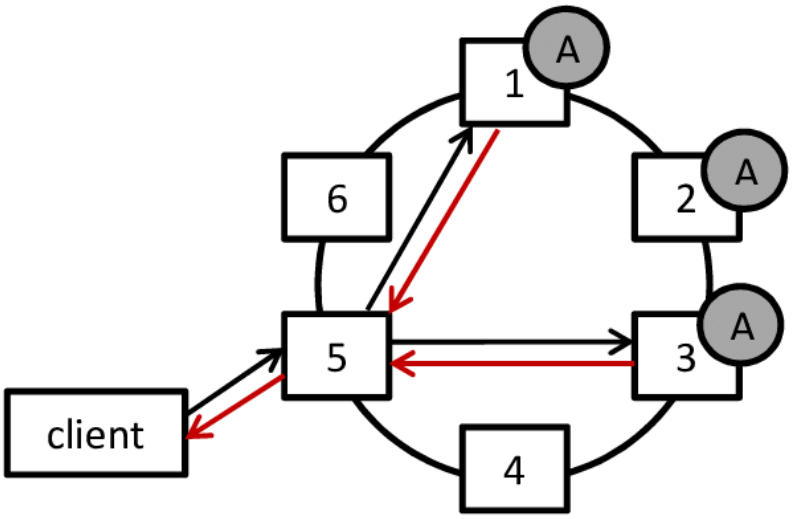


Figure 4.17: Example of a read operation on a 6 node Cassandra cluster with replication factor 3 and consistency level 2.

Since it is a distributed database, Cassandra can (and usually does) have multiple nodes as we can see in Figure 4.18. A node represents a single instance of Cassandra. These nodes communicate with one another through a protocol called gossip, which is a process of computer peer-to-peer communication.[61] Cassandra also has a masterless architecture in which any node in the database can provide the exact same functionality as any other node by contributing to Cassandra's robustness and resilience. Multiple nodes can be organized logically into a cluster, or *ring*. We can also have multiple datacenters.

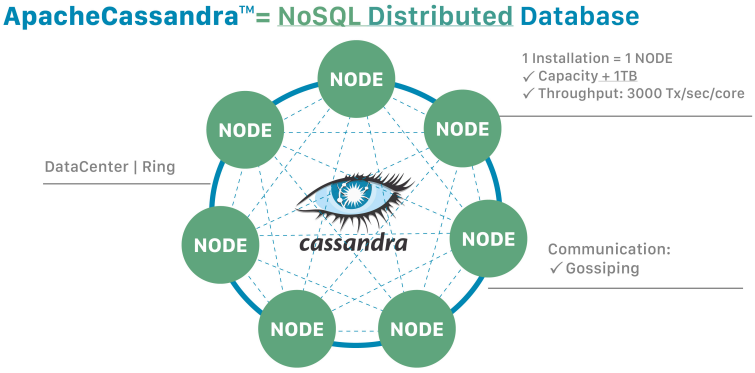


Figure 4.18: Cassandra as a distributed database.

Gossip is a peer-to-peer communication protocol in which nodes periodically exchange state information about themselves and about other nodes they know about. The gossip process runs

every second and exchanges state messages with up to three other nodes in the cluster. The nodes exchange information about themselves and about the other nodes that they have gossiped about, so all nodes quickly learn about all other nodes in the cluster. A gossip message has a version associated with it, so that during a gossip exchange, older information is overwritten with the most current state for a particular node.

To prevent problems in gossip communications, we use the same list of seed nodes for all nodes in a cluster. By default, a node remembers other nodes it has gossiped with between subsequent restarts. The seed node designation has no purpose other than bootstrapping the gossip process for new nodes joining the cluster. Seed nodes are not a single point of failure, nor do they have any other special purpose in cluster operations beyond the bootstrapping of nodes.

Data Models

In Cassandra, data modeling is query-driven.[61] The data access patterns and application queries determine the structure and organization of data which are then used to design the database tables.

- Query-driven modeling: Unlike a relational database model in which queries make use of table joins to get data from multiple tables, joins are not supported in Cassandra so all required fields (columns) must be grouped together in a single table. Since each query is backed by a table, data is duplicated across multiple tables in a process known as denormalization. Data duplication and a high write throughput are used to achieve a high read performance.

- Goals: The choice of the primary key and partition key is important to distribute data evenly across the cluster. Keeping the number of partitions read for a query to a minimum is also important because different partitions could be located on different nodes and the coordinator would need to send a request to each node adding to the request overhead and latency. Even if the different partitions involved in a query are on the same node, fewer partitions make for a more efficient query.

- Partitions: Apache Cassandra is a distributed database that stores data across a cluster of nodes. A partition key is used to partition data among the nodes. Partition keys are parts of Cassandra keyspaces (keyspace is an outermost object that determines how data replicates on nodes) as it is shown in Figure 4.19. Cassandra partitions data over the storage nodes using a variant of consistent hashing for data distribution. Hashing is a technique used to map data with which given a key, a hash function generates a hash value (or simply a hash) that is stored in a hash table. A partition key is generated from the first field of a primary key. Data partitioned into hash tables using partition keys provides for rapid lookup. Fewer the partitions used for a query faster is the response time for the query.

- Denormalization: In relational database design, we are often taught the importance of normalization. This is not an advantage when working with Cassandra because it performs best when the data model is denormalized. It is often the case that companies end up denormalizing data in relational databases as well. There are two common reasons for this. One is performance. Companies simply can't get the performance they need when they have to do so many joins on years' worth of data, so they denormalize along the lines of known queries. This ends up working, but goes against the grain of how relational databases are intended to

be designed, and ultimately makes one question whether using a relational database is the best approach in these circumstances.

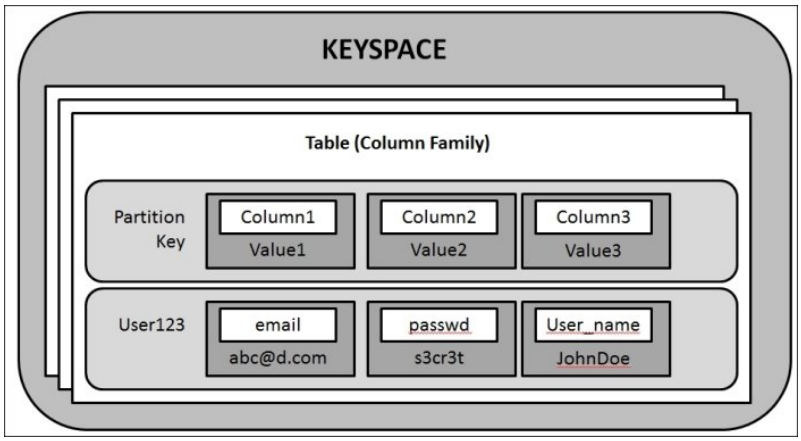


Figure 4.19: Cassandra's storage model.

Query Language

The Cassandra Query Language (CQL) is the primary language for communicating with the Apache Cassandra™ database.[61] The most basic way to interact with Apache Cassandra is using the CQL shell (shell is an interface to the operating system), cqlsh. Using cqlsh, we can create keyspaces and tables, insert and query tables, plus much more. We found CQL pretty easy to use compared to other alternatives presented in this thesis.

Security

There are three main components to the security features provided by Cassandra.[61]

- TLS/SSL (Transport Layer Security/Secure Sockets Layer) Encryption: Cassandra provides secure communication between a client machine and a database cluster and between nodes within a cluster. Enabling encryption ensures that data in flight is not compromised and is transferred securely. The options for client-to-node and node-to-node encryption are managed separately and may be configured independently.
- SSL (Secure Sockets Layer) Certificate Hot Reloading: Beginning with Cassandra 4, Cassandra supports hot reloading of SSL Certificates. If SSL/TLS support is enabled in Cassandra, the node periodically polls the Trust and Key Stores specified in cassandra.yaml. When the files are updated, Cassandra will reload them and use them for subsequent connections. Please note that the Trust & Key Store passwords are part of the yaml so the updated files should also use the same passwords. The default polling interval is 10 minutes.
- Client authentication and Authorization.

Transactions

Cassandra does not use RDBMS ACID transactions with rollback or locking mechanisms, but instead offers atomic, isolated, and durable transactions with eventual/tunable consistency that lets

the user decide how strong or eventual they want each transaction's consistency to be.[61]

As a non-relational database, Cassandra does not support joins or foreign keys, and consequently does not offer consistency in the ACID sense. For example, when moving money from account A to B the total in the accounts does not change. Cassandra supports atomicity and isolation at the row-level, but trades transactional isolation and atomicity for high availability and fast write performance. Cassandra writes are durable. In what follows we explain how Cassandra deals with ACID properties.

- Atomicity: In Cassandra, a write is atomic at the partition-level, meaning inserting or updating columns in a row is treated as one write operation. A delete operation is also performed atomically. Cassandra uses client-side timestamps to determine the most recent update to a column. The latest timestamp always wins when requesting data, so if multiple client sessions update the same columns in a row concurrently, the most recent update is the one seen by readers.

- Isolation: Full row-level isolation is in pace, which means that writes to a row are isolated to the client performing the write and are not visible to any other use until they are complete. Delete operations are performed in isolation. All updates in a batch operation belonging to a given partition key are performed in isolation.

- Durability: Writes in Cassandra are durable. All writes to a replica node are recorded both in memory and in a commit log on disk before they are acknowledged as a success. If a crash or server failure occurs before the memtables are flushed to disk, the commit log is replayed on restart to recover any lost writes. In addition to the local durability (data immediately written to disk), the replication of data on other nodes strengthens durability.

Documentation

We found Cassandra's documentation extremely integrated and easy to read. This is the link for the official documentation for Apache Cassandra: `http://cassandra.apache.org/doc`

4.4 MongoDB

Database

MongoDB is a source-available cross-platform document-oriented database program.[62][63] Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL). 10gen software company began developing MongoDB in 2007 as a component of a planned platform as a service product. In 2009, the company shifted to an open-source development model, with the company offering commercial support and other services. In 2013, 10gen changed its name to MongoDB Inc.MongoDB supports field, range query, and regular-expression searches.[64] Queries can return specific fields of documents and also include user-defined JavaScript functions. Queries can also be configured to return a random sample of results of a given size.

Architecture

As a definition, MongoDB is an open-source database that uses a document-oriented data model and a non-structured query language.[65] It is one of the most powerful NoSQL systems and databases around, today. MongoDB Atlas is a cloud database solution for contemporary applications that is available globally. This best-in-class automation and established practices offer to deploy fully managed MongoDB across AWS, Google Cloud, and Azure. It also ensures availability, scalability, and compliance with the most stringent data security and privacy requirements. MongoDB Cloud is a unified data platform that includes a global cloud database, search, data lake, mobile, and application services. Being a NoSQL tool means that it does not use the usual rows and columns that we so much associate with relational database management. MongoDB uses sharding techonology which is the process of breaking up large tables into smaller chunks called shards that are spread across multiple servers as we can see in Figure 4.20. Sharding is also being used in Neo4j as we have recently seen in Section 4.2, but it is not used in MySQL database system. MongoDB's architecture is basically an architecture that is built on collections and documents. The basic unit of data in this database consists of a set of key-value pairs. It allows documents to have different fields and structures. This database uses a document storage format called BSON which is a binary style of JSON documents.

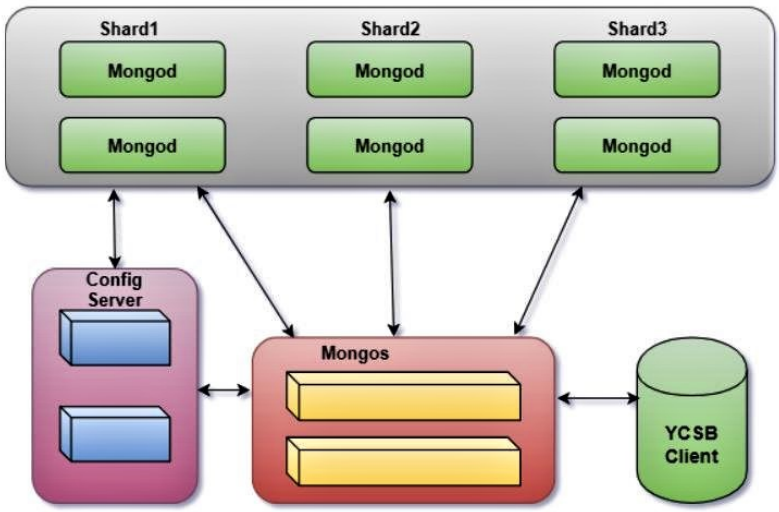


Figure 4.20: Representation of MongoDB's architecture.

Scalability

Partitioning distributes data across multiple nodes in a cluster.[65] Each replica set (known in MongoDB as a shard) in a cluster only stores a portion of the data based on a collection sharding key (sharding strategy), which determines the distribution of the data. We can have a look at a sharded cluster's components in Figure 4.21. This makes it possible to scale the storage capacity of the cluster virtually without limit. Since each node is only responsible for processing the data it stores, overall processing capacity for both reads and writes is increased as well. Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations. Database systems with large data sets or high throughput applications can challenge the capacity of a single server. For example,

high query rates can exhaust the CPU capacity of the server. Working set sizes larger than the system's RAM stress the I/O capacity of disk drives. MongoDB supports horizontal scaling through sharding. Horizontal scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required. While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server. Expanding the capacity of the deployment only requires adding additional servers as needed, which can be a lower overall cost than high-end hardware for a single machine. The trade off is increased complexity in infrastructure and maintenance for the deployment.

A MongoDB sharded cluster consists of the following components.

- Shard: Each shard contains a subset of the sharded data, which can be deployed as a replica set.
- Mongos: The mongos acts as a query router, providing an interface between client applications and the sharded cluster.
- Config servers: Config servers store metadata and configuration settings for the cluster.

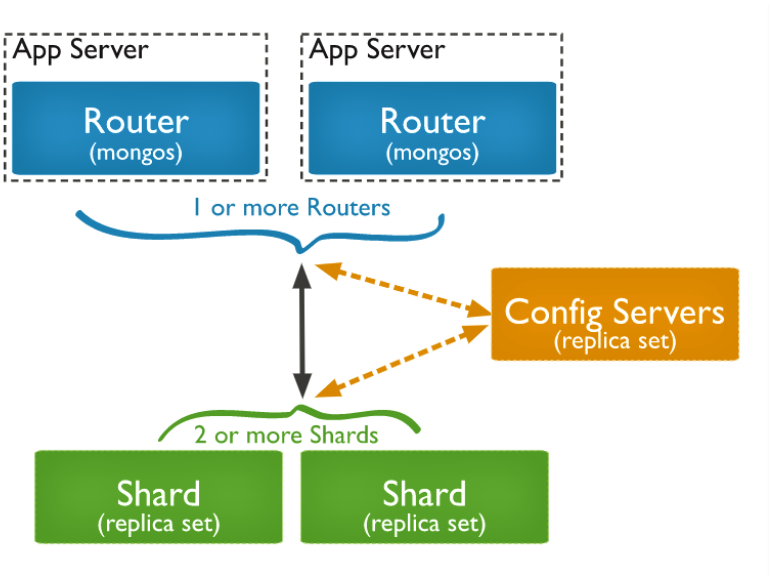


Figure 4.21: The interaction of components within a sharded cluster.

