Wk1 - Introducing NoSQL

NoSQL means Not only SQL. The term refers to a **class of databases** that are non-relational in architecture. They have their roots in the open source community. NoSQL databases have become more popular due to the demands of Big Data. In this module, you will learn about the characteristics of NoSQL and the **four main categories** of NoSQL databases available. You will explore the differences between the **ACID** and **BASE** consistency models, the advantages and challenges of distributed systems, and the CAP theorem and its characteristics. You will learn how to decide when to use RDBMS and when to use NoSQL.

Learning Objectives

- Define the term NoSQL and the technology it references.
- Explain the characteristics of NoSQL databases.
- List the four main categories of NoSQL databases available.
- Describe the Key-Value NoSQL database category, including its architecture.
- Explain the document-based NoSQL database category, including its architecture.
- Describe the column-based NoSQL database category, including its architecture.
- Explain the Graph NoSQL database category and contrast it with the other three NoSQL database categories.
- Define Database-as-a-Service (DBaaS) and distinguish between the available database hosting options.
- Discuss the differences between ACID and BASE operations.
- Summarize the advantages and challenges of distributed systems.
- Define the CAP theorem and describe its characteristics.
- Identify the best cases in which to use RDBMS and NoSQL.

Course Introduction

- Course/Section(Module) Summaries
- Refers to the entire Course Module Summary learning path as well as the technologies and objectives/key takeaways from each section for reference

