Polytechnic School. University of Lleida

NoSQL databases

What, when and why?

Hanna Płomecka, Oskar Koła

2022-10-30

1 TABLE OF CONTENTS

2 What are NoSQL databases

2.1 The origins of NoSQL

In the mid-1990s, when the Internet gained more popularity, non-relational databases began to develop. They were a response to the need for fast data processing and diversity.

The acronym NoSQL was first used in 1998 by Carlo Strozzi, calling its open-source lightweight "relational" database that does not use SQL. The name reappeared in 2009 when Eric Evans and Johan Oskarsson when organized an event to discuss "open-source distributed, non-relational databases". The name attempted to label the emergence of an increasing number of non-relational, distributed data stores, including open-source clones of Google's Bigtable/MapReduce and Amazon's DynamoDB.

The name NoSQL can be understood as not SQL or not only SQL, but the term does not have a clear definition.

2.2 Types of NoSQL databases

There are four major NoSQL database types: key-value store, document store, column-oriented database, and graph database.

2.2.1 Key-value store

This is the least complicated NoSQL implementation. There are tables that contain two text columns. The first column is the key and the second is the value. They are based on a map that allows to add and read values by referencing a key. The advantage of this type of solution is the reading and writing speed. The disadvantage is the relatively low possibility of data segregation, so using such a database in everyday life is usually not recommended. Examples of such databases are: BerkeleyDB, LevelDB, Memcached, Project Voldemort.

2.2.2 Document store

In this NoSQL database type the data is stored in documents. Each document has a key-value structure. The advantage of this solution is high flexibility, which allows for faithful reproduction of real data in IT systems. The method of presenting and saving the data is more readable than in other types of databases. Examples of databases are MongoDB and CouchDB, Orient DB.

2.2.3 Column-oriented database

In this model, data is stored in columns instead of rows. This solution is used for storing large amounts of data, e.g., as data warehouse. The database is used by one of the most popular social networking sites e.g. Facebook was based on this model. It is called Cassandra. Other examples of columnar databases are Hypertable, Hadoop / Hbase.

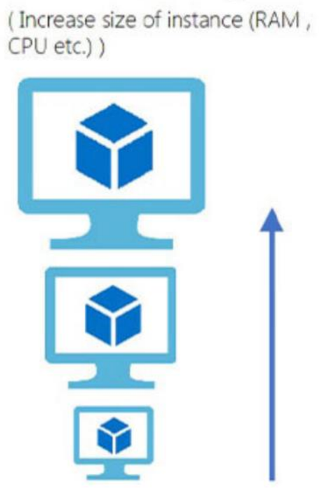
2.2.4 Graph database

These databases are based on graph theory. They don't use indexes, only pointers linked to adjacent elements. This allows for faster data searches in relation to relational databases. The most popular base of this type is Neo4j. Other examples are: FlockDB, HyperGraphDB, Infinite Graph.

2.3 Two types of scaling

Scalability is the ability to expand or decrease the capacity of system resources to support changing application usage. This can refer to both increasing and decreasing application usage. There are two approaches in database scaling

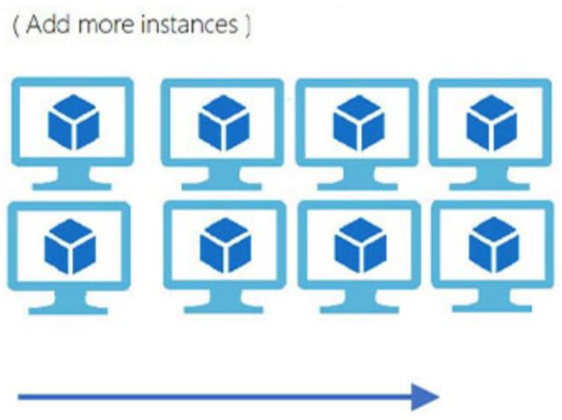
2.3.1 Vertical scaling



Vertical scaling is about increasing the computing power for one cluster. Such scaling is possible for both relational and non-relational databases, but its costs increase significantly with high-performance hardware. Therefore, such scaling is good if no bulk storage and processing is required.

Another benefit of vertical scaling is that nothing changes in the database infrastructure except the hardware specifications of the machine on which the database is running.

2.3.2 Horizontal scaling



Horizontal scaling refers to introducing additional nodes to share the load.

This is not used with relational databases due to the difficulty of distributing related data between nodes, but it is good option for non-relational databases. The collections are self-contained and not relational, so decompose them into nodes more easily because queries don't have to "join" them together between nodes.

