

NoSql databases



Datainsights · Follow

8 min read · Jul 28, 2024



1



Introduction to NoSQL Databases

NoSQL databases have gained prominence in the world of data management, providing an alternative to traditional relational database management

systems (RDBMS). The term “NoSQL” stands for “Not Only SQL,” emphasizing the flexibility of these databases to support a variety of data models, including key-value, document, column-family, and graph formats. NoSQL databases emerged to address the limitations of RDBMS in handling the increasing volume, variety, and velocity of data generated in the digital age.



Open in app ↗

Medium

Search

Write



CouchDB
relax



Historical Context

The origin of NoSQL databases can be traced back to the early 2000s, a period marked by the rapid growth of web-based applications and services. Traditional relational databases, which rely on structured schemas and SQL queries, struggled to keep up with the needs of these applications, particularly in terms of scalability and performance. Companies like Google, Amazon, and Facebook, which were at the forefront of this data explosion, spearheaded the development of new database technologies that could handle large-scale, distributed, and unstructured data.

Key Characteristics and Advantages

1. **Scalability:** One of the most significant advantages of NoSQL databases is their ability to scale horizontally. Horizontal scalability is increasing capacity by adding more computers to the system. This means that as the amount of data grows, additional servers can be added to distribute the load, thereby enhancing performance and storage capacity.
2. **Flexibility:** Unlike RDBMS, which requires predefined schemas, NoSQL databases allow for dynamic schema design. This flexibility makes it easier to handle unstructured and semi-structured data, accommodating changes in data types and structures without the need for extensive modifications. However, some NoSQL databases, such as column-family stores like Apache Cassandra, do require schemas, but they still offer flexibility in terms of allowing different rows to have different structures and dynamically adding columns.
3. **High Performance:** NoSQL databases are designed to deliver high performance for read and write operations. Their architecture supports rapid data access and can handle large volumes of concurrent transactions, making them ideal for real-time applications.
4. **Reduced Complexity:** The ability to store data in a more natural and intuitive format reduces the complexity of database design and management. NoSQL databases often use object-oriented programming paradigms, which can simplify the development process.

Types of NoSQL Databases

Key-Value oriented databases:

A key-value store is a type of NoSQL database that uses a simple key-value pair to store data. Each key is unique and is used to retrieve the associated

value. This model is highly efficient for lookups and is ideal for applications where simplicity and speed are paramount.

Example Using Redis

Redis is a popular key-value store known for its speed and versatility. Here's an example of how you might use Redis in a simple web application to store and retrieve user session data.

Storing User Session Data

Let's say you have a web application where users log in, and you want to store session information for each user.

1. **Set a Session:** When a user logs in, you create a session ID and store user information associated with this session ID.

```
SET session:12345 { "user_id": "42", "username": "john_doe",  
"email": "john.doe@example.com" }
```

2. **Retrieve a Session:** When you need to retrieve the session information (e.g., to verify the user's identity), you use the session ID to get the data.

```
GET session:12345
```

This command returns the JSON object:

```
{  
  "user_id": "42",  
  "username": "john_doe",  
  "email": "john.doe@example.com"  
}
```

3. Delete a Session: When the user logs out, you can delete the session.

```
DEL session:12345
```

Practical Application

Key-value stores like Redis are particularly useful in scenarios where:

- **Caching:** Store frequently accessed data to improve performance.
- **Session Management:** Maintain user session data in web applications.
- **Real-time Analytics:** Store counters, leaderboards, and other rapidly changing data.

Document oriented databases

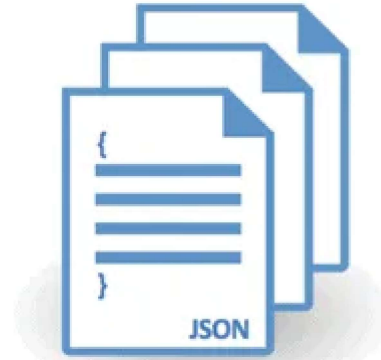
What is a Document Store?

