NoSQL

Introduction to NoSQL: Lecture 3

Piotr Fulmański

FACULTY OF MATHEMATICS
AND COMPUTER SCIENCE
University of Lodz

NoSQL Theory and examples

by Piotr Fulmański

Piotr Fulmański, 2021

PIOTR FULMAŃSKI

NoSQL
Theory and examples

SIMPLE INTRODUCTION SERIES

- The most crucial motivation and its consequences
- A radical change in mindse
- BASE, CAP and consistency

What is NoSQL?

- It’s much more than rows in tables. NoSQL systems store and retrieve data from many formats: key-value, graph, column-family, document, and of course rows in tables.
- It’s free of joins. NoSQL systems allow us to extract our data using simple interfaces without joins.
- It’s schema-free. In most cases we don't have to create an entity-relational model.
- NoSQL systems are easily scalable.
    - In consequence they works on multiple processors and can run well on low-cost separate computer systems - no need for expensive nodes to get high-speed performance.
    - Scalability supported by NoSQL systems is closed to linear. In consequence increasing the number of processing units by factor of N we get increase in performance by factor close to N.
- NoSQL is a response to nowadays business data related factors:
    - volume and velocity, referring to the ability to handle large datasets that arrive fast;
    - variability, referring to how diverse data types don’t fit into structured tables;
    - and agility, referring to how fast an organization responds to business changes.

What is not NoSQL?

- It’s not against the SQL language. SQL as well as other query languages are used with NoSQL databases.
- NoSQL is not Big Data.
- It’s not about cloud computing.
- It’s not close group of companies, software or product. Anybody can be a big player in this market if only offer innovative solutions to business problems.

Motivations

Availability and speed

Today people don’t read. They scan.

Application	2016	2018
airbnb	words: 6, font size: 43	words: 4, font size: 91
tinder	words: 6, font size: 50	words: 3, font size: 112
Spotify	words: 4, font size: 45	words: 3, font size: 100
Pinterest	words: 10, font size: 40	words: 3, font size: 90

Motivations

Scalability

If we agree that availability and speed is a key feature we must have, this raises a short question: *How?*

The answer: *use scaling*.

Scalability is the property of a system to handle a growing amount of work by adding resources to the system.

When availability is considered, easy scalability is one of the most required feature.

Motivations

Scalability - how it is possible: no joins

If we agree that scalability is a key feature we must have, this raises again the same short question: *How?*

> *What should characterize data to make their parallel processing possible?*

Data should not be dependent each other. If we add, delete or update one part of it, it should not affect any other part. There should be no connections between data parts.

One of well known connection of that type, linking data and allowing traversal through them are relational foreign keys. Keys are the fundamental mechanism used to combine data distributed among tables.

Unfortunately what is necessary for the relational databases to function, at the same time is its limitation. We can’t operate on data in parallel as we have to obey keys constraint. To use any piece of data almost always we have to join them by keys. Without keys and joins nothing can’t be done in relational systems.

Motivations

No joins, no schema... Schema-free trap

At the first sight you may feel happy as in NoSQL no schemas are required. You don’t have to think ahead how you split your date into tables and columns. You are finally free from having to think about normal forms.

Motivations

No joins, no schema... Schema-free trap

At the first sight you may feel happy as in NoSQL no schemas are required. You don’t have to think ahead how you split your date into tables and columns. You are finally free from having to think about normal forms.

If you think that this way your life would be simpler and easier from now, you are wrong. Completely wrong!

Motivations

No joins, no schema... Schema-free trap

At the first sight you may feel happy as in NoSQL no schemas are required. You don’t have to think ahead how you split your date into tables and columns. You are finally free from having to think about normal forms.

If you think that this way your life would be simpler and easier from now, you are wrong. Completely wrong!

If you think that schema-free freedom allows you to do what you want and when you want, you are on the straight way to total disaster.

Motivations

No joins, no schema... Schema-free trap

At the first sight you may feel happy as in NoSQL no schemas are required. You don't have to think ahead how you split your date into tables and columns. You are finally free from having to think about normal forms.

If you think that this way your life would be simpler and easier from now, you are wrong. Completely wrong!

If you think that schema-free freedom allows you to do what you want and when you want, you are on the straight way to total disaster.

In SQL you never ask *who* and *how* is going to use data. In NoSQL both questions are the most fundamental and future actions depend on the answers to these questions.

Motivations

No joins, no schema... Schema-free trap

Data modeling in Apache Cassandra column store

Data modeling is the process of identifying entities and their relationships. In relational databases, data is placed in normalized tables with foreign keys used to reference related data in other tables. Queries that the application will make are driven by the structure of the tables and related data are queried as table joins.

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

Data is modeled around specific queries.

Motivations

No joins, no schema... Schema-free trap

A radical change in mindset

No schema doesn't mean no rules. **We have to change our point of view from being universal of one size fits all type to being precise and focused.**

While in SQL databases whatever task we do, our database has the same form, in NoSQL data is modeled around specific queries.

Query-driven data modeling means that the data access patterns and application queries determine the structure and organization of data.