Sharded cluster infrastructure requirements and complexity require careful planning, execution, and maintenance. Once a collection has been sharded, MongoDB provides no method to unshard a sharded collection.

Replication(Nodes, Internode Communication)

MongoDB achieves replication by the use of replica set.[65] A replica set (or cluster) is a group of mongod instances that host the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments. In a replica, one node is primary node that receives all write operations. We can see how MongoDB's replication works in Figure 4.22.

All other instances, such as secondaries, apply operations from the primary so that they have the same data set. Replica set can have only one primary node.

- Replica set is a group of two or more nodes (generally minimum 3 nodes are required).
- In a replica set, one node is primary node and remaining nodes are secondary.
- All data replicates from primary to secondary node.
- At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
- After the recovery of failed node, it again join the replica set and works as a secondary node.

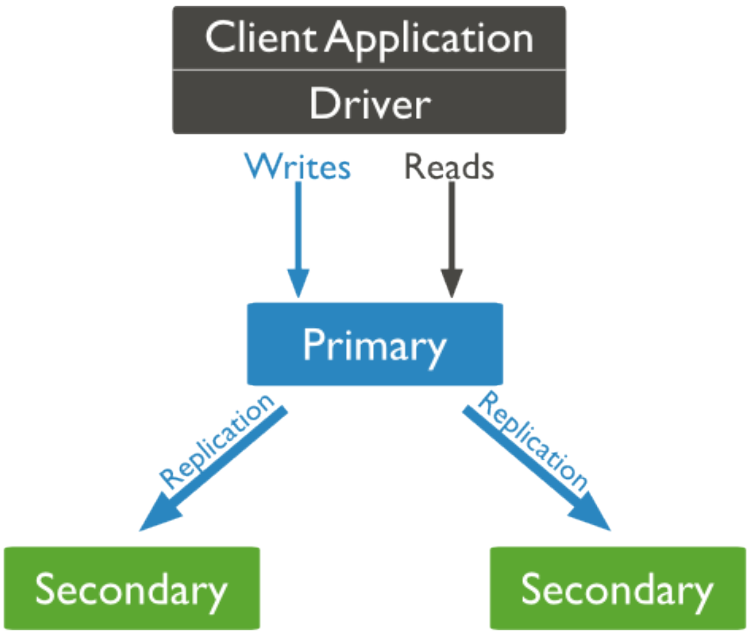


Figure 4.22: A typical diagram of MongoDB's replication in which client application always interact with the primary node and the primary node then replicates the data to the secondary nodes.

MongoDB supports TLS/SSL (Transport Layer Security/Secure Sockets Layer) to encrypt all of MongoDB's network traffic. TLS/SSL ensures that MongoDB network traffic is only readable by the intended client. MongoDB can use any valid TLS/SSL certificate issued by a certificate authority or a self-signed certificate. Forward Secrecy (FS) cipher suites (which is actually an encryption system that changes the keys used to encrypt and decrypt information frequently and automatically) create an ephemeral session key that is protected by the server's private key but is never transmitted. The use of an ephemeral key ensures that even if a server's private key is compromised, we cannot decrypt past sessions with the compromised key.

Data Models

Unlike SQL databases, where we must determine and declare a table's schema before inserting data, MongoDB's collections, by default, do not require their documents to have the same schema.[65] That is:

- The documents in a single collection do not need to have the same set of fields and the data type for a field can differ across documents within a collection.

- To change the structure of the documents in a collection, such as add new fields, remove existing fields, or change the field values to a new type, update the documents to the new structure.

This flexibility facilitates the mapping of documents to an entity or an object. Each document can match the data fields of the represented entity, even if the document has substantial variation from other documents in the collection. Starting in version 3.6, MongoDB supports JSON Schema validation.

MongoDB unlike other SQL databases, uses a *Document Structure*. The key decision in designing data models for MongoDB applications revolves around the structure of documents and how the application represents relationships between data. MongoDB allows related data to be embedded within a single document. Below, we discuss how embedded documents and references work.

- Embedded documents capture relationships between data by storing related data in a single document structure as it is shown in Figure 4.23. MongoDB documents make it possible to embed document structures in a field or array within a document. These denormalized data models allow applications to retrieve and manipulate related data in a single database operation.

Figure 4.23: Embedded documents, example.

- References store the relationships between data by including links or references from one document to another as we can see in Figure 4.24. Applications can resolve these references to access the related data. Broadly, these are normalized data models.

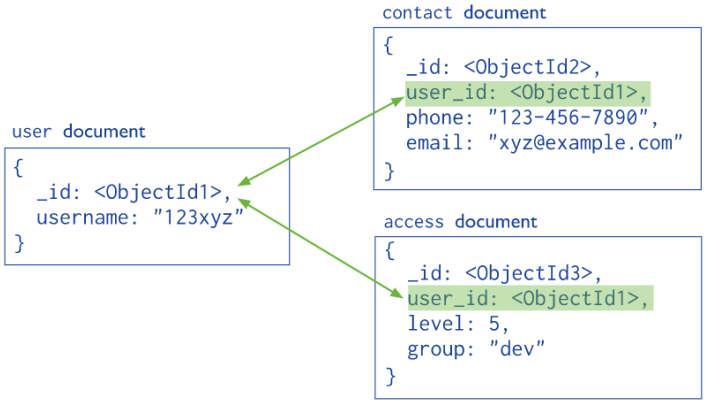


Figure 4.24: References, example.

For many use cases in MongoDB, the denormalized data model is optimal, unlike MySQL (normalized and denormalized data definitions are provided in our datasheet in Section 3.2.2). MongoDB unlike other SQL systems embed related data in a single structure or document. Denormalization actually allows MongoDB to avoid some application-level joins, at the expense of having more complex and expensive updates.

Support

MongoDB can back up with *Atlas* which is the hosted MongoDB service option in the cloud.[65] MongoDB offers the following two fully-managed methods for backups.

- Cloud Backups, which utilize the native snapshot functionality of the deployment's cloud service provider to offer robust backup options. Cloud Backups provide on-demand snapshots, which allow us to trigger an immediate snapshot of our deployment at a given point in time and continuous Cloud Backups, which allow us to schedule recurring backups for our deployment.
- Legacy Backups (Deprecated), which take incremental backups of data in our deployment.

MongoDB can also back up with *Cloud Manager* which is a hosted back up, monitoring, and automation service for MongoDB. MongoDB Cloud Manager supports backing up and restoring MongoDB replica sets and sharded clusters from a graphical user interface as follows.

- The MongoDB Cloud Manager supports the backing up and restoring of MongoDB deployments. MongoDB Cloud Manager continually backs up MongoDB replica sets and sharded clusters by reading the oplog data from our MongoDB deployment. MongoDB Cloud Manager creates snapshots of our data at set intervals, and can also offer point-in-time recovery of MongoDB replica sets and sharded clusters.
- With Ops Manager, MongoDB subscribers can install and run the same core software that powers MongoDB Cloud Manager on their own infrastructure. Ops Manager is an on-premise solution that has similar functionality to MongoDB Cloud Manager and is available with Enterprise Advanced subscriptions.

Security

Data in MongoDB is protected with preconfigured security features for authentication, authorization, encryption, and more as we can see in Figure 4.25.[65]

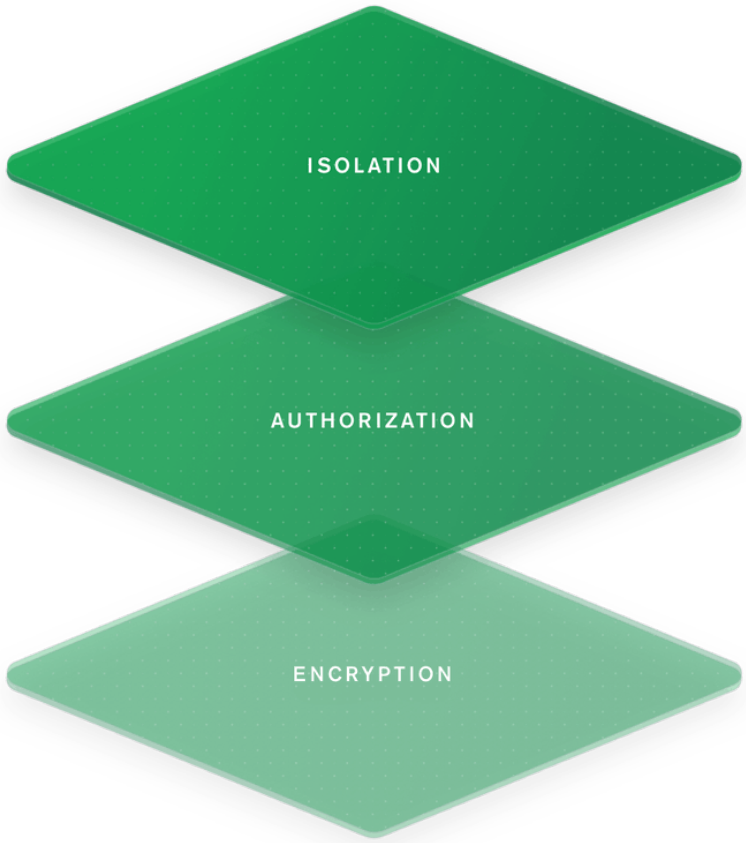


Figure 4.25: Isolation, Authorization, Encryption.

MongoDB includes extensive capabilities to defend, detect, and control access to data as follows.

- Authentication: MongoDB offers a strong challenge-response mechanism based on SCRAM-256, along with integration to enterprise security infrastructure, including LDAP, Windows Active Directory, Kerberos, x.509 certificates, and AWS IAM.
- Network isolation: For users running fully managed databases in MongoDB Atlas, user data and underlying systems are fully isolated from other users. Database resources are associated with a user group, which is contained in its own virtual private cloud (VPC). Access can be granted only by IP whitelisting or VPC peering.
- Authorization: Role-based access control (RBAC) enables us to configure granular permissions for a user or application based on the privileges they need to do their jobs.
- Auditing: For regulatory compliance, security administrators can use MongoDB's native audit log to record all database activity and changes.

- Encryption everywhere: MongoDB data can be encrypted while in motion across the network, while in use in the database, and while at rest, whether on disk or in backups. With client-side field-level encryption(FLE), we have access to some of the most advanced data protection controls anywhere.

We conclude that MongoDB uses a variety of authentication and encryption security measures just like MySQL as we have recently seen in Section 4.1. MongoDB also supports TLS/SSL(Transport Layer Security/Secure Sockets Layer) to encrypt all of MongoDB's network traffic.

Transactions

MongoDB added support for multi-document ACID transactions in version 4.0 in 2018 and extended that support for distributed multi-document ACID transactions in version 4.2 in 2019.[65] The following diagram 4.26 describes the updates made from MongoDB 3.0 to MongoDB 4.2. We can see that MongoDB satisfies the relational database's ACID properties after that specific update. MongoDB's document model allows related data to be stored together in a single document. The document model, combined with atomic document updates, obviates the need for transactions in a majority of use cases. Nonetheless, there are cases where true multi-document, multi-collection MongoDB transactions are the best choice. MongoDB transactions work similarly to transactions in other databases. To use a transaction, we start a MongoDB session through a driver. Then, we use that session to execute our group of database operations. We can run any of the CRUD (create, read, update, and delete) operations across multiple documents, multiple collections, and multiple shards.

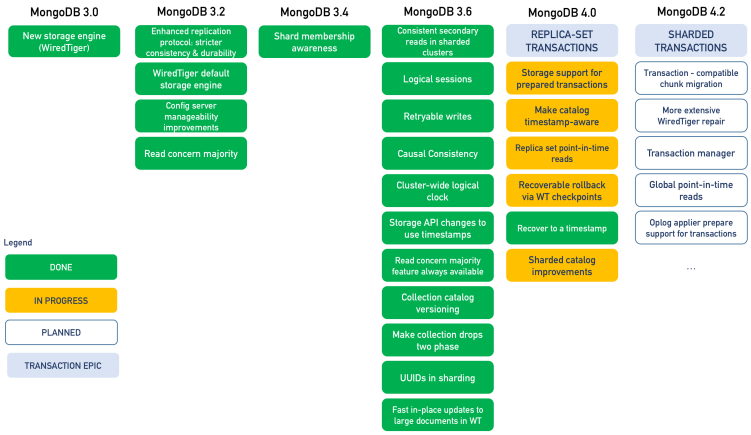


Figure 4.26: This diagram summarizes the ACID transactions updates from MongoDB 3.0 to MongoDB 4.2.

Documentation

We found MongoDB's quite well structured and complete in every way. This is the link for the official documentation for MongoDB: https://docs.mongodb.com

4.5 Redis

Database

Redis is an in-memory data structure store, used as a distributed, in-memory key–value database, cache and message broker, with optional durability. Redis released in May 10 2009.[66][67] Redis supports different kinds of abstract data structures, such as strings, lists, maps, sets, sorted sets, HyperLogLogs, bitmaps, streams, and spatial indices. The name Redis means Remote Dictionary Server. The Redis project began when Salvatore Sanfilippo, nicknamed antirez, the original developer of Redis, was trying to improve the scalability of his Italian startup, developing a real-time web log analyzer.[68][69][70] After encountering significant problems in scaling some types of workloads using traditional database systems, Sanfilippo began to prototype a first proof of concept version of Redis in Tcl. Later Sanfilippo translated that prototype to the C language and implemented the first data type, the list. After a few weeks of using the project internally with success, Sanfilippo decided to open source it, announcing the project on Hacker News. The project began to get traction, particularly among the Ruby community, with GitHub and Instagram being among the first companies adopting it.

Architecture

Redis employs a primary-replica architecture and supports asynchronous replication where data can be replicated to multiple replica servers.[71][72] This provides improved read performance (as requests can be split among the servers) and faster recovery when the primary server experiences an outage. For persistence, Redis supports point-in-time backups (copying the Redis data set to disk).