A document store is a type of NoSQL database that stores data in JSON-like document formats. Each document contains semi-structured data that can vary in structure, allowing for flexibility and ease of handling complex data types. Document stores are ideal for applications requiring the management of nested data and variable schema.



Relational data model

Highly-structured table organization with rigidly-defined data formats and record structure.



Document data model

Collection of complex documents with arbitrary, nested data formats and varying "record" format.

Example Using MongoDB

MongoDB is one of the most popular document stores, known for its flexibility and scalability. Here's an example of how you might use MongoDB in a web application to store and retrieve user profile information.

Storing User Profiles

Let's say you have a web application where users can create profiles. You want to store information about each user, such as their name, email, and a list of their favorite books.

1. **Insert a User Profile:** When a user creates a profile, you insert a document into the database.

```
db.users.insertOne({
  "user_id": "42",
  "name": "John Doe",
  "email": "john.doe@example.com",
  "favorites": {
    "books": [
      {"title": "1984", "author": "George Orwell"},
      {"title": "To Kill a Mockingbird", "author": "Harper Lee"}
    ]
  }
})
```

```
    ],  
    "movies": [  
      {"title": "Inception", "director": "Christopher Nolan"},  
      {"title": "The Matrix", "director": "The Wachowskis"}  
    ]  
  }  
})
```

In this example:

- The collection users stores documents where each document represents a user profile.
- Each document can contain nested data, such as the favorites array, which includes books and movies.

2. Retrieve a User Profile: When you need to retrieve the user's profile information, you query the database by the user ID.

```
db.users.findOne({"user_id": "42"})
```

This command returns the user document:

```
{  
  "user_id": "42",  
  "name": "John Doe",  
  "email": "john.doe@example.com",  
  "favorites": {  
    "books": [  
      {"title": "1984", "author": "George Orwell"},  
      {"title": "To Kill a Mockingbird", "author": "Harper Lee"}  
    ],  
    "movies": [  
      {"title": "Inception", "director": "Christopher Nolan"},  
      {"title": "The Matrix", "director": "The Wachowskis"}  
    ]  
  }  
}
```

```
"movies": [  
  {"title": "Inception", "director": "Christopher Nolan"},  
  {"title": "The Matrix", "director": "The Wachowskis"}  
]  
}  
}
```

3. Update a User Profile: To update the user's email address, you can use the update command.

```
db.users.updateOne(  
  {"user_id": "42"},  
  { $set: {"email": "john.doe_new@example.com"} }  
)
```

4. Delete a User Profile: When the user deletes their profile, you remove the document from the collection.

```
db.users.deleteOne({"user_id": "42"})
```

Practical Application

Document stores like MongoDB are particularly useful in scenarios where:

- 1. Content Management:** Manage and store varying content types such as articles, blogs, and user-generated content.
- 2. E-commerce:** Handle product catalogs, customer data, and order histories with complex nested data.

3. Mobile and Web Apps: Store user preferences, profiles, and app configurations that require flexible schema design.

Column-oriented databases

What is a Column-Family Store

A column-family store is a type of NoSQL database that stores data in columns rather than rows, which is the traditional format used by relational databases. Data is organized into column families, which are groups of related columns. This model is highly efficient for read-heavy workloads and can handle large-scale data storage, making it suitable for applications that require high performance and scalability.

Example Using Apache Cassandra

Apache Cassandra is one of the most widely used column-family stores, known for its distributed architecture and ability to handle large amounts of data across many commodity servers without a single point of failure. Here's an example of how you might use Cassandra in a web application to store and retrieve user activity logs.

Storing User Activity Logs

Let's say you have a web application where you want to keep track of user activities, such as page views and clicks. Each user activity is logged with a timestamp and stored in a column-family store.