Motivations

Query-driven data modeling... Aggregations

Data modeling in Apache Cassandra column store

Data modeling is the process of identifying entities and their relationships. In relational databases, data is placed in normalized tables with foreign keys used to reference related data in other tables. **Queries that the application will make are driven by the structure of the tables and related data are queried as table joins.**

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

Data is modeled around specific queries. Queries are best designed to access a single table, which implies that all entities involved in a query must be in the same table to make data access (reads) very fast. Data is modeled to best suit a query or a set of queries. A table could have one or more entities as best suits a query. As entities do typically have relationships among them and queries could involve entities with relationships among them, **a single entity may be included in multiple tables**.

Motivations

Query-driven data modeling... Aggregations

Query-driven modeling in Apache Cassandra column store

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.

Motivations

Query-driven data modeling... Aggregations

Because in NoSQL there are no keys and joins we can’t combine data coming from multiple sources (multiple tables) into one group provided to the user.

Instead **we organize our data into one self-contained group**. Whatever we want to get with one query should be enclosed within it.

We shouldn’t think in terms of data organized into tables according to normal forms rules but rather in terms of **atomic group holding all data accessed together with just one call**.

Such a base unit of data organization, an atomic and self-contained group of data, is named ***aggregate***. Under the name *aggregation* we understand the whole process of transforming our data into such a units.

Aggregate should contain all data needed to complete single request. And that should be only that data which are really needed.

Motivations

Query-driven data modeling... Aggregations

Consider a group of people taking part in different projects:

```
person A: project 1
person B: project 1
person B: project 2
person C: project 2
person A: project 3
person C: project 3
```

Motivations

Query-driven data modeling... Aggregations

If we expect or know that information about persons involved in the project will be needed, we should operate with the following aggregates:

```
{
  {project: project1,
   person: [personA: {...},
            personB: {...}]},

  {project: project2,
   person: [personB: {...},
            personC: {...}]},

  {project: project3,
   person: [personA: {...},
            personC: {...}]}
}
```

If additionally we need an information about the total working time of each person in all their projects, we should use another aggregates:

```
{
  {person: personA,
   totalWorkingTime: 123},

  {person: personB,
   totalWorkingTime: 27},

  {person: personC,
   totalWorkingTime: 62}
}
```

BASE

Query-driven data modeling... Aggregations

***B**asic **a**vailability* means that the database appears to work most of the time. It allows systems to be temporarily inconsistent so that transactions are manageable. In BASE systems, the information and service capability are basically available. This means that there can be a partial failure in some parts of the distributed system but the rest of the system continues to function.

***S**oft-state* means that stores don’t have to be write-consistent, nor do different replicas have to be mutually consistent all the time. Some inaccuracy is temporarily allowed and data may change while being used. State of the system may change over time, even without input. This is because of eventual consistency.

***E**ventual consistency* means that there may be times when the database is in an inconsistent state. Eventually, when all service logic is executed, the system is left in a consistent state.

Contrary to ACID systems, which are **pessimistic** and because of this are ready to survive any disaster we can ever imagine, BASE are **optimistic** as they assume that eventually, in not so distance future, all systems will catch up and become consistent.

CAP theorem

The CAP theorem is about how distributed database systems behave in the face of network instability.

According to the CAP theorem introduced by Eric Brewer in 2000, any distributed database system can have at most two of the following three desirable properties.

Consistency. Consistency is about having a single, up-to-date, readable version of our data available to all clients. Our data should be consistent – no matter how many clients read the same items from replicated and distributed partitions, we should get consistent results. All writes are atomic and all subsequent requests retrieve the new value.

High availability. This property states that the database will always allow clients to make operations like select or update on items without delay. Internal communication failures between replicated data shouldn't prevent operations on it. The database will always return a value as long as a single server is running.

Partition tolerance. This is the ability of the system to keep responding to client requests even if there's a communication failure between database partitions. The system will still function even if network communication between partitions is temporarily lost.

The CAP theorem helps us understand that **once we partition our data, we must determine which of two exclusive options best match our business requirements: consistency or availability.**

Consistency

Working with NoSQL system we have to remember that our data is only guaranteed to be almost accurate. This makes a big difference compared to relational system where data is always accurate.

- **Casual consistency.** Casual consistency means that the database reflects the order in which operations were updated.

- **Read-your-writes consistency.** Read-your-writes consistency means that once we have updated a record, all of our subsequent reads of that record will return the updated value.

- **Session consistency.** Session consistency means read-your-writes consistency but at session level. Session can be identified with a conversation between a client and a server. As long as the conversation continues, we will read everything we have wrote during this conversation. If the session ends and we start another session with the same server, there is no this guarantee that we can read values we have wrote during previous conversation.

- **Monotonic read consistency.** Monotonic read consistency means that whenever we make a query and see a result, we will never see an earlier version of the value.

- **Monotonic write consistency.** Monotonic write consistency means that every time we make several update commands, they would be executed in the order we issued them.

Bibliography

- [Ful] Piotr Fulmański, *NoSQL. Theory and examples*, Piotr Fulmański, 2021