3 When should NoSQL be used?

3.1 NoSQL features

3.1.1 Scalability

NoSQL databases have the ability to scale data horizontally dividing it into smaller fragments and distributing it on different servers. This means that when you need to add new resources, you can increase the number of servers. For this reason, NoSQL databases are used in big data.

3.1.2 Cost
NoSQL open-source databases provide affordable options for many organizations. It is perfect for cloud computing and handling very large and rapidly growing data sets, it scales horizontally, making it cost-effective to expand capacity. Instead of upgrading expensive hardware, they can grow cheaply by simply adding servers or cloud instances.

3.1.3 Flexibility/ Dynamic schemas
With greater freedom, speed, and flexibility in schema changes and queries, the NoSQL database makes it easier for developers to adapt to data requirements. Information stored in aggregate form allows for quick iterative improvements without the need to pre-design the schema.

In Relational databases structure and data types are fixed in advance. Each time you add a new feature in your application, the schema of your database changes. If the database is large, this is a very slow process and involves significant downtime. NoSQL databases are built to store data without a predefined schema. This makes it easy to make significant application changes in real-time.

3.2 NoSQL traps
So far, only the advantages of NoSQL have been listed, but it is not a database type that will work in all situations. It has many advantages, but it is also worth knowing that there are always two sides of the coin.

3.2.1 It forgives less
The speed of NoSQL databases is the result of appropriate data modeling, indexing, and partitioning. In a relational database, we could add columns, transform tables, move data from one table to another, add an index if we forgot about it before. In the case of NoSQL databases, this will not necessarily be possible. It is possible that we will have to use some external tools like Apache Spark.

3.2.2 No ACID
The ACID properties guaranteed on the database side simplify code writing. We don't need to handle errors related to the fact that table X already has data, and table Y data is not yet. Using NoSQL requires a different approach to data modeling and application logic. The code should be written in a more defensive way because it is not certain that the record you just changed is already available from another part of the application. Some databases advertise themselves as being consistent, and consistency is only ensured to a certain extent. MongoDB offers transactions, but only from version 4.0 it provides multiple document transactions.

3.2.3 Limited analytics and / or no JOINs
We are used to operating with GROUP BY or JOIN clauses, but not every database will offer such possibilities. Aggregations and joins can be very limited due to the nature of the database.

3.2.4 Schema management
Each NoSQL database approaches the schema in its own way. In some there is no schema, as in MongoDB, in some it can be dynamic, as in Elasticsearch, and in some it resembles the one from relational databases, as in Cassandra. In a conceptual model, the data ALWAYS has a schema. Entities, fields, names, types, relationships.

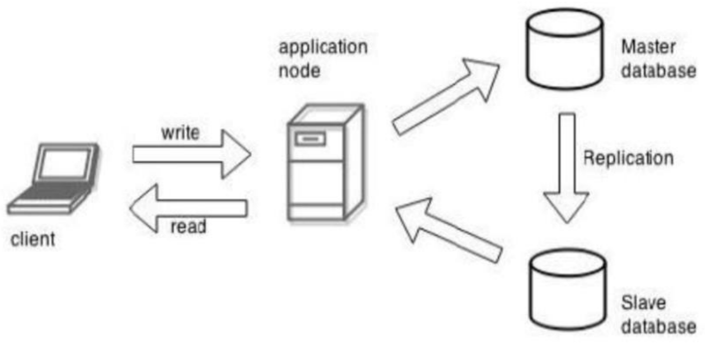
NoSQL databases give us freedom in terms of the schema. It can be a problem, because simple human error and the application malfunctions.

Another issue is the relationship between entities. Even if there is a schema, we must have documented dependencies between the data somewhere. On the basis of the relational database, we can generate an ERD diagram. For NoSQL databases, this may fail.

3.3 NoSQL Data Replication
You can replicate data in a NoSQL database using:

3.3.1 Master-slave NoSQL Data Replication

The NoSQL master-slave data replication technique creates a copy (master copy) of the database and maintains it as the key data source. Updates are made to this master copy and then sent to the slave copies. To maintain high performance, all read requests are managed by slave copies. In the event of a master copy failure, one of the slave copies is automatically assigned as a new master copy.



3.3.1.1 Pros of Using Master-slave NoSQL Data Replication:

- The Master-slave approach is extremely fast, and it doesn't operate on any performance or storage restrictions. Read and update tasks are divided among master and slave copies, it makes it possible to perform both operations in quick successions without facing any time delay.
- It can be used to split the data read and write requests and allocate them to different servers. This will improve the data processing speed and efficiency.