… now back to the first module / Section

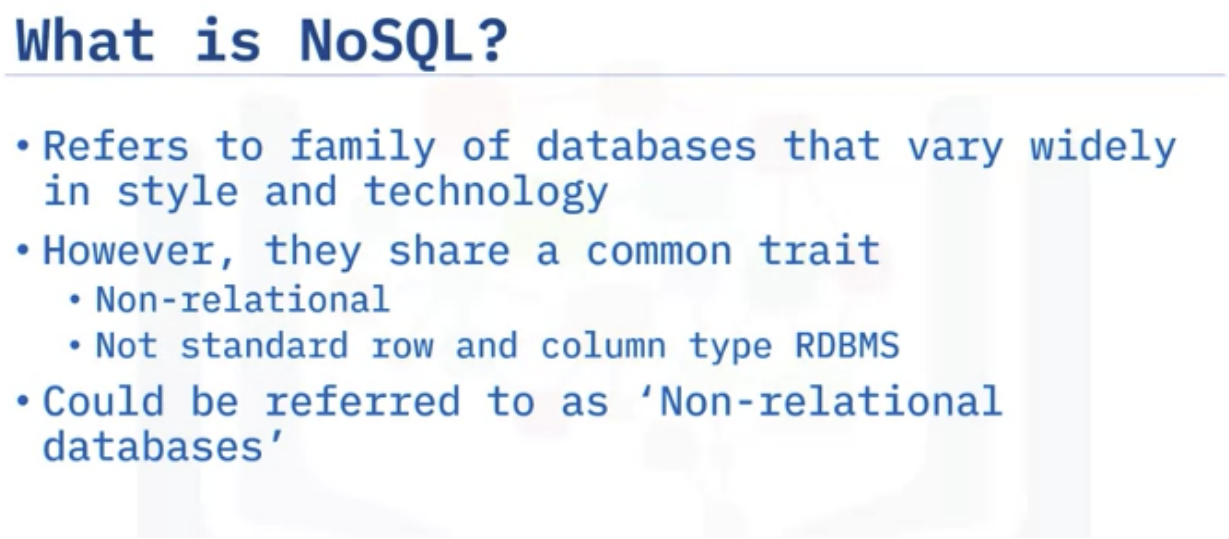
Overview of NoSQL

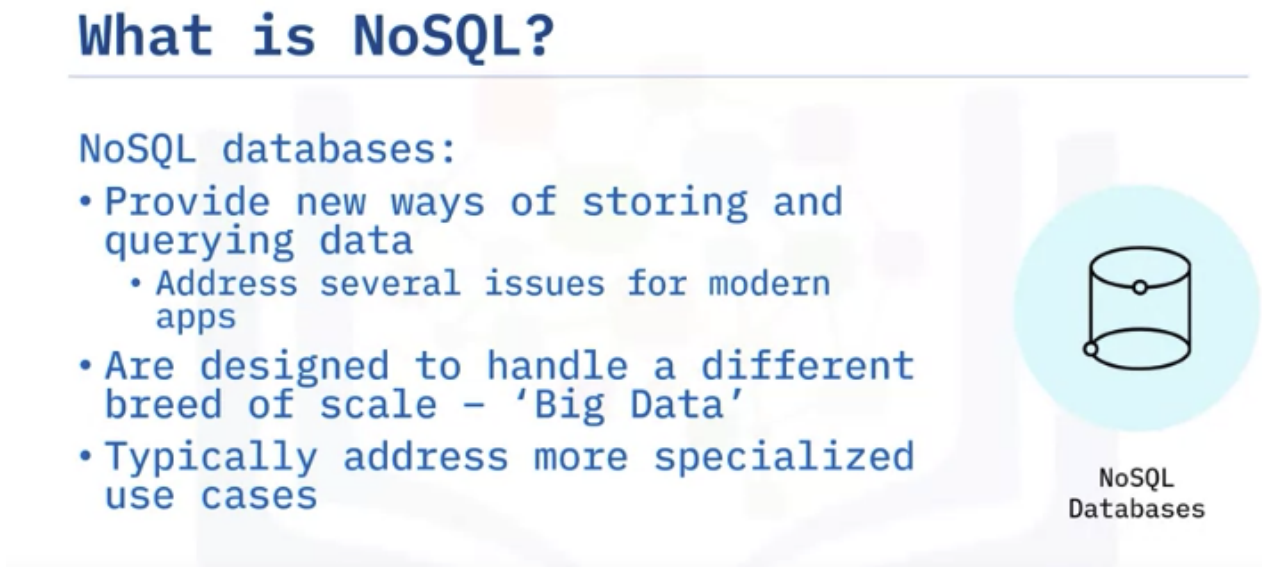
- First, let's talk about the name NoSQL.
- The name was introduced at an event to discuss all of the new open-source distributed databases that were coming onto the scene, and the name, NoSQL, has stuck ever since.

Not NoSQL … actually Not Only SQL

- Contrary to what it sounds like, NoSQL actually stands for 'Not Only SQL' and not 'NO SQL'.
    - The name NoSQL really describes what it is not.