A Redis cluster is composed of identical nodes that are deployed within a data center or stretched across local availability zones. Redis Enterprise architecture is made up of a management path and data access path:

- Management path includes the cluster manager, proxy and secure REST API/UI for programmatic administration. In short, cluster manager is responsible for orchestrating the cluster, placement of database shards as well as detecting and mitigating failures. Proxy helps scale connection management.

- Data Access path is composed of master and replica Redis shards as it is shown in Figure 4.27. Clients perform data operations on the master shard. Master shards maintain replica shards using the in-memory replication for protection against failures that may render master shard inaccessible.

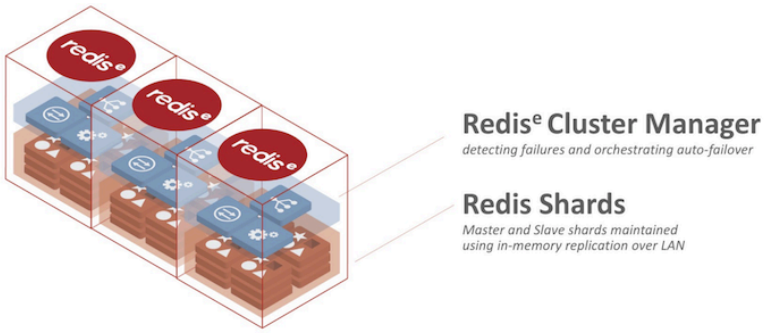


Figure 4.27: Redis Nodes with blue layer representing the management path and red tiles representing the data access path with Redis as the shards.

Scalability

When we scale using the offline process, our cluster is offline for a significant portion of the process and thus unable to serve requests.[71] When we scale using the online method, because scaling is a compute-intensive operation, there is some degradation in performance, nevertheless, our cluster continues to serve requests throughout the scaling operation. How much degradation we experience depends upon our normal CPU utilization and our data.

There are two ways to scale our Redis (cluster mode enabled) cluster, horizontal and vertical scaling as follows.

- Horizontal scaling allows us to change the number of node groups (shards) in the replication group by adding or removing node groups (shards). The online resharding process allows scaling in/out while the cluster continues serving incoming requests.
- Vertical Scaling change the node type to resize the cluster. The online vertical scaling allows scaling up/down while the cluster continues serving incoming requests.

Replication(Nodes, Internode Communication)

At the base of Redis replication there is a very simple to use and configure leader follower (master-replica) replication which allows replica Redis instances to be exact copies of master instances.[72] The replica will automatically reconnect to the master every time the link breaks, and will attempt to be an exact copy of it regardless of what happens to the master. In a typical Redis Cluster, one master has multitype slaves attached as it is shown in Figure 4.28. When data will be updated on this master, the master will push those changes to its replicas This system works by using the following three main mechanisms.

- When a master and a replica instances are well-connected, the master keeps the replica updated by sending a stream of commands to the replica, in order to replicate the effects on the dataset happening in the master side due to: client writes, keys expired or evicted, any other action changing the master dataset.
- When the link between the master and the replica breaks, for network issues or because a timeout is sensed in the master or the replica, the replica reconnects and attempts to proceed

with a partial resynchronization: it means that it will try to just obtain the part of the stream of commands it missed during the disconnection.

- When a partial resynchronization is not possible, the replica will ask for a full resynchronization. This will involve a more complex process in which the master needs to create a snapshot of all its data, send it to the replica, and then continue sending the stream of commands as the dataset changes.

Redis uses by default asynchronous replication, which being low latency and high performance, is the natural replication mode for the vast majority of Redis use cases. However, Redis replicas asynchronously acknowledge the amount of data they received periodically with the master. So the master does not wait every time for a command to be processed by the replicas, however it knows, if needed, what replica already processed what command. This allows having optional synchronous replication.

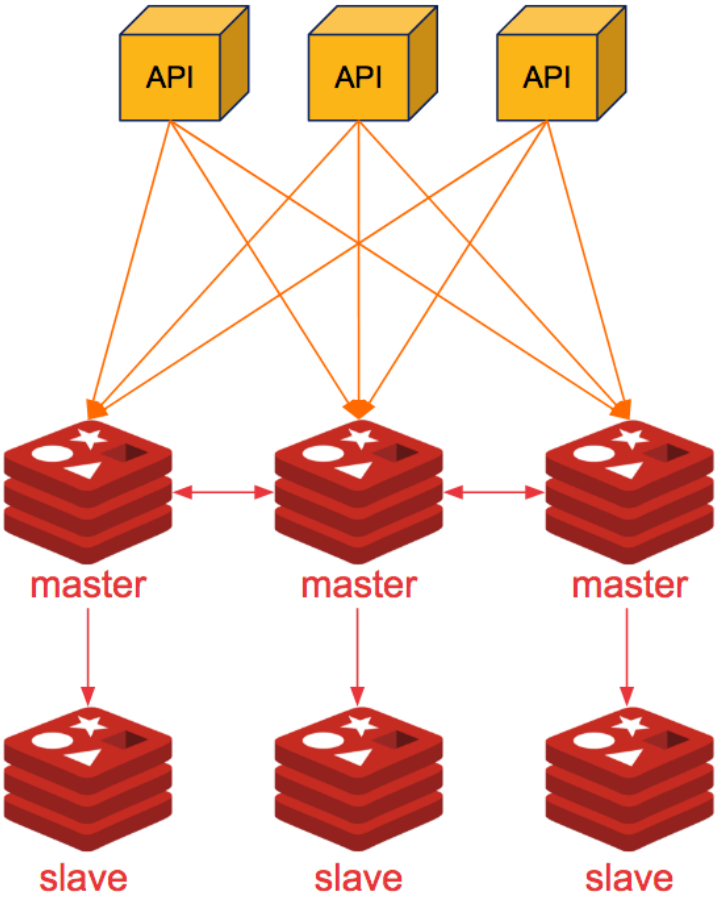


Figure 4.28: A typical Redis Cluster.

Redis supports internode encryption, which encrypts internal communication between nodes as we can see in Figure 4.29. This improves the security of data as it travels within a cluster. Internode encryption is enabled for the control plane, which manages the cluster and its databases. Internode encryption is supported for the data plane, which encrypts communication used to replicate shards

between nodes and proxy communication with shards located on different nodes. Redis supports TLS/SSL (Transport Layer Security/Secure Sockets Layer).

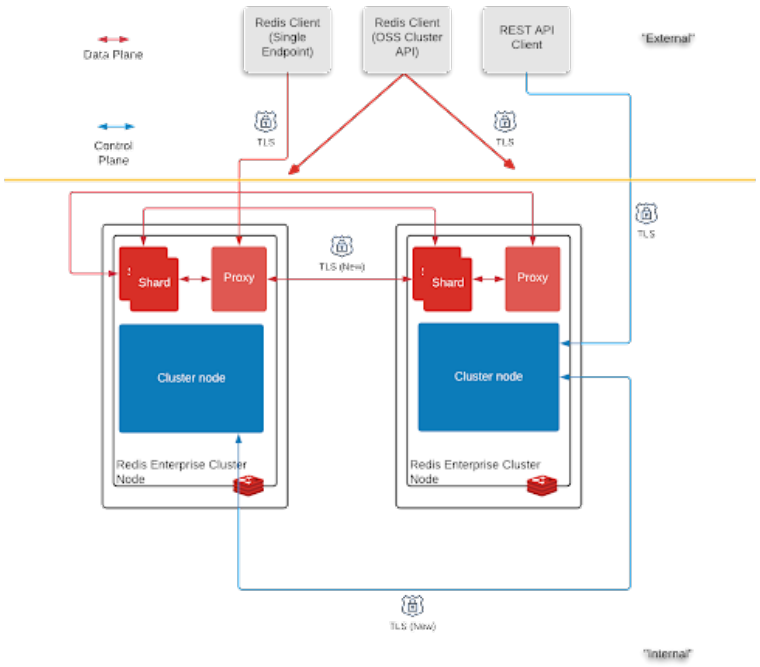


Figure 4.29: Internode encryption in Redis.

Data Models

Redis is not a plain key-value store, it is actually a data structures server, supporting different kinds of values.[72] What this means is that, while in traditional key-value stores we associate string keys to string values, in Redis the value is not limited to a simple string, but can also hold more complex data structures. An example of a data structure in Redis is shown in Figure 4.30. Redis has a relatively rich set of data types when compared to many key-value data stores. The following is the list of all the data structures supported by Redis.

- Binary-safe strings.
- Lists: collections of string elements sorted according to the order of insertion. They are basically linked lists.
- Sets: collections of unique, unsorted string elements.
- Sorted sets, similar to Sets but where every string element is associated to a floating number value, called score. The elements are always taken sorted by their score, so unlike Sets it is possible to retrieve a range of elements.
- Hashes, which are maps composed of fields associated with values. Both the field and the value are strings. This is very similar to Ruby or Python hashes.
- Bit arrays (or simply bitmaps): it is possible, using special commands to handle String values like an array of bits.

- HyperLogLogs: this is a probabilistic data structure which is used in order to estimate the cardinality of a set
- Streams: append-only collections of map-like entries that provide an abstract log data type.

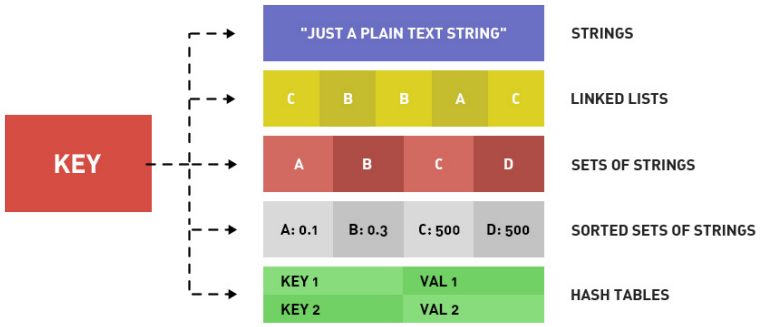


Figure 4.30: Data structure example in Redis.

We can see that Redis uses a wide variety of data types, in contrast with MySQL which mainly uses numeric types, date and time types, string (character and byte) types, spatial types and the JSON data type.

Query Language

As a NoSQL database, Redis doesn't use structured query language, otherwise known as SQL.[72] Redis instead comes with its own set of commands for managing and accessing data. Redis uses RediSearch which is a source available secondary index, query engine and full-text Search over Redis, developed by Redis. Redisearch implements a secondary index on top of Redis, but unlike other Redis indexing libraries, it does not use internal data structures such as sorted sets. This also enables more advanced features, such as multi-field queries, aggregation, and full text search capabilites. These capabilities include exact phrase matching and numeric filtering for text queries.

Support

Redis provides a different range of persistence options.[72]

- RDB(Redis Database): The RDB persistence performs point-in-time snapshots of our dataset at specified intervals as we can see in Figure 4.31.
- AOF (Append Only File): The AOF persistence logs every write operation received by the server, that will be played again at server startup, reconstructing the original dataset. Commands are logged using the same format as the Redis protocol itself, in an append-only fashion. Redis is able to rewrite the log in the background when it gets too big.
- No persistence: If we wish, we can disable persistence completely, if we want our data to just exist as long as the server is running.

- RDB + AOF: It is possible to combine both AOF and RDB in the same instance. Notice that, in this case, when Redis restarts the AOF file will be used to reconstruct the original dataset since it is guaranteed to be the most complete.

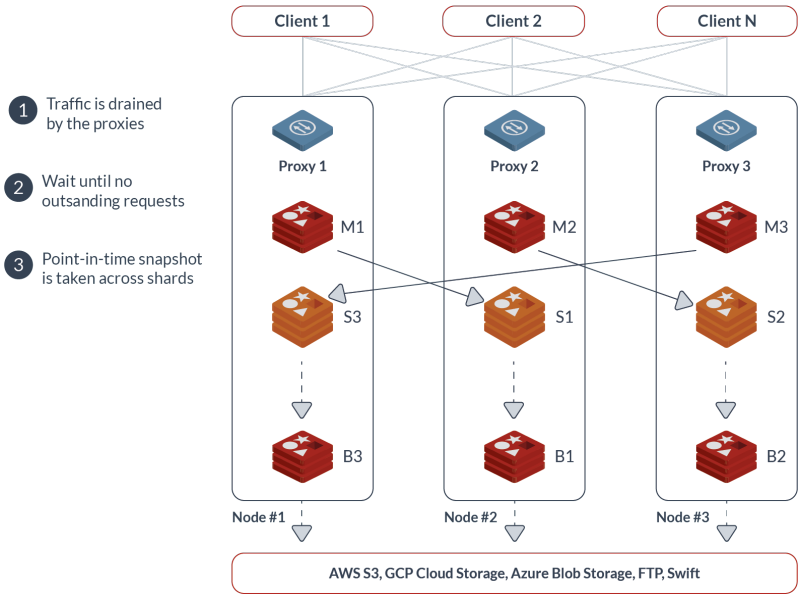


Figure 4.31: Illustration of the backup process of Redis.

Security

Redis is designed to be accessed by trusted clients inside trusted environments.[72] This means that usually it is not a good idea to expose the Redis instance directly to the internet or, in general, to an environment where untrusted clients can directly access the Redis TCP port or UNIX socket.

Furthermore, Redis provides network security, in which access to the Redis port should be denied to everybody but trusted clients in the network, so the servers running Redis should be directly accessible only by the computers implementing the application using Redis.

Unfortunately many users fail to protect Redis instances from being accessed from external networks. Many instances are simply left exposed on the internet with public IPs. For this reasons a special mode enters called *protected mode*. In this mode Redis only replies to queries from the loopback interfaces, and reply to other clients connecting from other addresses with an error, explaining what is happening and how to configure Redis properly.

When the authorization layer is enabled, Redis will refuse any query by unauthenticated clients. A client can authenticate itself by sending the AUTH command followed by the password.

Redis has optional support for TLS (Transport Layer Security) on all communication channels, including client connections, replication links and the Redis Cluster bus protocol.

We can see that Redis provides decent security measures for the protection of a Redis instance, through specific configurations, such as network security, authorization layer, TLS, and *protected mode*.

Transactions

Redis guarantees ACID-ish (Atomicity, Consistency, Isolation, Durability) operations by merely creating a single Redis instance with a *master* role and having it configured with AOF every write (*appendfsync always*) to a persistent storage device.[73] This configuration provides ACID characteristics in the following ways.