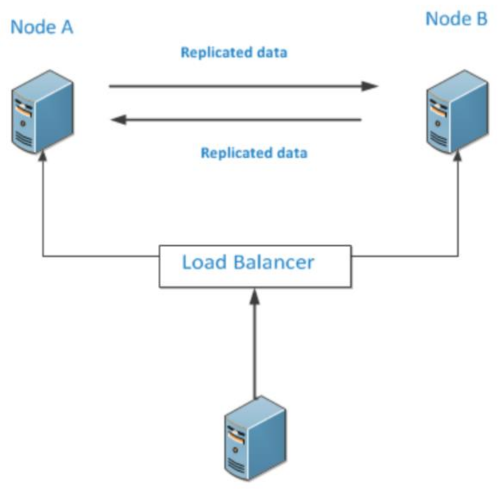
3.3.1.2 Cons of Using Master-slave NoSQL Data Replication:

- This technique works asynchronously and is therefore not always reliable. In cases where the master copy fails, some committed transactions will disappear, and no slave copy will contain this information.
- The master-slave technique does not support high scaling of write requests.

3.3.2 Peer-to-Peer NoSQL Data Replication

Peer to peer replication is built on the concept of transaction replication, which propagates consistent transactional data. Peer to peer replication involves Multiple servers are called as (nodes). In this, each node acts as a publisher as well as a subscriber that means it receives and sends transactions to other nodes, data is synchronized across the nodes.

If any node goes down, the application will still be functional. Later, once the node is up it can be again brought in sync. This way we can achieve both high availability and fault tolerance.



3.3.2.1 *Pros of Using Peer-to-Peer NoSQL Data Replication:*

- If a node fails, the application layer can commute that node's read requests to other adjacent nodes and maintain a lossless processing environment and data availability.
- The catalog queries are stored across multiple nodes; the performance of Peer-to-Peer NoSQL Data Replication remains constant even if your data load increases.

3.3.2.2 *Cons of Using Peer-to-Peer NoSQL Data Replication:*

- Replicating changes is costly in terms of latency in Peer-to-Peer replication.
- During modifying a particular row at more than one database node, it can cause a data loss by triggering a conflict

3.4 SQL VS NoSQL

3.4.1 Differences in features

NoSQL as SQL is not ideal, each of them is designed for a certain type of problems, when choosing the right type of databases, pay attention to the differences between them parameters.

Feature	**SQL Base**	**NOSQL Base**
Type	Relational Base	Non-relational base
Structure	Data is stored in tables with a certain numbers of columns	Document, key-value, graph...
Type of data structure	Fixed data structure	Variable data structure
Scalability	Vertical	Horizontal
Inquiries	Usually SQL	There is no declared query language
Important features	Data integrity, consistency, stability	Scalability, flexibility, quick inquires

3.4.2 ACID vs BASE

3.4.2.1 *ACID*

ACID is a set of properties that guarantee the correct processing of transactions in databases. ACID is an abbreviation from words atomicity, consistency, isolation, durability.

ATOMIC
CONSISTENT
ACID
ISOLATED
DURABLE

Atomic – each transaction will either be carried out in full or not at all.

Consistent – a processed transaction will never endanger the structural integrity of the database.

Isolated – if two transactions execute concurrently, they usually, depending on the isolation level, do not see the changes they are making. The isolation level in databases is usually configurable and determines what anomalies can be expected when executing transactions.

Durable – the system can start up and provide consistent, intact, and up-to-date data recorded as part of committed transactions, for example after a sudden power failure.

Relational database management system is ACID compliant. These include MySQL, PostgreSQL, Oracle, SQLite, and Microsoft SQL Server. ACID databases are used for example by financial institution.

3.4.2.2 BASE

The rise of NoSQL databases provided a flexible and fluid way to manipulate data. As a result, a new database model was designed. BASE means Basically Available, Soft State and Eventually Consistent.

BASICALLY AVAILABLE
SOFT STATE
BASE
EVENTUALLY CONSISTENT

Basically Available – means that rather than enforcing immediate consistency, BASE-modeled NoSQL databases ensure data availability by propagating and replicating across the database cluster nodes.