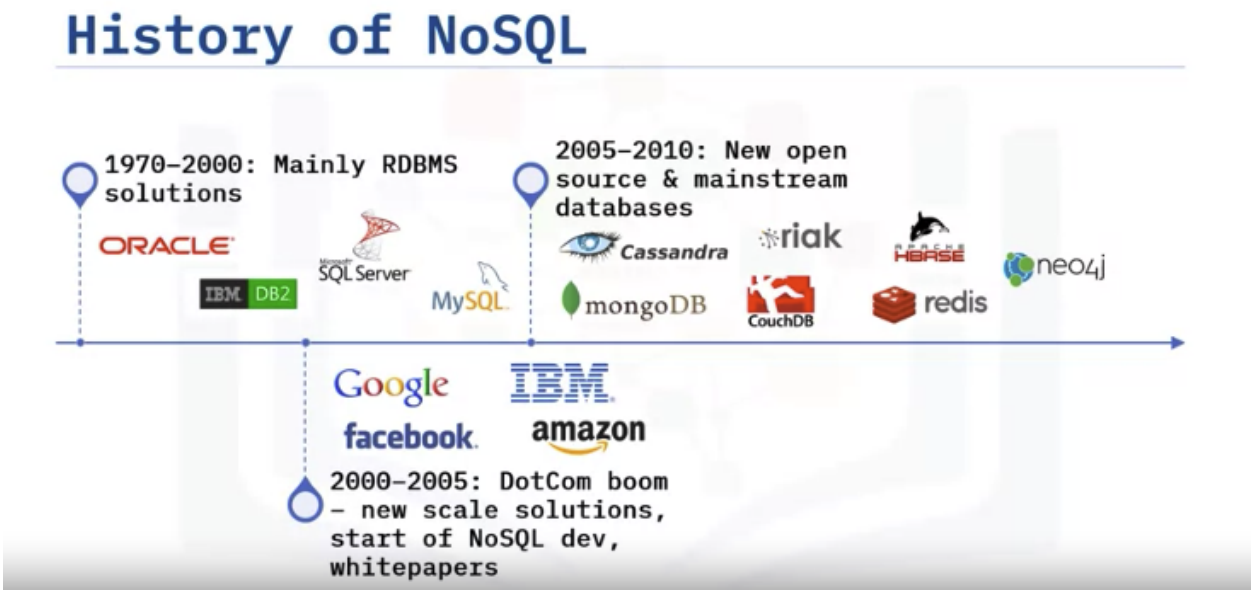
What is NoSQL





- NoSQL databases provide new ways of storing and querying data that address a number of issues for modern applications.
    - Most importantly, the majority of NoSQL databases are geared to handle a different breed of scale problems that have arisen associated with the “big data” movement.
    - By scale, we are referring to both the size of the data, but also the concurrent users acting on that data.
- NoSQL databases are typically also more specialized in their use cases and can be much simpler to develop application functionality for, than relational databases.

History & Rise of NoSQL



- When internet applications and companies started exploding during the dotcom boom in the late 90's/early 2000's, applications went from predominately serving thousands of internal employees at companies, to needing to serve millions of users on the public internet.
    - For these applications, availability and performance were paramount and these new scale problems led to a drive to create new scalable technologies to support them.
- At this time, several large tech companies such as IBM, Google, and Facebook, developed a lot of great technology, and in turn released whitepapers, and/or open-sourced the technology, to the community.
    - Specific examples include Google's MapReduce whitepaper which described how to process large data sets on distributed systems and Amazon's Dynamo whitepaper which detailed a way to evenly distribute data and workload within a cluster in a quorum style architecture.
- In the late 2000's, several new databases emerged on the scene, a large number of them from open-source communities.
    - Databases like **Apache CouchDB**, **Cassandra**, and **Hbase**, as well as **Mongo**, **Redis**, **Riak**, and **Neo4j** became more prevalently used in applications, particularly in ones that required larger scale than a relational database could handle.
- In the last 10 years or so, several NoSQL databases have leveraged a fully managed service model, otherwise called database-as-a-service (or DBaaS) to offload the administration and maintenance from the end user and allow developers to focus on building applications with these modern databases.
    - Examples include **IBM Cloudant** and **Amazon DynamoDB**.

Summary

In this video, you learned that:

- The name 'NoSQL' stands for Not only SQL
- NoSQL refers to a class of databases that are non-relational in architecture
- Implementations of NoSQL databases differ technically, but share common traits
- Since 2000 NoSQL databases have become more popular in the marketplace, due to scale demands of Big Data

Characteristics of NoSQL Databases

- What types of NoSQL databases are available? And what is common to them?

Categories

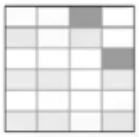
NoSQL Database Categories

General consensus is...

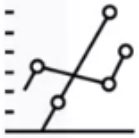
...NoSQL databases fit into four categories

Key-Value

Document



Column



Graph

NoSQL Database Characteristics

But what ties NoSQL databases together?

- Majority have their roots in the open source community
- Many have been used and leveraged in an open source manner
- Open source community support is fundamental to their industry growth