1. Create a Keyspace and Table: First, you define a keyspace (similar to a database) and a table to store user activities.

```
CREATE KEYSPACE user_activity
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 };

USE user_activity;

CREATE TABLE activities (
  user_id UUID,
  activity_time TIMESTAMP,
  activity_type TEXT,
  page_url TEXT,
  PRIMARY KEY (user_id, activity_time)
);
```

In this example:

- user_activity is the keyspace.
- activities is the table that stores user activities.
- The PRIMARY KEY is a composite key consisting of user_id and activity_time, ensuring that each activity is uniquely identified by both the user and the time it occurred.

2. Insert User Activities: When a user performs an action, you insert a record into the activities table.

```
INSERT INTO activities (user_id, activity_time, activity_type, page_url)
VALUES (uuid(), toTimestamp(now()), 'page_view', 'https://example.com/home');
```

3. Retrieve User Activities: To get the activity log of a specific user, you query the table by the user ID.

```
SELECT * FROM activities WHERE user_id = <specific-uuid>;
```

This command returns all activities performed by the user with the given user_id.

4. Delete User Activities: To delete all activity logs for a specific user, you can execute a delete command.

```
DELETE FROM activities WHERE user_id = <specific-uuid>;
```

Practical Application

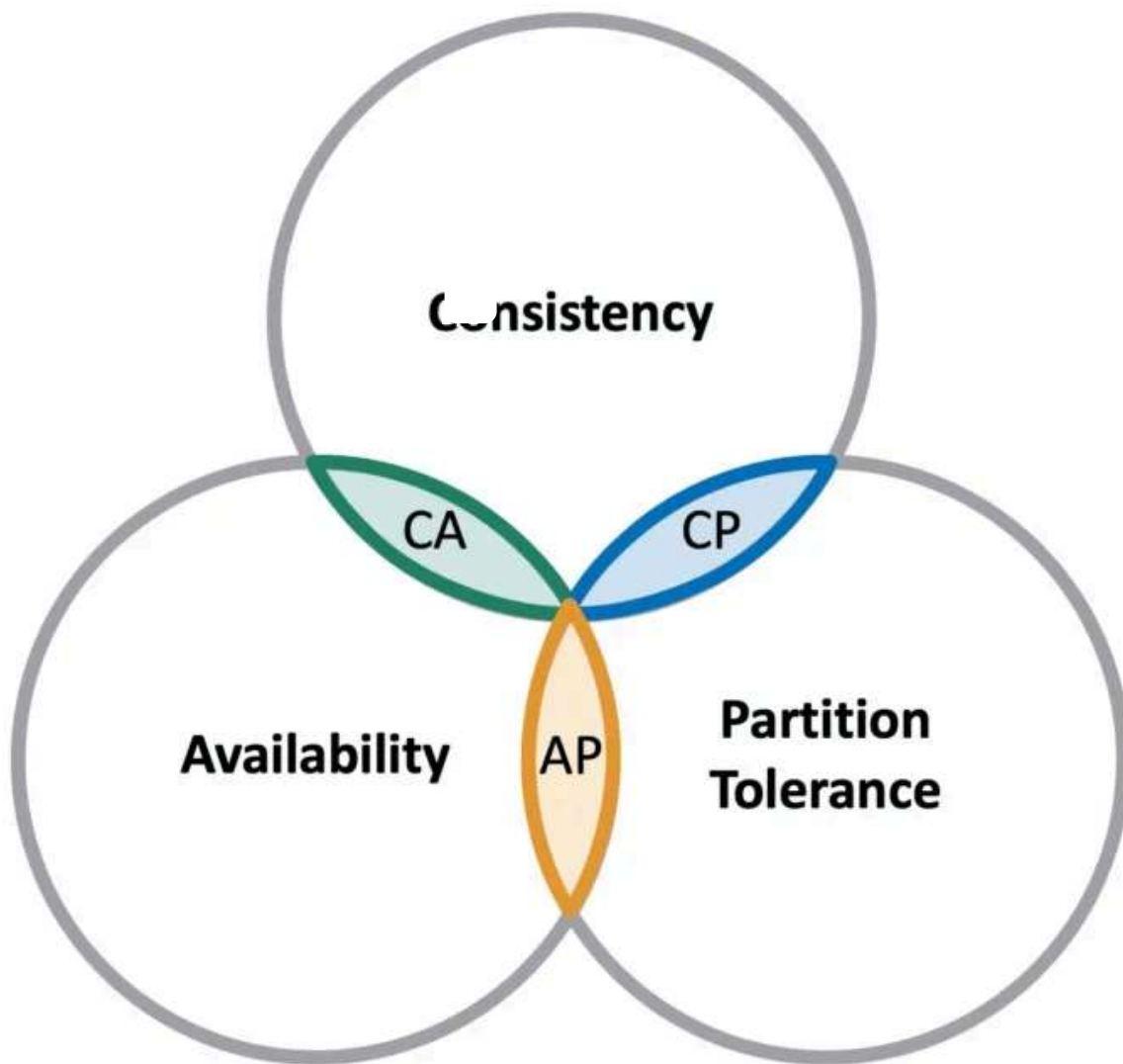
Column-family stores like Cassandra are particularly useful in scenarios where:

- Time-Series Data: Store and query large volumes of time-stamped data, such as logs, sensor data, and financial transactions.
- Recommendation Engines: Efficiently store and retrieve user interactions and preferences to generate personalized recommendations.
- Real-Time Analytics: Perform real-time data analysis on streaming data, such as monitoring and alerting systems.

Other types of Nosql databases:

1. **Graph Databases:** Designed for applications that involve complex relationships between data entities, graph databases like Neo4j and OrientDB represent data as nodes and edges. They are particularly useful for social networks, recommendation engines, and fraud detection systems.
2. **Time-Series Databases:** Specialized for managing time-stamped data, these databases are ideal for applications that track changes over time, such as IoT data, financial markets, and log management. Examples include InfluxDB and TimescaleDB.
3. **Search Engines:** Databases optimized for full-text search and indexing, such as Elasticsearch, are crucial for applications requiring fast and accurate search capabilities.

The CAP Theorem and NoSQL



The CAP theorem, introduced by Eric Brewer, is fundamental to understanding the trade-offs inherent in distributed database systems. It states that a distributed system can guarantee only two out of the following three properties simultaneously:

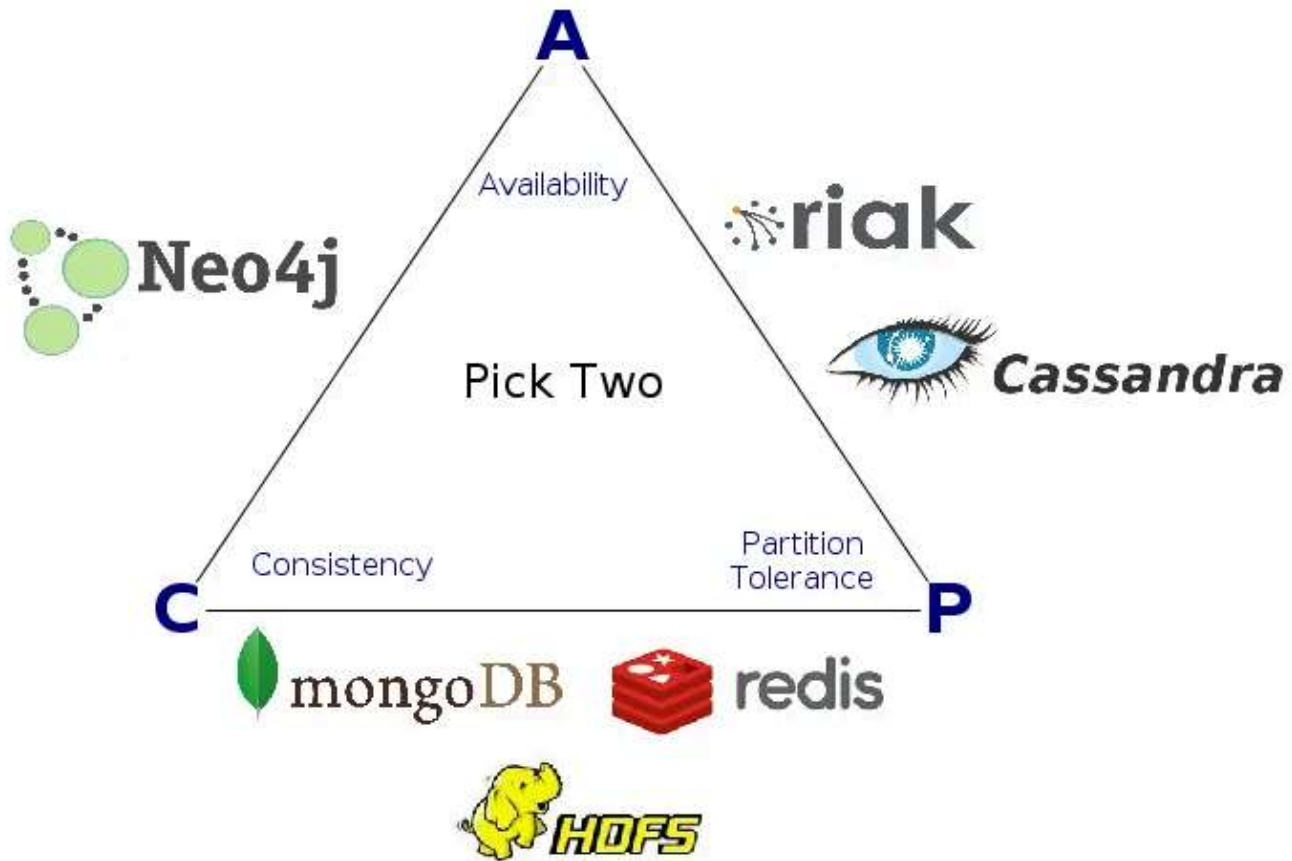
1. **Consistency:** Every read receives the most recent write or an error.
2. **Availability:** Every request receives a response, without guarantee that it contains the most recent write.