- Atomicity: indivisible and irreducible characteristics are achieved using Redis transaction commands or Lua scripting. *All or nothing* property is almost always achieved, excluding cases like OOM (Out Of Memory) or a buggy Lua script.
- Consistency: Redis validates that any write operation affects the target data-structure in allowed ways, and any programming errors cannot result in the violation of any defined rules
- Isolation: at the command level, isolation is achieved due to the fact that Redis is single threaded, for multiple operations, isolation can be achieved using transactions or inside a Lua script.
- Durability: with AOF (Append Only File) every write, Redis replies to the client after the write operation has been successfully written to disk, guaranteeing durability. In addition, we should connect our persistent storage to our network so that data can be easily recovered by attaching a new node to the same persistent storage used by the failed node.

We conclude that Redis satisfies ACID transactions only through the usage of AOF (Append Only File) configuration.

Documentation

Documentation in the main website of Redis covered all our criteria we were looking for. This is the link for the official documentation for Redis: `https://redis.io/documentation`

Chapter 5

System Comparison

In this chapter we extensively study and eventually compare the database systems presented in Chapter 3 based on their characteristics nad evaluation criteria.

5.1 Testing based on the Evaluation Criteria

In this section we are going to test our database systems introduced in Chapter 3 based on the evaluation criteria presented in Section 3.2 (data visualization, performance on read & write capability, other features).

For this testing we are going to use on all our database systems the datasheet provided in Subsection 3.2.2. Then we will evaluate and monitor the performance of these database systems on their data visualization capabilities, read & write performance and data manipulation.

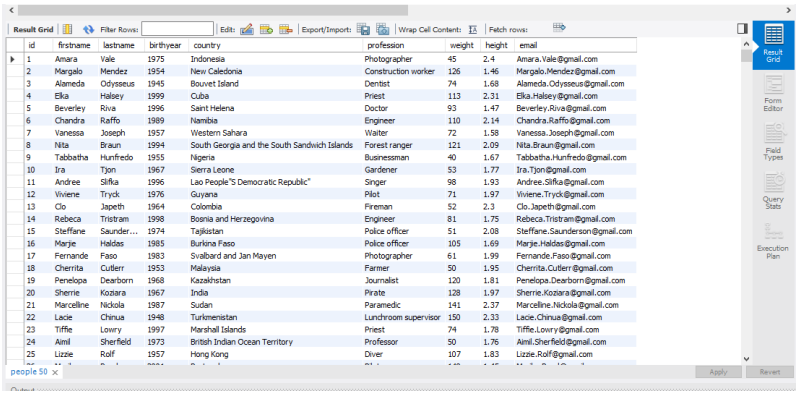
5.1.1 MySQL

For our tests we are using the MySQL Workbench 8.0.27. which is a unified visual tool for database architects, developers, and DBAs (Doing Business As).

Database Visualization

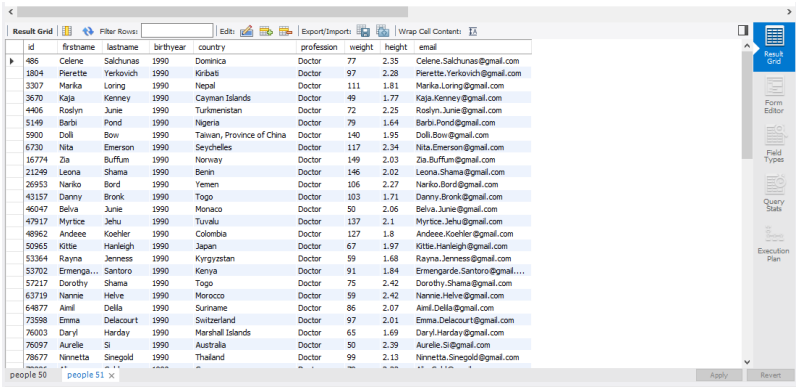
MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more. It is a free, DBMS-independent, cross-platform SQL query tool. Data in MySQL Workbench can be visualized in a table form, as shown in the example presented in Figure 5.1.

In Figure 5.2 we can see how this query example transfers to the screen:
SELECT * FROM people WHERE birthyear=1990 AND profession='Doctor' LIMIT 200000;



id	firstname	lastname	birthyear	country	profession	weight	height	email
1	Amara	Vale	1975	Indonesia	Photographer	45	2.4	Amara.Vale@gmail.com
2	Margalo	Mendez	1954	New Caledonia	Construction worker	126	1.46	Margalo.Mendez@gmail.com
3	Alameda	Odysseus	1945	Bouvet Island	Dentist	74	1.68	Alameda.Odysseus@gmail.com
4	Elka	Hakley	1999	Cuba	Priest	113	2.31	Elka.Hakley@gmail.com
5	Beverley	Riva	1996	Saint Helena	Doctor	93	1.47	Beverley.Riva@gmail.com
6	Chandra	Raffo	1989	Namibia	Engineer	110	2.14	Chandra.Raffo@gmail.com
7	Vanessa	Joseph	1957	Western Sahara	Waiter	72	1.58	Vanessa.Joseph@gmail.com
8	Nita	Braun	1994	South Georgia and the South Sandwich Islands	Forest ranger	121	2.09	Nita.Braun@gmail.com
9	Tabbatha	Hunfredo	1955	Nigeria	Businessman	40	1.67	Tabbatha.Hunfredo@gmail.com
10	Ira	Tjon	1967	Sierra Leone	Gardener	53	1.77	Ira.Tjon@gmail.com
11	Andree	Slifka	1996	Lao People'S Democratic Republic"	Singer	98	1.93	Andree.Slifka@gmail.com
12	Viviene	Tryck	1976	Guyana	Pilot	71	1.97	Viviene.Tryck@gmail.com
13	Clo	Japeth	1964	Colombia	Fireman	52	2.3	Clo.Japeth@gmail.com
14	Rebeca	Tristram	1998	Bosnia and Herzegovina	Engineer	81	1.75	Rebeca.Tristram@gmail.com
15	Steffane	Saunder...	1974	Tajikistan	Police officer	51	2.08	Steffane.Saunderson@gmail.com
16	Marje	Haldas	1985	Burkina Faso	Police officer	105	1.69	Marje.Haldas@gmail.com
17	Fernande	Faso	1983	Svalbard and Jan Mayen	Photographer	61	1.99	Fernande.Faso@gmail.com
18	Cherrita	Cutlerr	1953	Malaysia	Farmer	50	1.95	Cherrita.Cutlerr@gmail.com
19	Penelopa	Dearborn	1968	Kazakhstan	Journalist	120	1.81	Penelopa.Dearborn@gmail.com
20	Sherrie	Kozara	1967	India	Pirate	128	1.97	Sherrie.Kozara@gmail.com
21	Marcelline	Nickola	1987	Sudan	Paramedic	141	2.37	Marcelline.Nickola@gmail.com
22	Lacie	Chinua	1948	Turkmenistan	Lunchroom supervisor	150	2.33	Lacie.Chinua@gmail.com
23	Tiffie	Lowry	1997	Marshall Islands	Priest	74	1.78	Tiffie.Lowry@gmail.com
24	Aimil	Sherfield	1973	British Indian Ocean Territory	Professor	50	1.76	Aimil.Sherfield@gmail.com
25	Lizzie	Rolf	1957	Hong Kong	Diver	107	1.83	Lizzie.Rolf@gmail.com

Figure 5.1: Example of how data is being visualized in a table form.



id	firstname	lastname	birthyear	country	profession	weight	height	email
486	Celene	Salchunas	1990	Dominica	Doctor	77	2.35	Celene.Salchunas@gmail.com
1804	Pierette	Yerkovich	1990	Kiribati	Doctor	97	2.28	Pierette.Yerkovich@gmail.com
3307	Marika	Lorring	1990	Nepal	Doctor	111	1.81	Marika.Lorring@gmail.com
3670	Kaja	Kenney	1990	Cayman Islands	Doctor	49	1.77	Kaja.Kenney@gmail.com
4406	Roslyn	Junie	1990	Turkmenistan	Doctor	72	2.25	Roslyn.Junie@gmail.com
5149	Barbi	Pond	1990	Nigeria	Doctor	79	1.64	Barbi.Pond@gmail.com
5900	Dolli	Bow	1990	Taiwan, Province of China	Doctor	140	1.95	Dolli.Bow@gmail.com
6730	Nita	Emerson	1990	Seychelles	Doctor	117	2.34	Nita.Emerson@gmail.com
16774	Zia	Buffum	1990	Norway	Doctor	149	2.03	Zia.Buffum@gmail.com
21249	Leona	Shama	1990	Benin	Doctor	146	2.02	Leona.Shama@gmail.com
26953	Nariko	Bord	1990	Yemen	Doctor	106	2.27	Nariko.Bord@gmail.com
43157	Danny	Bronk	1990	Togo	Doctor	103	1.71	Danny.Bronk@gmail.com
46047	Belva	Junie	1990	Monaco	Doctor	50	2.06	Belva.Junie@gmail.com
47917	Myrtice	Jehu	1990	Tuvalu	Doctor	137	2.1	Myrtice.Jehu@gmail.com
48962	Andeee	Koehler	1990	Colombia	Doctor	127	1.8	Andeee.Koehler@gmail.com
50965	Kittie	Hanleigh	1990	Japan	Doctor	67	1.97	Kittie.Hanleigh@gmail.com
53364	Rayna	Jenness	1990	Kyrgyzstan	Doctor	59	1.68	Rayna.Jenness@gmail.com
53702	Ermenga...	Santoro	1990	Kenya	Doctor	91	1.84	Ermengarde.Santoro@gmail....
57217	Dorothy	Shama	1990	Togo	Doctor	75	2.42	Dorothy.Shama@gmail.com
63719	Nannie	Helvie	1990	Morocco	Doctor	59	2.42	Nannie.Helvie@gmail.com
64877	Aimil	Della	1990	Suriname	Doctor	86	2.07	Aimil.Della@gmail.com
73598	Emma	Delacourt	1990	Switzerland	Doctor	97	2.01	Emma.Delacourt@gmail.com
76003	Daryl	Harday	1990	Marshall Islands	Doctor	65	1.69	Daryl.Harday@gmail.com
76097	Aurelie	Si	1990	Australia	Doctor	50	2.39	Aurelie.Si@gmail.com
78677	Ninnetta	Sinegold	1990	Thailand	Doctor	99	2.13	Ninnetta.Sinegold@gmail.com

Figure 5.2: A query example.

After a query, MySQL Workbench also provides a *Query Stats* table, which can be shown in the following figure (Figure 5.3).

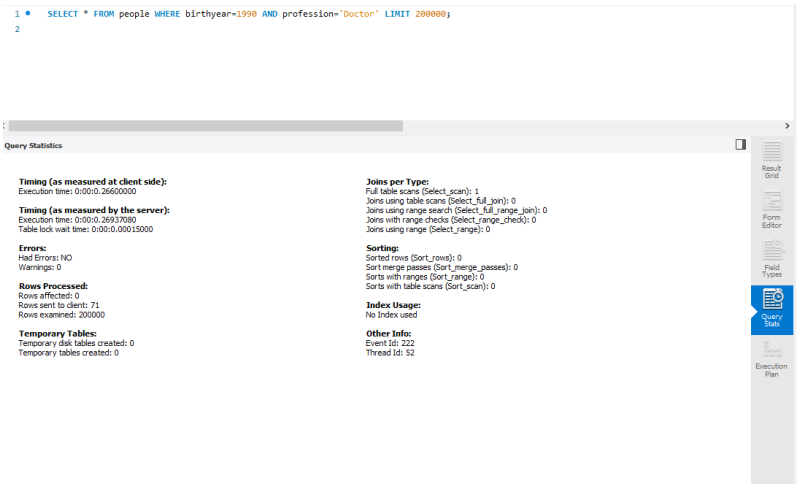


Figure 5.3: Query stats table.

Performance on Read/Write Speed

MySQL as a relational database system, supports both relational data model and flat data type and features tons of optimizations and indexing capabilities, allowing for huge performance increases. MySQL Workbench also uses the Table Data Export Wizard, which supports import and export operations using CSV and JSON files and includes several configuration options (separators, column selection, encoding selection and more). The wizard can be performed against local or remotely connected MySQL servers and the import action includes table, column, and type mapping.

- Flat file importation and querring:
  - The flat file importation took 2973.579s to complete (which is the average import time of 10 tests for this file as we discussed in Subsection 3.2.2). The datasheet is provided in Subsection 3.2.2. Below we can see one of our shell notifications:
    - File C:\people.csv was imported in 3032.938s
      Table peopledb.people has been used
      200000 records imported
  - For the read speed we used 6 different queries and we counted the average query execution time. These are the MySQL queries we have used (query execution time in these tests are indications of the time the tests were performed):
    - SELECT * FROM people WHERE country='France' AND profession= 'Doctor' OR profession= 'Pilot' LIMIT 200000;
      Shell notification: Execution time 0.220 sec
    - SELECT AVG(weight), AVG(height) FROM people LIMIT 200000;
      Shell notification: Execution time 0.125 sec
    - SELECT * FROM people WHERE birthyear=1990 AND profession='Doctor' LIMIT 200000;
      Shell notification: Execution time 0.261 sec
    - SELECT * FROM people WHERE birthyear>1990 AND weight<68 LIMIT 200000;
      Shell notification: Execution time 0.219 sec
    - SELECT * FROM people WHERE birthyear>1990 AND weight<68 AND weight>50 LIMIT 200000;
      Shell notification: Execution time 0.187 sec
    - SELECT COUNT(id) FROM people WHERE profession='Engineer' AND weight>=100 LIMIT 200000;
      Shell notification: Execution time 0.218 sec

    The average query time of all 6 queries is 205ms (we took an average query time of 10 tests for each of our 6 queries as we discussed in Subsection 3.2.2).

- Relational data files importation and querring:

  – For this part we imported 5 files (which is the average import time of 10 tests for each file as we discussed in Subsection 3.2.2). The datasheet is provided in Subsection 3.2.2. Below we can see some example shell notifications:

    * File C:\people_related.csv was imported in 2743.352 s
      Table peopledb.people has been used
      200000 records imported
    * File C:\Country was imported in 0.652s
      Table peopledb.country has been used
      243 records imported
    * File C:\Profession was imported in 0.109s
      Table peopledb.profession has been used
      55 records imported
    * File C:\Lives_in was imported in 3122.637s
      Table peopledb.lives_in has been used
      200000 records imported
    * File C:\Works_as was imported in 3253.676s
      Table peopledb.works_as has been used
      200000 records imported

    The average relational data files importation time took 8958.352s to complete.