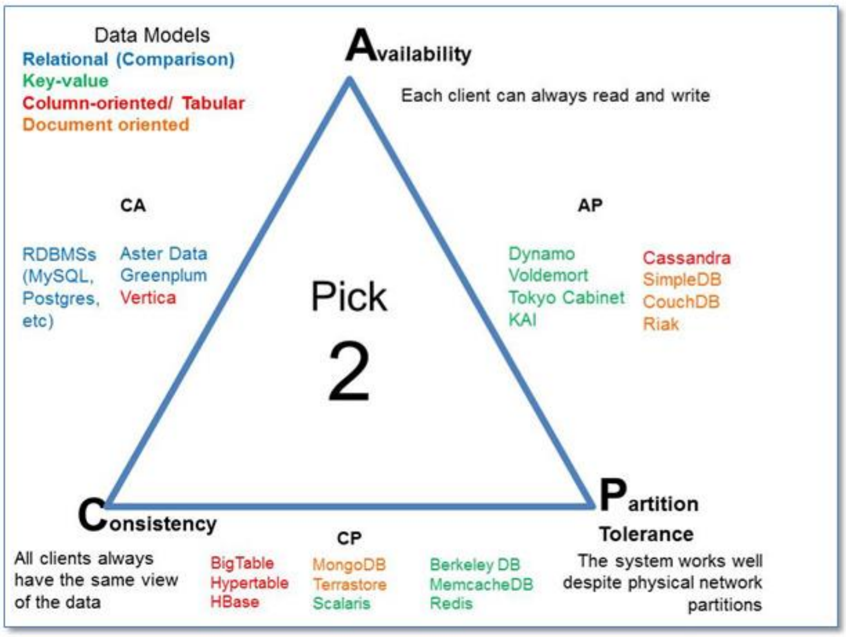
Soft State – Indicates that the state of the system can change over time, even without making changes to the data

Eventually Consistent – means that the system will become coherent after some time (e.g. that two nodes will be synchronized after some time), even when no data is entered / changed / deleted into the system. one of the nodes is unavailable

NoSQL databases tend to conform to BASE principles.
BASE will be preferable for marketing and customer service companies that deal with sentiment analysis during conducting social network research. Social network channels are not well structured but contain huge amounts of data that a BASE modeled database can easily store.

3.4.3 CAP theorem

According to cap, a distributed system can only provide two of the three desirable characteristics: consistency, availability, and partition tolerance.



Traditional relational databases (MySQL, PostgreSQL, etc.) are located on the side of the CA of the triangle, they emphasize the consistency and availability of data, but they scale primarily vertically. The other sides are covered mainly by NoSQL databases.

4 Representatives of NoSQL databases

4.1 MongoDB

4.1.1 Features of MongoDB

MongoDB is a document-based NoSQL database developed since 2009. It uses collections and documents in place of tables and rows from relation-based databases. Documents are created and stored in BSON, or Binary JSON, format. The use of JSON means that it is extremely easy to convert the queries and results to a format that understands the frontend code. It is also more human readable. This NoSQL solution includes hierarchy, automatic fragmentation, and built-in replication for better scalability and high availability.

Each database is a set of collections which consists of documents within them. Documents do not have predefined schema; fields are added while populating collection with data. It is possible to store complex structures represented by hierarchical relationships.

4.1.1.1 Replication

MongoDB uses a concept called Replica Set, which is a set of nodes containing the same data. This enables data replication, the purpose of which is to increase the availability and protect against database server failures. Well-designed architecture also allows faster access to data. The heartbeat mechanism is essential to the entire replication concept. Each of the nodes (members) polls the remaining nodes every 2 seconds to check their availability. If the main server is unavailable, a new one is selected. This process consists in selecting the one with the highest priority from among the remaining instances. The documentation states that the replica can have up to 50 nodes, of which only 7 can participate in voting, and the successor is selected from among them. The remaining servers, called non-Voting members, must have votes and priority set to 0. It is recommended that the number of instances with voting be odd, so the minimum number of nodes in a replica is 3.

4.1.2 Use cases

- MongoDB and Product Data Management - E-Commerce
  MongoDB's flexible schema is ideal for storing and managing a product catalogues. It is also possible to

handle shopping cart – inventory interaction.
eBay uses MongoDB for its search suggestion feature and managing its enormous amount of customer data.

- MongoDB and Shutterfly, and FIFA
  MongoDB allows to store and synchronize multiple clusters while also efficiently managing transactions. This feature is being used by Shutterfly, a company hosting online photo sharing platform with over six billion images. Another platform using MongoDB due to its horizontal scalability is Electronic Arts' FIFA online which uses over 250 servers allowing access to millions of players.