3. Partition Tolerance: The system continues to operate despite network partitions.

NoSQL databases are designed to prioritize different combinations of these properties, depending on the specific use case. For instance:

- CA (Consistency and Availability): Databases like relational databases (MySQL) focus on consistency and availability but may sacrifice partition tolerance.
- CP (Consistency and Partition Tolerance): Databases such as MongoDB and HBase ensure consistency and can handle network partitions but may compromise on availability.

AP (Availability and Partition Tolerance): Systems like Cassandra and DynamoDB prioritize availability and partition tolerance, potentially at the cost of consistency.



References:

<https://www.decipherzone.com/blog-detail/nosql-databases>

<https://www.mongodb.com/resources/basics/databases/nosql-explained>

<https://www.dataversity.net/a-brief-history-of-non-relational-databases/>

<https://en.wikipedia.org/wiki/NoSQL>

<https://www.simplilearn.com/rise-of-nosql-and-why-it-should-matter-to-you-article>

<https://hazelcast.com/glossary/cap-theorem/>

<https://dataconomy.com/2014/07/01/sql-vs-nosql-need-know/>

<https://www.decipherzone.com/blog-detail/nosql-databases>

Big Data

NoSQL

Data Scienc

Nosql Database

Data Engineering



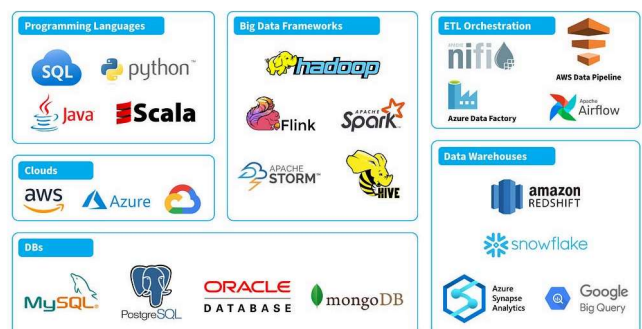
Written by Datainsights

Follow

31 Followers

The mission of Data Insights is to democratize access to data literacy and empower individuals and communities through education and resource sharing.