  – For the read speed we used 5 different queries and we counted the average query execution time. It is important to point out that we used indexes in these queries. A database index is a redundant copy of some of the data in the database for the purpose of making searches of related data more efficient. These are the 5 SQL queries we have used (query execution time in these tests are indications of the time the tests were performed):

    * SELECT firstname, lastname, birthyear, weight, height, email, profession, country FROM peoplerel, works_as,profession,country,lives_in WHERE peoplerel.person_id = works_as.person_id AND works_as.profession_id = profession.profession_id AND peoplerel.person_id = lives_in.person_id AND lives_in.country_id = country.country_id AND profession='Doctor' AND birthyear='1990'
      Shell notification: Duration: 0.109s

    * SELECT firstname, lastname, birthyear, weight, height, email, profession, country FROM peoplerel, works_as,profession,country,lives_in WHERE peoplerel.person_id = works_as.person_id AND works_as.profession_id = profession.profession_id AND peoplerel.person_id = lives_in.person_id AND lives_in.country_id = country.country_id AND profession='Doctor' AND birthyear>1995 AND weight>50 AND weight<68
      Shell notification: Duration: 0.107s

    * SELECT firstname, lastname, birthyear, weight, height, email, profession, country FROM peoplerel, works_as,profession,country,lives_in WHERE peoplerel.person_id

= works_as.person_id AND works_as.profession_id = profession.profession_id AND peoplerel.person_id = lives_in.person_id AND lives_in.country_id = country.country_id AND (profession='Doctor' OR profession ='pilot') AND birthyear>1990 AND weight>50 AND weight<68
Shell notification: Duration: 0.219s

* SELECT avg(weight), avg(height) FROM peoplerel,works_as,profession,country,lives_in WHERE peoplerel.person_id = works_as.person_id AND works_as.profession_id = profession.profession_id AND peoplerel.person_id = lives_in.person_id AND lives_in.country_id = country.country_id AND (profession='Doctor' OR profession ='pilot') AND birthyear>1990 AND weight>50 AND weight<68
Shell notification: Duration: 0.218s

* SELECT firstname, lastname, birthyear, weight, height, email, profession, country FROM peoplerel, works_as,profession,country,lives_in WHERE peoplerel.person_id = works_as.person_id AND works_as.profession_id = profession.profession_id AND peoplerel.person_id = lives_in.person_id AND lives_in.country_id = country.country_id AND (profession='Doctor' OR profession ='pilot') AND birthyear>1990 ORDER BY weight LIMIT 1
Shell notification: Duration: 0.188s

The average query time of all 5 queries is 168,6ms (we took an average query time of 10 tests for each of our 5 queries as we discussed in Subsection 3.2.2).

Other Features

MySQL allows a command to alter the column definition such as name and type according to our needs. We can do this with the help of an ALTER TABLE statement. We can also add new columns to our table with the usage of ADD COLUMN clause.

Furthermore, in MySQL we can use the UPDATE statement to update our data. We can also use the Form Editor provided in Workbench as we can see in Figure 5.4.

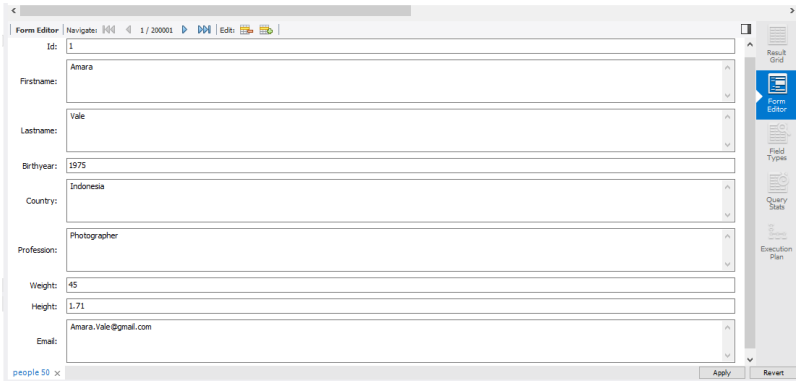


Figure 5.4: The *Form Editor* UPDATE provided by Workbench.

In contrast with other NoSQL databases, MySQL requires us to define our tables and columns

before we can store anything, and every row in a table must have the same columns.

Traditionally, relational databases such as MySQL have usually required a schema to be defined before documents can be stored.[54] With some configurations we can use MySQL as a schema-less and flexible document store. These configurations require server features such as X Plugin (a prerequisite for using MySQL as a document store) and X Protocol (which allows pipelining of commands and extends the message layer).

5.1.2 Neo4j

For our tests we are using the Neo4j Enterprise 4.4.0 Desktop Edition.

Database Visualization

Neo4j Browser is an interactive cypher command shell that allows us to interact with our graph and visualize the information in it. Neo4j Browser is bundled with Neo4j and is available in all editions and versions of Neo4j. Furthermore, Neo4j uses the Cypher query language which differs from other conventional languages but delivers a great variety of capabilities. Data can be visualized in graph, table and text forms. In Figure 5.5 we can see how data is being visualized in a graph form with a 25 node limit, while in Figure 5.6 we used a 500 node limit.

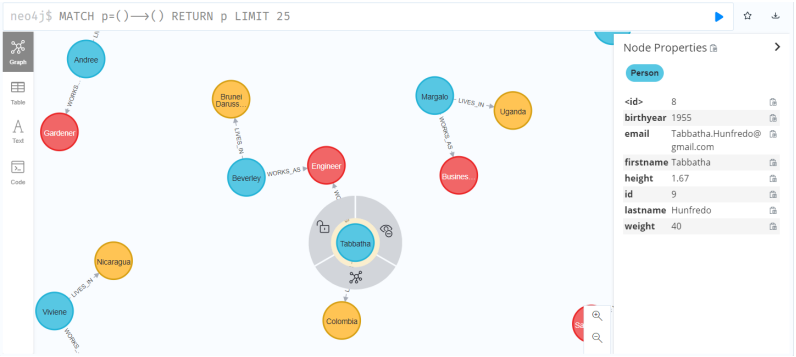


Figure 5.5: Data visualization in a graph form with a 25 node limit.

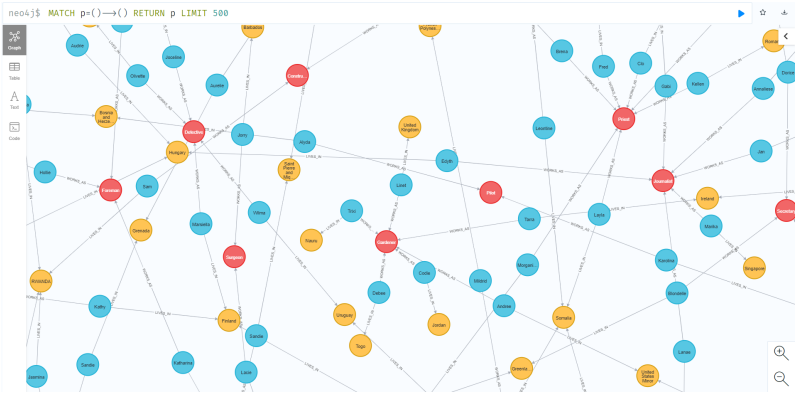


Figure 5.6: Data visualization in a graph form with a 500 node limit.

Figure 5.7: Data visualization in a table form.

In Figure 5.7 we can see how data is being visualized in a table form, while in Figure 5.8 we can see how data is being visualized in a text form.

Figure 5.8: Data visualization in a text form.

In Figure 5.9 we can see how this query example transfers to the screen:
MATCH (o:Person)-[]-(e:Country)
MATCH (o:Person)-[]-(h:Profession)
WHERE (h.profession=”Doctor” or h.profession=”Pilot”) and o.birthyear>1990 and o.weight<68 and o.weight>50 and o.height<1.70
RETURN o,e,h LIMIT 50000;

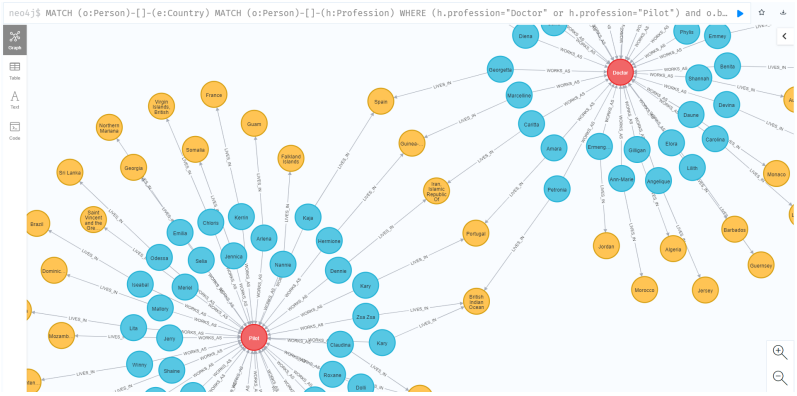


Figure 5.9: A query example.

Performance on Read/Write Speed

Neo4j as a graph database, supports the relational data form, so both ”relational data model” and ”flat data type” will be tested on this system.

- Flat file importation and querring:
  - The flat file importation took 3190ms to complete (which is the average import time of 10 tests for this file as we discussed in Subsection 3.2.2). The datasheet is provided in Subsection 3.2.2. Below we can see one of our shell notifications:
    - Added 200000 labels, created 200000 nodes, set 1800000 properties, completed after 3203 ms.
  - For the read speed we used 6 different queries and we counted the average query execution time. These are the Cypher queries we have used (query execution time in these tests are indications of the time the tests were performed):
    - MATCH (n:personprofession: ’Doctor’,
      birthyear:”1990”)
      RETURN n LIMIT 10000
      Shell notification: Started streaming 71 records after 1 ms and completed after 228 ms.
    - MATCH (n)
      WHERE n.birthyear>1995 and n.weight<68
      RETURN n LIMIT 50000
      Shell notification: Started streaming 6801 records after 2 ms and completed after 29 ms.
    - MATCH (n)
      WHERE n.birthyear>1995 and n.weight<68 and
      n.weight>50 RETURN n LIMIT 50000
      Shell notification: Started streaming 4182 records in less than 1 ms and completed

after 12 ms.

* MATCH (n)
  WHERE n.country=”France” and (n.profession=”Doctor” or n.profession=”Pilot”)
  RETURN n LIMIT 50000
  Shell notification: Started streaming 26 records after 1 ms and completed after 227 ms.

* MATCH (n)
  RETURN avg(n.weight),avg(n.height)
  Shell notification: Started streaming 1 records in less than 1 ms and completed after 221 ms.

* MATCH (n)
  WHERE n.profession=”Engineer” and n.weight>=100
  RETURN count(n) LIMIT 50000
  Shell notification: Started streaming 1 records after 1 ms and completed after 213 ms.

The average query time of all 6 queries is 155ms (we took an average query time of 10 tests for each of our 6 queries as we discussed in Subsection 3.2.2).

- Relational data files importation and querring:

  – For this part we imported 5 files (which is the average import time of 10 tests for each file as we discussed in Subsection 3.2.2). The datasheet is provided in Subsection 3.2.2. Below we can see some example shell notifications:

    * Added 200000 labels, created 200000 nodes, set 1400000 properties, completed after 2140 ms.
    * Added 243 labels, created 243 nodes, set 243 properties, completed after 16 ms.
    * Added 55 labels, created 55 nodes, set 110 properties, completed after 19 ms.
    * Created 200000 relationships, completed after 3771 ms.
    * Created 200000 relationships, completed after 3601 ms.

  The average relational data files importation time took 9346ms to complete.

  – For the read speed we used 5 different queries and we counted the average query execution time. It is important to point out that we used indexes in these queries. A database index is a redundant copy of some of the data in the database for the purpose of making searches of related data more efficient. These are the 5 Cypher queries we have used (query execution time in these tests are indications of the time the tests were performed):

    * MATCH (o:Person)-[]-(e:Country)
      MATCH (o:Person)-[]-(h:Profession)
      WHERE (h.profession=”Doctor” or h.profession=”Pilot”) and o.birthyear>1990 and o.weight<68 and o.weight>50

RETURN avg(o.weight),avg(o.height) LIMIT 50000;
Shell notification: Started streaming 1 records after 16 ms and completed after 30 ms.

* MATCH (o:Person)-[]-(e:Country)
MATCH (o:Person)-[]-(h:Profession)
WHERE (h.profession="Doctor" or h.profession="Pilot") and o.birthyear>1990
RETURN o,e,h
ORDER BY o.weight LIMIT 1;
Shell notification: Started streaming 1 records in less than 1 ms and completed after 24 ms.

* MATCH (o:Person)-[]-(e:Country)
MATCH (o:Person)-[]-(h:Profession)
WHERE h.profession="Doctor" and o.birthyear=1990
RETURN o,e,h LIMIT 50000;
Shell notification: Started streaming 62 records in less than 1 ms and completed after 14 ms.

* MATCH (o:Person)-[]-(e:Country)
MATCH (o:Person)-[]-(h:Profession)
WHERE h.profession="Doctor" and o.birthyear>1995 and o.weight<68 and o.weight>50
RETURN o,e,h LIMIT 50000;
Shell notification: Started streaming 82 records after 1 ms and completed after 15 ms.

* MATCH (o:Person)-[]-(e:Country)
MATCH (o:Person)-[]-(h:Profession)
WHERE (h.profession="Doctor" or h.profession="Pilot") and o.birthyear>1990 and o.weight<68 and o.weight>50
RETURN o,e,h LIMIT 50000;
Shell notification: Started streaming 260 records after 22 ms and completed after 63 ms.

The average query time of all 5 queries is 29,2ms(we took an average query time of 10 tests for each of our 5 queries as we discussed in Subsection 3.2.2).

Other Features

Neo4j is often described as schema optional, meaning that it is not necessary to create indexes and constraints.[58] This means that we can create nodes, relationships and properties without defining a schema up front.

In Neo4j the SET clause is used to update labels on nodes and properties on nodes and relationships. We can also add new properties to a node with the SET clause. That show us how flexible the schema of this database system is.

With Neo4j we can also use unique property constraints to ensure that property values are

unique for all nodes with a specific label. Unique constraints do not mean that all nodes have to have a unique value for the properties-nodes without the property are not subject to this rule.

5.1.3 Cassandra

For our tests we are using the Apache Cassandra 3.11.