4.1.3 DynamoDB

Features of DynamoDB

Amazon DynamoDB is the perfect solution for applications that require incredibly low latency and reliability. This tool is designed for super-efficient applications. Moreover, it is extremely easy to configure, cost-effective and, which is the norm among AWS services, fully scalable.

The user can start using it using only minimal resources, later increasing them can take place in real time, depending on the growing requirements of the applications being created. It is also worth mentioning that DynamoDB has built-in mechanisms that enable data partitioning on different servers to be able to handle even the most demanding queries. We also emphasize that Amazon DynamoDB was created in such a way that it was possible to create document databases and key-value.

Amazon DynamoDB will be perfect for users who:

- create demanding applications that require low delays in writing and reading data
- require unlimited scaling
- they use the key-value mechanism to access data
- want to delegate the trouble of database administration to AWS

4.1.4 Use cases

- DynamoDB and Amazon Games
  Amazon Games is responsible for creating a massive multiplayer online game "New World." They are using DynamoDB for saving over 800,000 game states every 30 minutes on over 500 servers and DynamoDB is capable of performing that flawlessly.
- DynamoDB and Duolingo
  DynamoDB's availability, high performance and DevOps support was a perfect fit for the Duolingo team. 30 billion data objects, 24,000 read units/s and 3,300 write units/s is being supported by DynamoDB. Another huge advantage of this data base is its automatic scaling which is perfect for small startup that do not want to spend time and resources on manually adjusting the size.

4.2 Redis

4.2.1 Features of Redis

Redis, or more specifically the Remote Dictionary Server, is a key-value database. It provides data structures in RAM - not on a hard disk. It also includes structures such as strings, lists, string sets, ordered string sets, hashes, bitmaps, data related to objects in space, as well as data streams. Redis as a database works with data in RAM, but it is possible to transfer data to disks. Using this extensive database gives the user a lot of possibilities. The operation is quite simple, so even less experienced people will be able to manage Redis. By far the biggest advantages are its excellent performance and multitasking. In this database, we do not work on disks so searching for individual elements in the database is much faster. The response time is counted in microseconds, while Redis itself can perform several million requests per second.

4.2.2 Use cases

- Redis and mobile games
  Growth of the gaming market is currently based on mobile gaming, thanks to Redis' low latency millions of

players can share real-time leaderboards. Mobile game developer Scopely uses Redis for leaderboard, API management and queue workload management. It supports various data structures, caching, and high availability.

- Redis and E-commerce
  Garment retailer Gap Inc. wanted to provide their e-commerce customers with real-time shipping information for each product added by buyers to their cart. The company encountered issues with delays and inaccurate inventory information. This problem resulted in poor customer service that inflated costs and undermined brand loyalty. Application developers at Gap Inc. found Redis Enterprise's linear scalability and sub-millisecond performance on a massive scale to be a huge help.

4.3 ArangoDB

4.3.1 Features of ArangoDB

ArangoDB is an open-source native multi-model database. It supports graphs, documents, and key-value data models, allowing users to freely combine all data models in a single query. Using a multi-model database allows you to combine up to three types of NoSQL that better fit the way your data is used by different components within your application. The database implements both vertical and horizontal scaling allowing you to quickly adapt to growing requirements. It also supports independent scaling of different data models allowing you to scale down your application to save on expenses. Multi-model databases allow the use of different storage technologies that better fit the way data is used by different components in the application. ArangoDB uses the binary JSON format to store data. It is lean, self-contained, does not allocate too much memory, and covers all JSON plus dates, binary data, integers as well as arbitrary precision numbers. ArangoDB allows access to any data (regardless of its model) with a single declarative query language (AQL) that is like SQL.

4.3.2 Use cases

- ArangoDB and Oxford University
  The Institute of Biomedical Engineering of the University of Oxford, in partnership with the Oxford University Hospitals NHS Trust funded by the National Institute for Health Research is developing a system for cardio-pulmonary patients. They are developing an app that lets patients take test at home guided by their mobile phone application. ArangoDB was chosen as the database for that project because of its compatibility with Node.JS, allowing to store multiple data models and most importantly its SQL-like query language because the team includes researchers who need to explore the data efficiently for research purposes and without the need of developing complex APIs.
- ArangoDB and FlightStats
  FlightStats is an innovative data services company focused on commercial aviation. Providers of real-time global flight data to companies and individuals across the travel ecosystem. All major airlines, airports, hotels and leading information providers like Google or Yahoo use FlightStats´ products to improve their services reaching over 35% of global air travelers each day and exceeding 300 million data requests each month. Reference data is now used for all data products that relate to airports, airlines, and equipment. It is particularly important that the data is accurate and up to date. ArangoDB makes it easy to add temporal efficiency to your data - which is useful for both historical reporting and making changes that soon might be effective. AQL is a powerful query language, and its intuitive nature makes it easy to customize. In a dynamic environment, fast iteration and prototyping are especially important. With the Foxx microservice framework, basic environments are ready within hours. Good documentation means that a growing team can quickly introduce new members.