More from Datainsights



 Datainsights

Data Quality in Data engineering

Introduction

May 6

 1







 Datainsights

Data engineering life cycle

Introduction:

Apr 19

 21





 Datainsights

Data Engineering Tools and Technologies

Introduction:

Apr 22

 15







 Datainsights

Data Governance

Introduction

May 6

 1





See all from Datainsights

Recommended from Medium

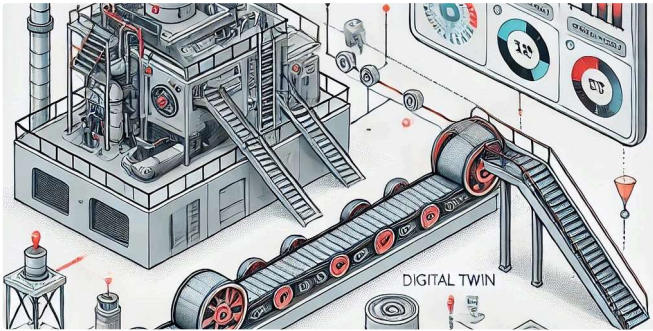


 Gayathry Dasika in Cloud Solutions Architect

Databases vs. Data Warehouses vs. Data Lakes vs Data Marts:...

We all heard these terms and emphatic rallies for datalakes, warehouses, marts, and bricks...

✦ Jun 11 🖱 9 📌 ...



 RoshanGavandi

Understanding Digital Twins: A Practical Guide with Real-World...

As software architects, we're constantly looking for ways to bridge the gap between...

Sep 15 🖱 51 📌 ...

Lists



Predictive Modeling w/ Python

20 stories · 1548 saves



Coding & Development

11 stories · 817 saves



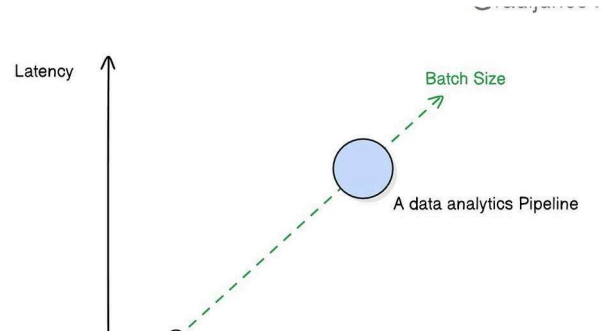
Practical Guides to Machine Learning


10 stories · 1877 saves



ChatGPT prompts

48 stories · 2012 saves



 Archana Goyal



 Hafiq Iqmal

Cheatsheet For Designing data pipelines

My articles are open to everyone; non-member readers can read the full article by...

★

Sep 15

👤

15

💬

1

🔖

+

...



Sai Parvathaneni in Towards Dev

Building a Log Analysis Data Pipeline Using Kafka,...

When it comes to analyzing logs, having a real-time, centralized, and automated soluti...

★

Sep 11

👤

69

🔖

+

...

System Design: Lawn Care Booking System

Revolutionizing Lawn Care: Designing a Lawn Care Booking System

★

Aug 3

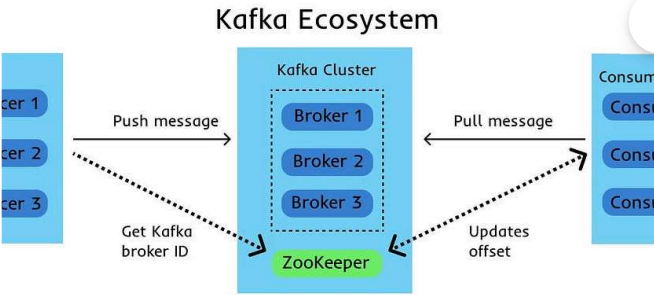
👤

11

🔖

+

...



Aditya Mangal

How Kafka Works with Same and Different Consumer Groups

Introduction to Apache Kafka

Sep 13

🔖

+

...

See more recommendations