Database Visualization

Cassandra does not use a desktop aplication or a browser shell to run. Users can access Cassandra through its nodes using Cassandra Query Language (CQL) from the command line. CQL treats the database (Keyspace) as a container of tables. Programmers use cqlsh: a prompt to work with CQL or separate application language drivers. Clients approach any of the nodes for their read-write operations a great variety of capabilities. Data can be visualized in table form as we can see in Figure 5.10.

```
cqlsh:imdb> SELECT * FROM people;

 id     | birthyear | country                                | email                                | firstname | height | lastname    | profession                | weight
--------+-----------+----------------------------------------+--------------------------------------+-----------+--------+-------------+---------------------------+--------
   4317 |      1996 |                              Sri Lanka |          Devina.Rheingold@gmail.com |    Devina |   1.74 |  Rheingold |                 Scientist |     54
  62693 |      1978 |                             Mozambique |             Tarra.Verger@gmail.com |     Tarra |   2.07 |      Verger |               Electrician |     81
 121478 |      1979 |                              Greenland |            Vinita.Brunell@gmail.com |    Vinita |   1.53 |     Brunell |                   Postman |     95
 113295 |      1973 |                             Mozambique |        Michaelina.Rudolph@gmail.com | Michaelina |   1.83 |     Rudolph |                    Geisha |    144
 176996 |      1962 |                          Liechtenstein |              Elie.Fosque@gmail.com |       Elie |   2.26 |      Fosque |                     Pilot |     80
  51078 |      1974 |                                  Sudan |           Lisette.Gower@gmail.com |   Lisette |   2.35 |       Gower |                 Carpenter |    131
 143843 |      1959 |                               Honduras |             Viki.Orlene@gmail.com |      Viki |   1.69 |      Orlene |                 Housewife |     87
  77328 |      1994 |                              Hong Kong |          Averyl.Durante@gmail.com |    Averyl |   2.39 |     Durante |                Journalist |     48
  84757 |      1948 | Palestinian Territory, Occupied        |             Tera.Khorma@gmail.com |      Tera |   2.38 |      Khorma |                  Mechanic |     82
 122937 |      1963 |                               Bermuda |          Gwyneth.Ioab@gmail.com |   Gwyneth |   2.05 |        Ioab |                  Reporter |    121
  35262 |      1968 |                                  Cuba |            Jany.Brodench@gmail.com |      Jany |   1.64 |    Brodench |           Police officer |    130
  54191 |      1974 |                            Costa Rica |              Jackie.Suk@gmail.com |    Jackie |   2.19 |         Suk |                    Farmer |    111
 155320 |      1952 |                               Somalia |        Michaelina.Agle@gmail.com | Michaelina |    2.3 |        Agle |              Catholic nun |    103
 117179 |      2000 |                                 Samoa |           Cecile.Reinke@gmail.com |     Cecile |   2.04 |      Reinke |                  Reporter |    135
 143615 |      1981 |                                Bhutan |         Annabela.Travax@gmail.com |   Annabela |   1.47 |      Travax |           Forest ranger |    148
  25269 |      1961 |                                 Italy |          Teriann.Eckblad@gmail.com |    Teriann |   2.24 |     Eckblad |                Cameraman |    136
 199781 |      1953 |                                 Benin |          Evaleen.Velick@gmail.com |    Evaleen |   1.68 |      Velick |                 Secretary |     52
 130533 |      1973 |                               Pitcairn |          Angelique.Fink@gmail.com |   Angelique |   2.16 |        Fink | Lunchroom supervisor |     88
 123410 |      1994 |                Bosnia and Herzegovina |            Vevay.Afton@gmail.com |     Vevay |   1.82 |       Afton |                Cameraman |     78
  39433 |      1949 | Congo, The Democratic Republic of the  |            Lanae.Eachern@gmail.com |     Lanae |   1.66 |     Eachern |                   Singer |    100
   3372 |      1998 |                              Senegal   |          Cissiee.Hazlett@gmail.com |   Cissiee |   2.23 |     Hazlett |                   Postman |     92
  88056 |      1986 |                               Zambia   |       Gabriellia.Llovera@gmail.com | Gabriellia |   2.14 |     Llovera |                  Engineer |     68
  96748 |      1961 |                         Sierra Leone   |       Harmonia.Garrison@gmail.com |   Harmonia |   1.53 |    Garrison |              Catholic nun |    101
  91274 |      2003 |                              Armenia   |          Angelique.Juliet@gmail.com |   Angelique |   1.94 |      Juliet |                 Carpenter |    121
 180284 |      1961 |          Tanzania, United Republic of  |            Alexine.Peti@gmail.com |    Alexine |   1.49 |        Peti |           Forest ranger |    111
  84991 |      1945 |                              Guyana    |              Bee.Gaynor@gmail.com |       Bee |   2.18 |      Gaynor |                 Secretary |     40
  62602 |      1974 |                                 Cuba   |             Roz.Estella@gmail.com |       Roz |   2.26 |     Estella |                Footballer |     79
```

Figure 5.10: Data visualization in a table form.

In Figure 5.11 we can see how this query example transfers to the screen:
SELECT * FROM people WHERE country=’Greece’ AND profession=’Doctor’ ALLOW FILTERING;

```
 id     | birthyear | country | email                          | firstname | height | lastname      | profession | weight
--------+-----------+---------+--------------------------------+-----------+--------+---------------+------------+--------
 133353 |      1971 |  Greece |       Tabbatha.Gemini@gmail.com |   Tabbatha |   1.66 |        Gemini |     Doctor |     77
 164298 |      1982 |  Greece |            Dacia.Erb@gmail.com |     Dacia |   1.92 |           Erb |     Doctor |    150
  81114 |      1999 |  Greece |         Nikki.Nicoline@gmail.com |     Nikki |   1.78 |      Nicoline |     Doctor |     68
  32820 |      1955 |  Greece |      Carmencita.Vivle@gmail.com | Carmencita |   2.11 |         Vivle |     Doctor |     71
 132024 |      2003 |  Greece |   Antonietta.Kravits@gmail.com | Antonietta |   1.78 |       Kravits |     Doctor |     97
 112119 |      1969 |  Greece |       Raquela.Katrine@gmail.com |    Raquela |      2 |       Katrine |     Doctor |     94
 130437 |      1983 |  Greece |          Danika.Bari@gmail.com |    Danika |   1.52 |          Bari |     Doctor |    105
  51262 |      2000 |  Greece |        Xylina.Kristi@gmail.com |    Xylina |   2.25 |         Kristi |     Doctor |     63
 160074 |      2001 |  Greece |     Dianemarie.Anyah@gmail.com | Dianemarie |   2.27 |         Anyah |     Doctor |     90
  28076 |      1980 |  Greece |       Demetris.Lewes@gmail.com |   Demetris |   1.87 |         Lewes |     Doctor |    120
 113474 |      1999 |  Greece |          Laurene.Nea@gmail.com |    Laurene |   1.65 |           Nea |     Doctor |     56
   9113 |      1953 |  Greece |       Corry.Campball@gmail.com |     Corry |   1.54 |      Campball |     Doctor |    105
  76805 |      1955 |  Greece |       Leeanne.Camden@gmail.com |    Leeanne |   2.32 |        Camden |     Doctor |     99
    487 |      2001 |  Greece | Susette.Hollingsworth@gmail.com |    Susette |   2.18 | Hollingsworth |     Doctor |     57
 193447 |      1989 |  Greece |       Fanny.Tremayne@gmail.com |     Fanny |   2.34 |      Tremayne |     Doctor |     40
 190701 |      1988 |  Greece |       Jennica.Bandeen@gmail.com |    Jennica |   1.84 |       Bandeen |     Doctor |     94
  69731 |      1977 |  Greece |     Maryellen.Lymann@gmail.com |  Maryellen |   1.47 |        Lymann |     Doctor |    136
  61987 |      1954 |  Greece |         Amelia.Hoban@gmail.com |    Amelia |   1.92 |         Hoban |     Doctor |     74
```

Figure 5.11: A query example.

Performance on Read/Write Speed

In Cassandra the entities and their relationships are considered during table design. Queries are best designed to access a single table, so all entities involved in a relationship that a query encompasses must be in the table. Cassandra does not support joins. In relational database design,

we are often taught the importance of normalization. This is not an advantage when working with Cassandra because it performs best when the data model is denormalized as we read at Cassandra's documentation. So we are going to test only the flat data type on Cassandra.

- Flat file importation and querring:
  - The flat file importation took 10,653secs to complete (which is the average import time of 10 tests for this file as we discussed in Subsection 3.2.2). The datasheet is provided in Subsection 3.2.2. Below we can see one of our shell notifications:
    * Processed: 200000 rows; Rate: 18772 rows/s; Avg. rate: 18774 rows/s
      200000 rows imported from 1 files in 10.187 seconds (0 skipped).
  - For the read speed we used 6 different queries and we counted the average query execution time. There is not a shell notification when we run queries. So we have to use tracing in cqlsh. It provides time-stamps for each stage of the query, with millisecond precision. These are the 6 CQL queries we have used (query execution time in these tests are indications of the time the tests were performed):
    * SELECT COUNT(id) FROM people WHERE profession='Engineer' AND weight>=100 ALLOW FILTERING;
      timestamp: completed after 509ms.

    * SELECT * FROM people WHERE country='France' AND profession IN('Doctor', 'Pilot') ALLOW FILTERING;
      timestamp: completed after 450ms.

    * SELECT * FROM people WHERE birthyear=1990 AND profession='Doctor' ALLOW FILTERING;
      timestamp: completed after 643ms.

    * SELECT * FROM people WHERE birthyear>1990 AND weight<68 ALLOW FILTERING;
      timestamp: completed after 794ms.

    * SELECT * FROM people WHERE birthyear>1990 AND weight<68 AND weight>50 ALLOW FILTERING;
      timestamp: completed after 835ms.

    * SELECT AVG(weight), AVG(height) FROM people;
      timestamp: completed after 1231ms.

    The average query time of all 6 queries is 750ms (we took an average query time of 10 tests for each of our 6 queries as we discussed in Subsection 3.2.2).

Other Features

In Cassandra every row in a CQL table will have the predefined columns defined at table creation. Columns can be added later using an alter statement. Furthermore we cannot change the

primary key of a table once it is been created. The partition key and primary key determine how data is stored and how it is distributed (partitioned) across the nodes in the cluster, which means that it cannot be changed. If we need to change the primary key, it means that we have a completely new table so we will have to create a brand new one.

With the UPDATE command we can write one or more column values to a row in a Cassandra table. Like INSERT, UPDATE is an upsert operation, if the specified row does not exist, the command creates it. So it makes the schema flexible somehow.

5.1.4 MongoDB

For our tests we are using the MongoDB Compass.

Database Visualization

Compass is the graphical user interface of MongoDB. It is an interactive tool for querying, optimizing, and analyzing our MongoDB data. It also provides everything from schema analysis to index optimization to aggregation pipelines in a single, centralized interface. Data can be visualized in table and document forms. In Figure 5.12 we can see how data is being visualized in a table form, while in Figure 5.13 in a document form.

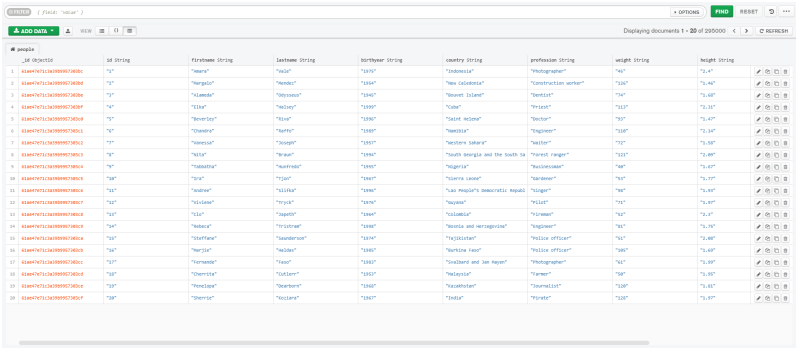


Figure 5.12: Data visualization in a table form.

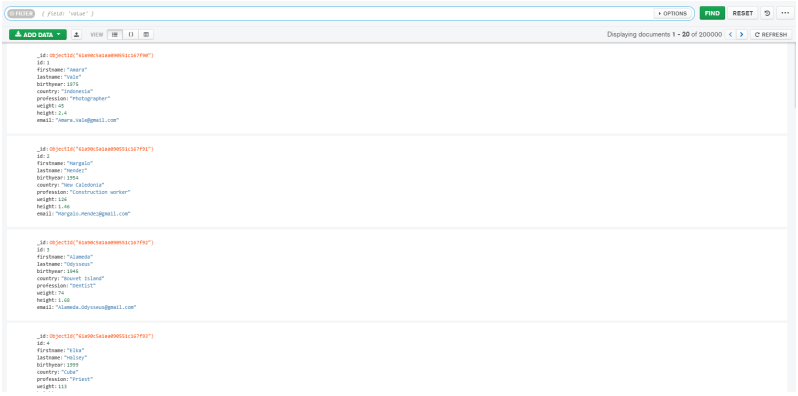


Figure 5.13: Data visualization in a document form.

In Figure 5.14 we can see how this query example transfers to the screen: (profession: 'Engineer', weight: ($gt:100))

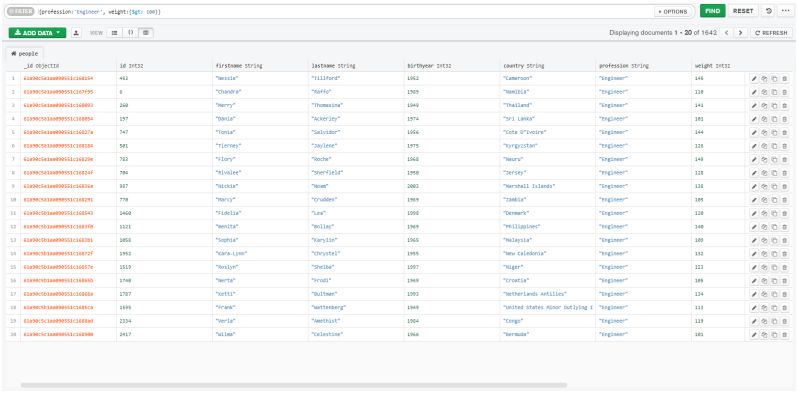


Figure 5.14: A query example.

Performance on Read/Write Speed

In MongoDB, data is stored as documents. These documents are stored in MongoDB in JSON (JavaScript Object Notation) format. JSON documents support embedded fields, so related data and lists of data can be stored with the document instead of an external table. Using MongoDB removes the complex object-relational mapping (ORM) layer that translates objects in code to relational tables. Instead of tables, a MongoDB database stores its data in collections. Each document has one or more fields. No complex joins are needed in MongoDB. So we are going to test only the flat data type on MongoDB since our relational tables format is not supported.