4.4 Apache Cassandra

4.4.1 Features of Cassandra

Apache Cassandra is an open-source distributed database management system. It is designed to manage large amounts of distributed data across multiple servers, which will continue to function even if one of the servers goes down. The Cassandra Query Language (CQL) is like SQL. This means that most developers should get used to it easily. With Cassandra, no single node is responsible for data replication in the cluster. Instead, each node can perform read and write operations. This improves performance and increases the reliability of the database. Data is replicated rapidly across hybrid environments. If a node fails, users will automatically be directed to the nearest good node. They will not even notice that the node has been shut down because the applications behave as designed, even in the event of a crash. This ensures that applications are always available, and data is never lost. Cassandra's built-in repair services fix problems as soon as they occur without an intervention of a developer. Cassandra allows you to scale horizontally by simply adding more nodes to the cluster.

4.4.2 Use cases

- Cassandra and Spotify
  Spotify uses Cassandra to store user profile attributes and metadata about entities like playlists, artists, etc. The project started with a small data size, but along the way, storage capacity was easily expanded by increasing the number of nodes in the cluster. Spotify is available in 183 countries around the world, with backend services supporting data centers in all regions. To ensure that users can continue to be served by personalization system in case of failure of any data center, NetworkReplicationStrategy is used to personalize clusters for data replication between data centers. This allows users to access data at the nearest Spotify data center. Considering the Spotify user base, real-time processing of personalized data on the user's listening results in a large amount of data to be stored in the database. Real-time computed personalization data is not transactional, and lost data can be easily replaced with new data in minutes from the user's listening stream, they could fine-tune the consistency level of read and write operations to sacrifice consistency for even less latency (by not waiting for all replicas responded before the operation was successful). Cassandra offers options for bulk importing data from other data sources by building SSTables and then streaming the tables to the cluster. It is much simpler, faster, and more efficient than sending an individual INSERT statements for all the data you want to load into Cassandra.

4.5 Neo4j

4.5.1 Features of Neo4j

Neo4j is one of the most popular, if not the most popular graph database. The graph is a composition of two types of elements, which are nodes and relations. A node may represent a specific type or several types and has its own properties. Relationships, apart from the name and their own properties, have - most importantly - the direction of their influence. These properties are collections of key-value pairs. They are used to store relevant information. For example: if the node is a person, its properties can be name, surname, age or a list of favorite books. Relationships between nodes in Neo4j are just as important data as nodes. We treat them as objects whose existence is determined by the presence of the given nodes. There is no justification for the existence of an independent relationship. Cypher is the query language used in Neo4j. People who have had the opportunity to use SQL will find this language familiar. This query language uses ASCII-Art to create patterns that make Cypher more readable. Neo4j supports ACID to fully support data integrity and ensure good transaction behavior. All data operations, such as access to a graph, indexes, or schema, should be performed in a transaction. The data collected while viewing the graph is in no way protected against modification by another transaction. Non-repeatable reads may occur - only write locks are applied during transactions. You can manually lock nodes and relationships to achieve a higher level of isolation. Deadlock detection is a mechanism built into the transaction management system. Transactions in Neo4j use the READ_COMMITED level. In addition, the Java API provides the ability to clarify locks on nodes and relationships. Interlocks provide the ability to simulate higher levels of isolation by placing and removing interlocks.

4.5.2 Use cases

- Neo4j and eBay
  eBay has chosen Neo4j as their native graph database which includes probabilistic models to help understand the conversational buying scenario. The Neo4j graph contains both the product catalog and the attributes of buyers' interactions when searching for products. For example, if a buyer searches for "brown

bags," the eBay application knows what details to ask for, such as type, style, brand, budget, or size. As it collects this information as you it moves through the graph, the app continuously checks the inventory for the best match. eBay engineers knew that delivering a chatbot to their user base required a highly resilient and accessible web scale, predictable responses in the millisecond range. This resulted in Neo4j, which includes high-availability clustering and exceptional read and write performance. Even with millions of nodes, the application responds very quickly to user requests.