- Flat file importation and querring:
  - The flat file importation took 109secs to complete (which is the average import time of 10 tests for this file as we discussed in Subsection 3.2.2). The datasheet is provided in Subsection 3.2.2. Below we can see one of our shell notifications:
    - Write time : 1m:32s:09ms
  - For the read speed we used 6 different queries and we counted the average query execution time. These are the 6 MongoDB queries we have used (query execution time in these tests are indications of the time the tests were performed):
    - (country: "France", profession: ($in:("Doctor","Pilot")))
      shell notification: Actual Query Execution Time (ms):130
    - (birthyear: ($gt: 1990), weight: ($lt: 68))
      shell notification: Actual Query Execution Time (ms):169
    - (birthyear:1990, profession:"Doctor")
      shell notification: Actual Query Execution Time (ms):167

* (birthyear: ($gt: 1990), weight: ($lt: 68), weight: ($gt: 50))
  shell notification: Actual Query Execution Time (ms):172

* avgweight:($avg:'weight'), avgheight:($avg:'height')
  shell notification: Actual Query Execution Time (ms):137

* (profession:'Engineer', weight:($gt: 100))
  shell notification: Actual Query Execution Time (ms):100

The average query time of all 6 queries is 142ms (we took an average query time of 10 tests for each of our 6 queries as we discussed in Subsection 3.2.2).

Other Features

Data in MongoDB has a flexible schema. Collections do not enforce document structure by default. That means that we do not need to design the schema of the database when we are using MongoDB.

We also may require to modify a document once it is inserted into a collection. The update operation is used for making changes in the existing database. MongoDB database provides two methods for update operations, updateOne() and updateMany(). The updateOne() updates a single document, while updateMany() updates multiple documents in the collection.

There are a few disadvantages of the MongoDB database as well. MongoDB uses high memory for data storage. There is a 16MB limit for document size. To store documents larger than the maximum size, MongoDB provides the GridFS API which stores them in multiple chunks. GridFS is a specification for storing and retrieving files that exceed the BSON-document (a binary-encoded serialization of JSON documents) size limit of 16 MB.

5.1.5 Redis

For our tests we are using the Open-Source Redis version 6.2.6.

Database Visualization

We connect to the Redis server through the Ubunty 20.04.3 LTS terminal and access the client through the same Terminal. We also use a program called Redis Inside, which helps the admin of the database to read, write, modify and delete keys and their values from the database. Data can be visualised in a key-value form as we can see in Figure 5.15.

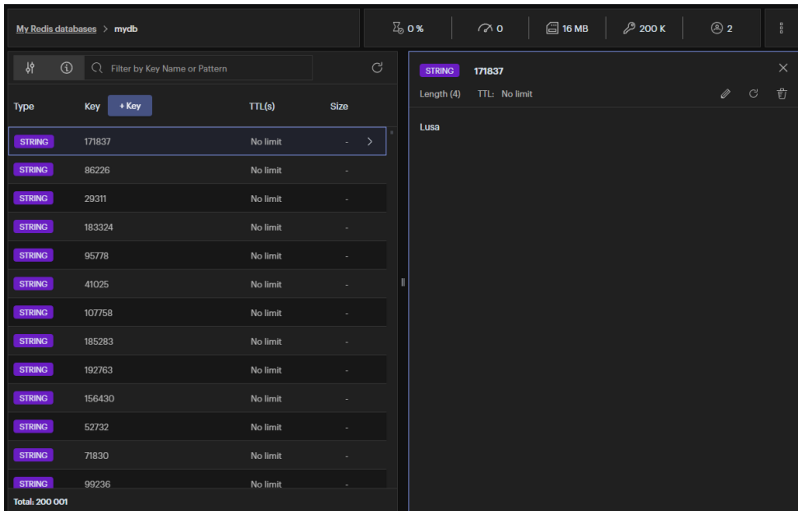


Figure 5.15: Data visualization in a key-value form.

In figure 5.16 we can see how a simple key search transfers to screen.

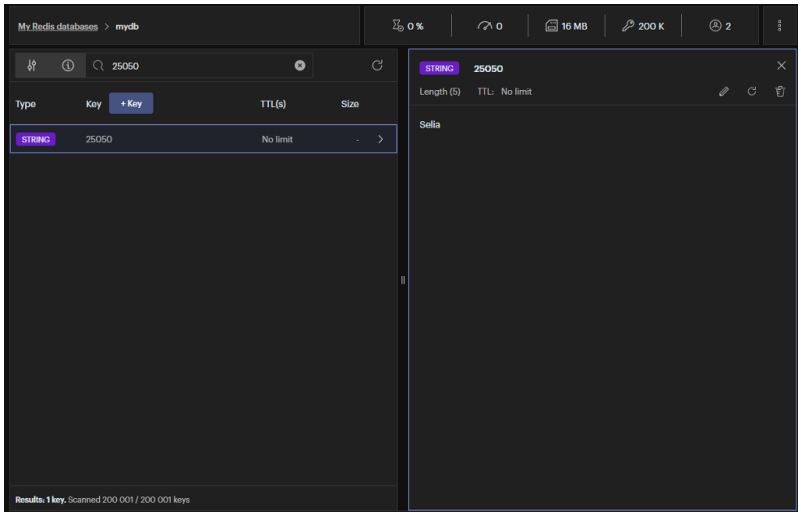


Figure 5.16: Key search example.

Performance on Read/Write Speed

Redis as data structure server does not use a structured query language and there is no support for relational algebra. We cannot submit ad-hoc queries. So, we can not execute the same queries as we did with the other database systems we tested. All data accesses should be anticipated by the developer and proper data access paths must be designed.

Although Redis developers provide great document tutorials and YouTube videos ([74][75]) on how to import large CSV files into the Redis database system, we could not manage to achieve that. Importing such files on Redis requires advanced knowledge on scripting languages like JavaScript and backend frameworks like Node.js. This also requires good skills on using database migration

tools like RIOT (RIOT Redis is a migration tool that allows for seamless live replication between two Redis databases). We also tried to use a utility called *Bulk Loader*, which is a command-line application tool that loads content and metadata (like CSV files) into a virtual content repository, but without sucess. Unfortunately, we do not have such technical skills.

So, we conclude that we cannot include Redis in our read/write performance tests. In addition, we can say that we performed operations to find records on Redis (Figure 5.16) and we had quick respond times. Redis is a great choice if we have clearly-defined data design and need a really fast access. For example if we want to use a database management system for a product business which stores many products with an id and a name which works as key(id)-value(name), Redis is a great choice.

Other Features

Redis provides the ability to update its data across its attributes with ease. Since it is a schema-less database, we can add new attributes inside Redis's keyspace.

There are a number of Redis commands that are useful for managing keys regardless of what type of data they hold. Such commands are *rename*, *randomkey*, *del*, etc.

5.2 System Comparison based on their Characteristics

In this part we will summarise and compare the characteristics of the systems presented earlier in Chapter 4 so we can have a clear look.

As we have seen, MySQL database management system follows a *Client-Server Architecture* (which consists of Client, Server and Storage layers). This type of architecture has also been adopted by the rest of the NoSQL database management systems we evaluated. This conclusion, however, is a superficial study of the architecture of each system. As we have seen, most of the systems we have studied relate their architecture to their ability of replication, scalability, data modeling and ACID compliance.

In terms of scalability, some of our database management systems share identical methods. For example, we were used to the fact that MySQL as a relational database system scales vertically, but we have recently seen that it provides horizontal scalability, making it a better, less expensive resource management system. MySQL uses the *sharding* technique for distributing datasets across multiple databases (MySQL launched sharding approach in early 2012 [54]). The same scaling technique is also used by Neo4j, Cassandra, MongoDB and Redis. This means that MySQL adapts to today's high data demands and competes with NoSQL databases in terms of data handling and system resource expenses. In Figure 5.17 we can see how a horizontally partioned (*sharded*) database system distributes its data.

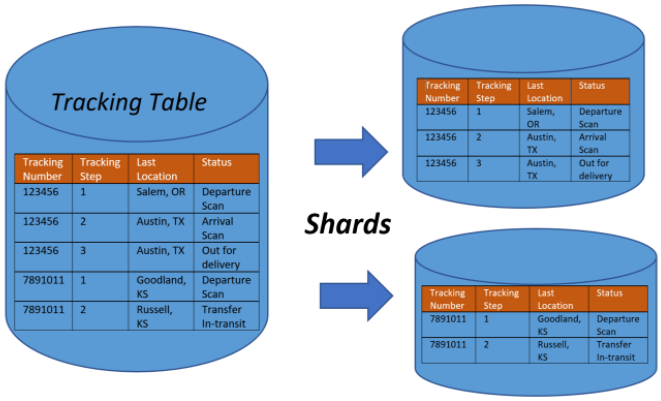


Figure 5.17: Data sharding example.

Every database system wants to achieve redudancy and high availability. All database management systems we studied have their own replication methods and share a similar one. The common master-slave replication model (which enables data from one database server to be replicated to one or more other database servers) is supported by MySQL, Neo4j, MongoDB and Redis. This model is presented in Figure 5.18. In contrast, Cassandra database system does not support that model. Cassandra as we have seen, since it is designed as a ring-type node management system, replicates and distributes its data across its nodes according to a *replication factor* we define.

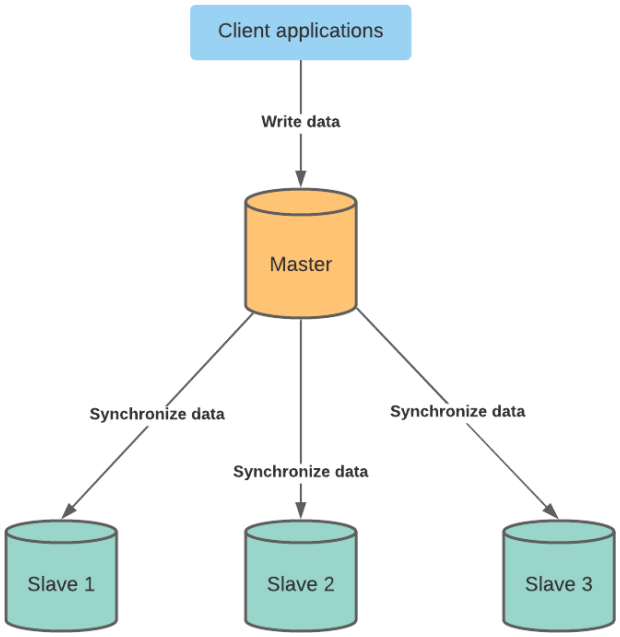


Figure 5.18: Master-slave replication example.

It goes without saying that each database has its own purpose and is used for specific types of data. The main difference between MySQL and the other systems we studied stands in their ability to handle the *relational data* model. We can stick with MySQL if we want our information

to remain in the structure we originally created and do not want to deal with massive amounts of data and numerous data types. MySQL as a SQL database system is also famous for its ability to perform JOIN operations (operations performed to enstablish connections between two or more database tables). This is the basic philosophy behind relational database systems. As we have seen in Subchapter 5.1.2, Neo4j supports the relational data model and extends its flexibility in terms of schema manipulation (schema altering). The same thing happens with MongoDB through advanced relational mapping. In contrast, Cassandra and Redis database management systems are designed for handling massive non-relational *flat* data. So we conclude that we can achieve the support of the relational data model in NoSQL systems through Neo4j and MongoDB.

In terms of security and support, each DBMS offers safe data transfer protocols and security measures. These systems share the basic TLS/SSL (Transport Layer Security/Secure Sockets Layer) data tranfer security protocols and other authenication and encryption methods, while they offer a variety of backup methods as we have seen in Chapter 4.

In our opinion, the most important factor in terms of database ability is ACID transactions. ACID compliance was one of main reasons SQL systems contrast with NoSQL ones. As we have seen in this thesis, ACID properties of transactions are supported from the most NoSQL databases we have examined. A special example is that MongoDB became ACID compliant in version 4.0 in 2018 (4.4). That means that there is still time for other non ACID compliant NoSQL databases to extend their abilities. Furthermore, Neo4j and Redis support the ACID transactions (Redis through AOF configuration). In contrast, Cassandra does not offer consistency in terms of transactions, so the ACID model is not supported.

The following Table 5.1 describes the main characteristics between MySQL, Neo4j, Cassandra, MongoDB and Redis.

Table 5.1: Summary of databases characteristics.

	MySQL	Neo4j	Cassandra	MongoDB	Redis
Database Type	Relational	Graph	Column-oriented	Document-based	Key-value store
Release Year	1995	2010	2008	2009	2009
Architecture	Client-Server Architecture which contains the following major layers: Client, Server and Storage	Neo4j offers High Availability Architecture (HA).	Cassandra is designed to handle big data workloads across multiple nodes with no single point of failure.	MongoDB is a cloud database solution for contemporary applications that is available globally.	Redis employs a primary-replica architecture and supports asynchronous replication.
Scalability	Vertical, Horizontal	Horizontal	Horizontal, Linear	Horizontal	Verical, Horizontal

Replication (Nodes, Internode Communication)	Synchronous replication which is a characteristic of NDB Cluster.	Neo4j's Causal Clustering provides three main features: Safety, Scale and Causal consistency.	Cassandra stores replicas on multiple nodes to ensure reliability and fault tolerance. Cassandra uses Gossip is a peer-to-peer communication protocol.	MongoDB achieves replication by the use of replica set. Replica sets provide redundancy and high availability.	Leader follower (master-replica) replication which allows replica Redis instances to be exact copies of master instances.
Query Language	MySQL uses the SQL language to query the database.	Cypher is Neo4j's graph query language that allows users to store and retrieve data from the graph database.	The Cassandra Query Language is the primary language for communicating with the Apache Cassandra™ database.	MongoDB uses the MongoDB Query Language, designed for easy use by developers.	Redis instead comes with its own set of commands for managing and accessing data.
Support	Physical and Logical backups.	Neo4j supports backing up and restoring both online and offline databases.	Cassandra's support for replicating across multiple datacenters provides low latency for users.	Cloud Backups, Legacy Backups with the support of backing up and restoring MongoDB replica sets and sharded clusters.	Redis provides a different range of persistence options: RDB, AOF, No persistence, RDB + AOF.

Security	MySQL supports TLS/SSL and X.509. Furthermore, MySQL Enterprise Edition supports an authentication method that enables MySQL Server to use PAM to authenticate MySQL users.	Neo4j supports TLS/SSL. Neo4j Browser has two mechanisms for avoiding users having to repeatedly enter their Neo4j credentials.	Cassandra supports TLS/SSL, Client authentication and authorization.	MongoDB supports authentication, network isolation, authorization, auditing and encryption.	Redis supports TLS/SSL, authebtication and protected mode.Redis is designed to be accessed by trusted clients inside trusted environments.
Transactions	ACID compliant	ACID compliant	Not ACID compliant.	ACID compliant in version 4.0.	ACID compliant with a specific configuration.

5.3 System Comparison based on the Evaluation Criteria

In Section 5.1 we saw that every DBMS has its unique graphical user interface. Some of these database systems offer a greater overall user interaction experience with data than the others. Of all these systems, we singled out Neo4j as it is a pioneering step in data management and usability. For us the visualization of every single record and data relation is very important. Neo4j prevailed over the other DBMS in visualization ability. In addition, DBMS like MySQL and MongoDB offered us a decent user experience which covered all of our needs. On the other hand, Cassandra and Redis did not meet our expectations in the visualization part.

On the performance part we have to admit that certain database systems impressed us. Since the workload factor does not exist in our tests, it is very important to point out that the performance tests we accomplished do not meet the actual capabilities of these systems, as we discussed in Subsection 3.2.2. So the read/write performance tests results correspond to limited estimates on overal database performance, but accurate results on personal-use, single-connection database performance. Figures 5.19 and 5.21 present the read/write performance results in chart form. The relational data files have been tested on MySQL and Neo4 database systems, while we don not have performance results on Redis as we discussed in Section 5.1.

For the write perfromance we have collectively the following results:

- Flat file
  - MySQL: 2973,579S
  - Neo4j: 3,19S

  - Cassandra: 10,653S
  - MongoDB: 109S

- Relational data files
  - MySQL: 8958,352S
  - Neo4j: 9,346S

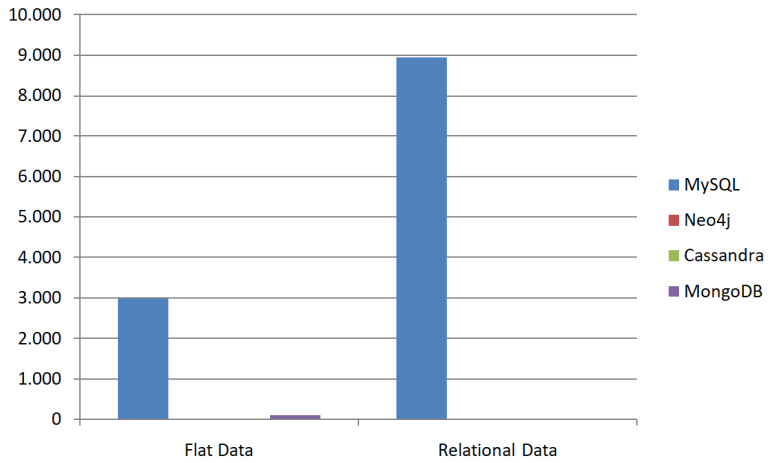


Figure 5.19: Write performance (in seconds).

Figure 5.19 shows the write performance (in seconds) for the different database systems tested. Less values indicate faster write transactions and thus, better performance.

Although we have used the same indexes on relational data file attributes and we have used the default configurations on each system, we can note a significantly large import time in MySQL. Since there is a large import time deviation between MySQL and the other systems, we will also provide a second chart (5.20) in which MySQL results are omitted, so that we can focus on the import performance of the rest systems.

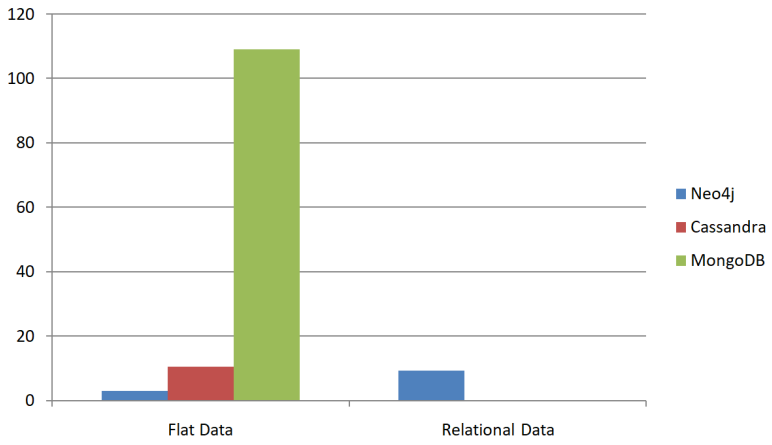


Figure 5.20: Write performance (in seconds) of the NoSQL systems.

Concering the read perfromance we have collectivelly the following results:

- Flat file
  - MySQL: 205ms
  - Neo4j: 155ms
  - Cassandra: 750ms
  - MongoDB: 142ms
- Relational data files
  - MySQL: 168,6ms
  - Neo4j: 29,2ms

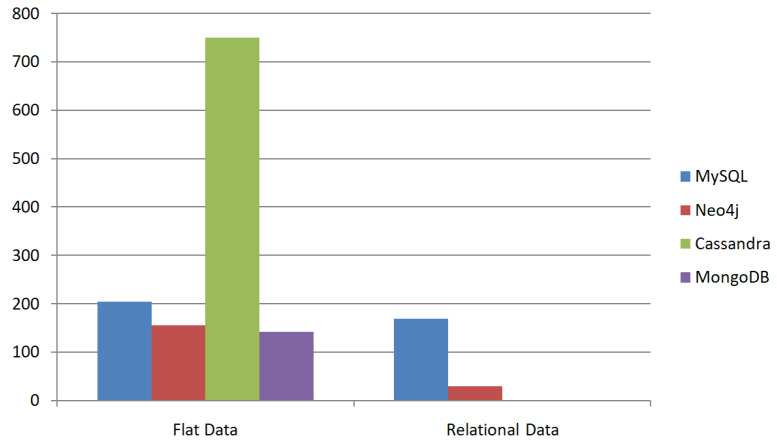


Figure 5.21: Read performance (in milliseconds).

In the results presented in Figure 5.21 we can see that some database systems offer greater performance on read and write speeds over other systems. Notice that less values indicate faster read transactions and thus, better performance.

Read time was high with MySQL, while Neo4j offered the best solution on this part. For the relational data part we can see that Neo4j can offer greater potential in terms of read/write performance than MySQL. In terms of read speed on flat files, MongoDB was the best solution, giving us remarkable results, while Cassandra was the worst with a relatively large deviation in speed time from the other systems.

Chapter 6

Conclusion and Future Directions

In this dissertation we have made an introduction to the operation and philosophy of databases. Our goal was to examine whether the transition from traditional database systems to new NoSQL systems is important and worth trying, by examining the main differences between these systems. We have seen that NoSQL databases are designed to support seamless, online horizontal scalability without significant single points of failure, while their data models are dynamic. Over time, they have become ACID compliant, countering the traditional relational database systems. We have also seen that SQL database systems like MySQL were modernized by extending their scaling capabilities. When we consider either database, it is also important to consider critical data needs. In other words, each database system is designed to manage specific data. NoSQL systems offer consistency and multiple choices in data needs.

Within this research, we would not like to hide our enthusiasm for a specific database system we have tried out. The system that impressed us the most in terms of usability is Neo4j. We came in contact with this system for the first time and we found it excellent in every way. Neo4j covered us in terms of visual experience, ease of use and capabilities over data.

Considering all the facts we examined, we can say with confidence that NoSQL systems are of the utmost importance and can offer as many capabilities or even more than the old relational SQL systems.

Bibliography

[1] Hector Garcia-Molina, Jeffrey D Ullman, and Jennifer Widom. *Database system implementation*. Vol. 672. Prentice Hall Upper Saddle River, 2000.

[2] *"Update – Definition of update by Merriam-Webster"*. URL: http://www.merriam-webster.com.

[3] *"Retrieval – Definition of retrieval by Merriam-Webster"*. URL: http://www.merriam-webster.com.

[4] *"Administration – Definition of administration by Merriam-Webster"*. URL: http://www.merriam-webster.com.

[5] *Data Models.* Prentice Hall, 1982.

[6] Paul Beynon-Davies. *Database systems*. Bloomsbury Publishing, 2017.

[7] William Harris Morehead Anne Fulche Nelson. *Building Electronic Commerce: With Web Database Constructions.* Prentice Hall, 2001.

[8] EF E F. Codd. *Derivability, redundancy and consistency of relations stored in large data banks*. Vol. 38. 1. ACM New York, NY, USA, 2009, pp. 17–36.

[9] Edgar F Codd. *A relational model of data for large shared data banks*. Springer, 2002, pp. 263–294.

[10] Edgar F Codd. *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[11] Christopher J. Date. *Date on Database: Writings 2000-2006*. Apress, 2007.

[12] Christopher J. Date. *An Introduction to Database Systems*. Addison Wesley, 2004.

[13] Theo Haerder and Andreas Reuter. *Principles of transaction-oriented database recovery*. Vol. 15. 4. ACM New York, NY, USA, 1983, pp. 287–317.

[14] URL: https://www.webopedia.com/definitions/atomic-operation/.

[15] URL: https://web.archive.org/web/20160303090517/http://archive.oreilly.com/pub/a/onjava/2001/11/07/atomic.html.

[16] Chris J Date. *SQL and relational theory: how to write accurate SQL code.*" O'Reilly Media, Inc.", 2011.

[17] URL: https://docs.microsoft.com/en-us/sql/connect/jdbc/understanding-isolation-levels?view=sql-server-ver15.

[18] *https://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-27.html*.

[19] URL: http://nosql-database.org/.

[20] Neal Leavitt. *Will NoSQL databases live up to their promise?* Vol. 43. 2. IEEE, 2010, pp. 12–14.

[21] C Mohan. *History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla*. 2013, pp. 11–16.

[22] URL: https://db-engines.com/en/blog_post/23.

[23] *https://martinfowler.com/bliki/NosqlDefinition.html*.

[24] URL: https://bigdata-ir.com/wp-content/uploads/2017/04/NoSQL-Distilled.pdf.

[25] Nishtha Jatana, Sahil Puri, Mehak Ahuja, Ishita Kathuria, and Dishant Gosain. *A survey and comparison of relational and non-relational database*. Vol. 1. 6. Citeseer, 2012, pp. 1–5.

[26] Eric Brewer. *CAP twelve years later: How the" rules" have changed*. Vol. 45. 2. IEEE, 2012, pp. 23–29.

[27] Eric A Brewer. *Towards robust distributed systems*. Vol. 7. 10.1145. 2000, pp. 343477–343502.

[28] URL: https://www.ue.katowice.pl/fileadmin/user_upload/wydawnictwo/SE_Artyku%C5%82y_231_250/SE_234/07.pdf.

[29] Douglas Kunda and Hazael Phiri. *A comparative study of nosql and relational database*. Vol. 1. 1. 2017, pp. 1–4.

[30] URL: https://www.mongodb.com/nosql-explained/nosql-vs-sql.

[31] URL: https://docs.microsoft.com/en-us/azure/architecture/guide/technology-choices/data-store-overview.

[32] URL: https://www.educative.io/blog/what-are-database-schemas-examples.

[33] URL: https://www.xplenty.com/blog/the-sql-vs-nosql-difference/#two.

[34] Sitalakshmi Venkatraman, Kiran Fahd, Samuel Kaspi, and Ramanathan Venkatraman. *SQL versus NoSQL movement with big data analytics*. Vol. 8. 12. 2016, pp. 59–66.

[35] URL: https://www.mongodb.com/databases/scaling.

[36] URL: https://www.rootstrap.com/blog/horizontal-vs-vertical-scaling/.

[37] URL: https://mariadb.com/resources/blog/acid-compliance-what-it-means-and-why-you-should-care/.

[38] URL: https://retool.com/blog/whats-an-acid-compliant-database/.

[39] URL: https://ravendb.net/articles/acid-transactions-in-nosql-ravendb-vs-mongodb.

[40] URL: https://cloudxlab.com/assessment/displayslide/345/nosql-cap-theorem.

[41] URL: https://www.ibm.com/cloud/learn/cap-theorem.

[42] URL: https://towardsdatascience.com/datastore-choices-sql-vs-nosql-database-ebec24d56106.

[43] URL: https://www.xplenty.com/blog/the-sql-vs-nosql-difference/#six.

[44] URL: https://www.mongodb.com/nosql-explained/best-nosql-database.

[45] URL: https://www.ibm.com/cloud/blog/sql-vs-nosql.

[46] URL: https://searchdatamanagement.techtarget.com.

[47] URL: https://gpdb.docs.pivotal.io.

[48] URL: https://devdotcode.com.

[49] URL: https://www.ibm.com.

[50] URL: https://www.ohloh.net/p/mysql/analyses/latest.

[51] URL: https://www.mysql.com/support/supportedplatforms/database.html.

[52] URL: https://www.mysql.com/downloads/.

[53] URL: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html.

[54] URL: https://dev.mysql.com/doc.

[55] URL: https://neo4j.com/blog/open-core-licensing-model-neo4j-enterprise-edition/.

[56] URL: https://web.archive.org/web/20110426031211/http://blogs.neotechnology.com/emil/2011/04/graph-databases-licensing-and-mysql.html.

[57] URL: https://neo4j.com.

[58] URL: https://neo4j.com/docs/.

[59] URL: https://github.com/apache/cassandra/releases/tag/cassandra-4.0.1..

[60] URL: https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/WIKIPEDI/W120911A.pdf.

[61] URL: http://cassandra.apache.org/doc.

[62] URL: https://www.mongodb.com/blog/post/state-of-mongodb-march-2010.

[63] URL: https://www.mongodb.com/docs/manual/release-notes/5.0/.

[64] URL: https://dzone.com/articles/why-mongodb-is-worth-choosing-find-reasons.

[65] URL: https://docs.mongodb.com.

[66] URL: https://redis.io/docs/about/.

[67] URL: https://redis.io/docs/getting-started/faq/.

[68] URL: https://gist.github.com/antirez/6ca04dd191bdb82aad9fb241013e88a8.

[69] URL: https://github.blog/2009-11-03-introducing-resque/.

[70] URL: https://instagram-engineering.com/storing-hundreds-of-millions-of-simple-key-value-pairs-in-redis-1091ae80f74c.

[71] URL: https://aws.amazon.com/redis/.

[72] URL: https://redis.io/documentation.

[73] URL: https://redis.com/blog/your-cloud-cant-do-that-0-5m-ops-acid-1msec-latency/.

[74] URL: https://www.youtube.com/watch?v=q51tPM4n3Tg&t=266s.

[75] URL: https://www.youtube.com/watch?v=q51tPM4n3Tg&t=266s.

[76] URL: https://www.wired.com/2012/01/amazon-dynamodb/.