- Neo4j and Zurich Insurance Group
  Zurich Switzerland is part of the Zurich Insurance Group. As for 2021 more than 1.4 million customers in Switzerland, a total of 6,100 employees answer a million calls at the claims center each year and process 500,000 claims. Neo4j is often used in banking industry for fraud detection, Zurich Switzerland triages potential fraud cases using data stored in Neo4j database which manages about 20 million nodes and 35 million relationships. The combination of a rule-based risk system, charting database and chart visualization allowed Zurich Switzerland to automate and speed up fraud detection. Most cases can be closed within the first day. Employees significantly reduce the time spent on segregation and easily save 10 minutes per case. The potential of charting technology is far from exhausted. Researchers are already using graph algorithms such as Shortest Path to identify the shortest connection between people or companies. The company's goal is to minimize the false positive rate and improve automation.

5 COMPARISON

FEATURE	ARANGODB	MONGODB	CASSANDRA	DYNAMODB	NEO4J
GENERAL					
INITIAL RELEASE	2012	2009	2008	2012	2007
LICENCE	Apache 2 / Commercial	AGPLv3 / Commercial	Apache V2 / Commercial	Commercial	AGPLv3 / Commercial
WRITTEN IN	C++	C++	Java	Java	Java
DATA-MODEL	Multi-model documents, graphs, key-value	Document	Partitioned Row-store	Document, key-value	Graph
SCHEMA FREE	Yes schema validation with Foxx	Yes additional schema validation	No	Yes	Yes
DATA FORMAT	JSON / VelocyPack	JSON / BSON	Tabular	JSON	JSON
STORAGE ENGINE	MMFiles / RocksDB	MMAPv1 / WIREDTIGER	SST	Partitions	Neo4j graph storage
JOURNALING	Yes	Yes	Yes	Yes	No
CLUSTER					
AUTO-SHARDING	Yes	Yes	Yes	Yes	No
REPLICATION	Sync / async	Async	Async	Sync	Async
REPLICATION CONFLICT RESOLUTION	Master/Master Master/Agent	Master/Agent	Master/Master	Master/Master	Master/Agent
ELASITC SCALABILITY	Yes	No	Yes	Yes	No
ZERO CONFIGURATION	Yes	No	No	Yes	No
NATIVE KUBERNETES / MESOS INTEGRATION	Yes	No	Yes	Yes	No

FEATURE	ARANGODB	MONGODB	CASSANDRA	DYNAMODB	NEO4J
TRANSACTIONS					
TRANSACTION MODEL	ACID	ACID	BASE	ACID	ACID
MULTI-DOCUMENT TRANSACTIONS	Yes	Yes	No	Yes	Yes
MULTI-COLLECTION TRANSACTIONS	Yes	Yes	No	Yes	Yes
QUERYING					
DECLARATIVE QUERY LANGUAGE	Yes, AQL	No	Yes, CQL	No	Yes, Cypher
GRAPH TRAVERSALS	Yes	-	Additional integrations needed	-	-
JOINS	Yes	Aggregation Framework	Additional integrations needed	No	-
RELATIONS	Edges	-	-	-	Edges
RELATIONAL JOINS	Yes	-	-	-	no
ADVANCED PATH-FINDING WITH MULTIPLE ALGORITHMS	Yes	No	Additional integrations needed	Amazon CloudSearch Console	Yes
TINKERPOP SUPPORT	Yes	No	No	No	Yes
TEXT SEARCH (INDEXING / QUERIES)	Yes	Yes	-	Yes	Yes
GEOSPATIAL (INDEXING / QUERIES)	Yes	Yes	Additional integrations needed	Yes	Yes
EXTENSIBILITY					
MICROSERVICE SUPPORT	Yes	No	No	Yes	No
SERVER-SIDE FUNCTIONS	Yes	Yes	Yes	Yes	Yes
SECURITY					
ENCRYPTION	TLS / SSL	TLS / SSL	TLS / SSL	TLS / SSL	no
AUTHENTICATION	Yes	Yes	Yes	Yes	Yes
ROLE-BASED ACCESS CONTROL	Yes	Yes	Yes	Yes	No
AUDITING	Yes (enterprise)	Yes (enterprise)	Yes (enterprise)	Yes	No
ADMINISTRATION					
WEB-BASED GUI (SELF-CONTAINED)	Yes	No	No	No	Yes

6 CONCLUSION

SQL systems and relational databases are very popular due to the general mechanisms for handling data management. They have been designed to be reliable. However, some SQL and relational requirements - such as a rigid schema and strict ACID - can make them less suitable for applications that require flexible data and high performance.

In response to these needs, NoSQL database systems were created, many of which were developed by huge companies such as Amazon with DynamoDB or Facebook and Apache Cassandra.

They provide many benefits, including flexible data models, flexible horizontal scaling, and ease of use for developers. NoSQL databases come in different types, including document databases, key-value databases, wide-column stores, and graph databases suitable for different use cases.

Tens of thousands of smaller businesses and start-ups have adopted NoSQL. Almost every business field is implementing some type of NoSQL database that’s specifically designed to their needs. Many companies discover surprising possibilities with graph-based NoSQL DB.

With NoSQL, enterprises are better able to grow agile and operate at any scale, and deliver the performance and availability required to meet the demands of digital economy enterprises.

7 SOURCES

https://www.ksi.mff.cuni.cz/~svoboda/courses/2016-2-MIE-PDB/lectures/Lecture-06-NoSQL.pdf
https://rperlinski.pl/strona/files/tzt/w02_NoSQL.pdf
https://redis.com/nosql/what-is-nosql/
https://technologypoint.in/advantages-and-disadvantages-of-nosql-databases/
https://www.jcommerce.pl/jpro/artykuly/nosql-vs-sql-bazy-danych
https://www.datastax.com/blog/sql-vs-nosql-pros-cons
https://www.dataversity.net/a-brief-history-of-non-relational-databases//
https://azure.microsoft.com/pl-pl/resources/cloud-computing-dictionary/what-is-nosql-database/
https://www.mongodb.com/databases/scaling
https://www.webairy.com/horizontal-and-vertical-scaling/
https://wiadrodanych.pl/bazy-danych/5-pulapek-nosql/
https://blog.e-zest.com/dynamic-schema-in-nosql/
https://hevodata.com/learn/nosql-data-replication/#Methods-picture
https://www.sqlshack.com/peer-to-peer-replication/-pictures
https://severalnines.com/blog/introduction-redis-what-it-what-are-use-cases-etc/
https://www.mindk.com/blog/arangodb/
https://www.scylladb.com/learn/dynamodb/introduction-to-dynamodb/
https://www.devopsschool.com/blog/what-is-arangodb/
https://ubuntu.com/blog/what-is-mongodb
https://www.bmc.com/blogs/neo4j-graph-database/
https://www.jcommerce.pl/jpro/artykuly/mongodb-nosql-w-ecommerce
https://www.datastax.com/what-is/cassandra
https://engineering.atspotify.com/2015/01/personalization-at-spotify-using-cassandra/
https://www.demandsage.com/spotify-stats/
https://cassandra.apache.org/_/case-studies.html
https://www.computerweekly.com/news/4500254443/Keeping-up-with-Cassandra-the-NoSQL-database
https://www.datastax.com/blog/apache-cassandra-benchmarking-40-brings-heat-new-garbage-collectors-zgc-and-shenandoah
https://www.jcommerce.pl/jpro/artykuly/neo4j-zaproszenie-do-grafowych-baz-danych
https://www.bbvaapimarket.com/en/api-world/neo4j-what-graph-database-and-what-it-used/
https://go.neo4j.com/rs/710-RRC-335/images/Neo4j-Case-Study-Zurich-EN-A4.pdf
https://www.arangodb.com/solutions/comparisons/arangodb-vs-mongodb/
https://www.linkedin.com/pulse/20140612181108-4788696-5-reasons-why-dynamodb-is-better-than-mongodb/?lipi=urn:li:page:d_flagship3_profile_view_base_post_details;a3bQyARaQOKJzmnzViAQYA
https://www.bmc.com/blogs/mongodb-vs-dynamodb/
https://www.integrate.io/blog/mongodb-vs-redis/
https://dynobase.dev/dynamodb-vs-redis/
https://brainhub.eu/library/arangodb-use-case
https://naiveskill.com/mongodb-vs-neo4j/
https://cloudinfrastructureservices.co.uk/cassandra-vs-dynamodb-whats-the-difference/
https://www.arangodb.com/solutions/comparisons/arangodb-vs-cassandra/

7.1 PICTURES SOURCES

https://medium.com/design-microservices-architecture-with-patterns/scalability-vertical-scaling-horizontal-scaling-adb52ff679f
https://www.sqlshack.com/peer-to-peer-replication/
https://phoenixnap.com/kb/acid-vs-base
http://staff.uz.zgora.pl/agramack/files/BazyDanych/NoSQL/NoSQL.pdf