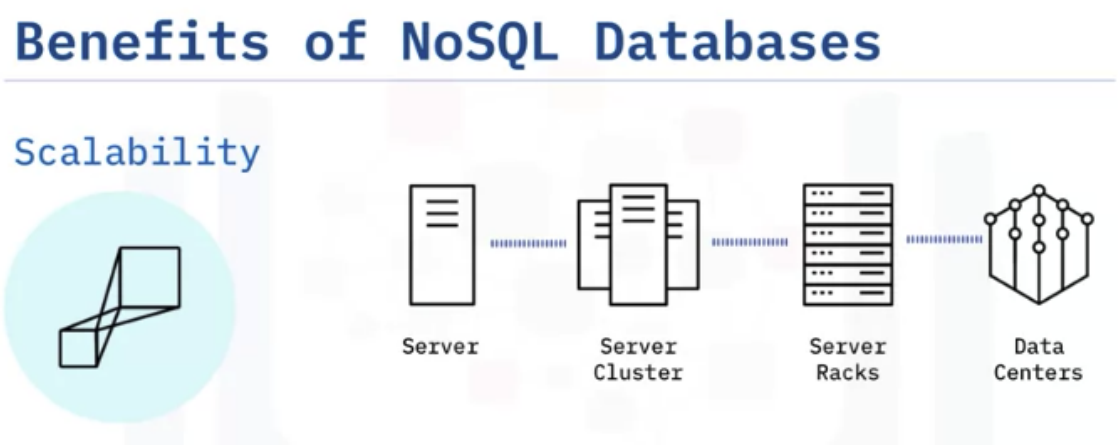
Characteristics

NoSQL Database Characteristics

- They all differ technically speaking, but have a few commonalities
- Most NoSQL databases:
  - Are built to scale horizontally
  - Share data more easily than RDBMS
  - Use a global unique key to simplify data sharding
  - Are more use case specific than RDBMS
  - Are more developer friendly than RDBMS
  - Allow more agile development via flexible schemas

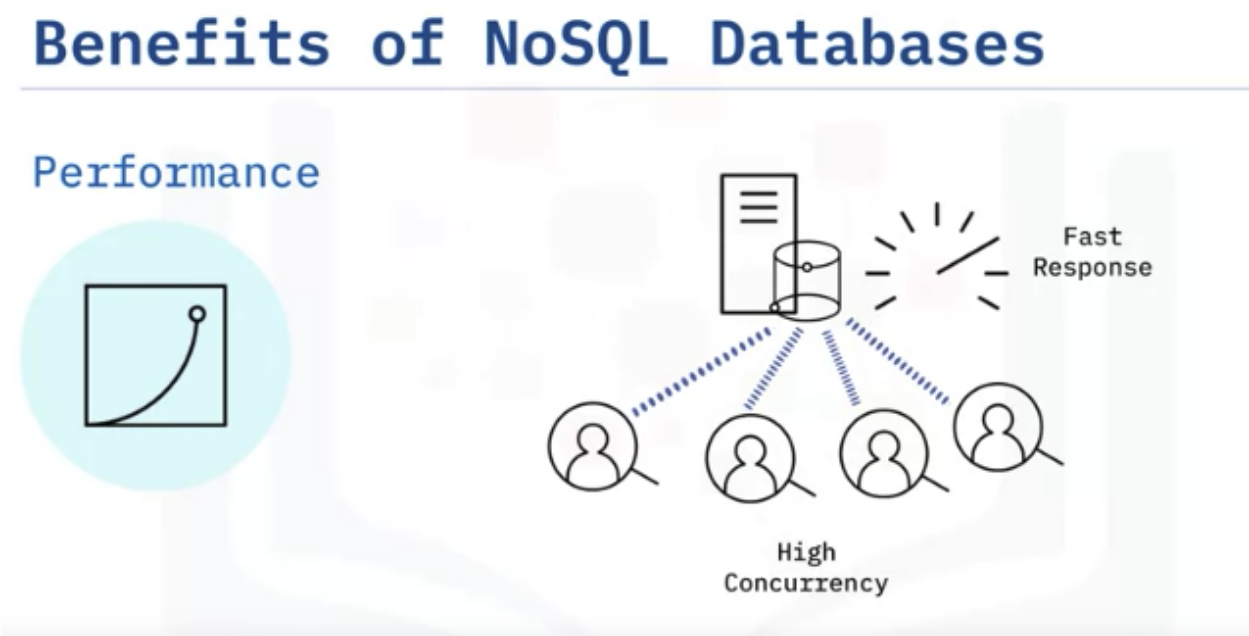
Benefits of NoSQL Databases

Scalability



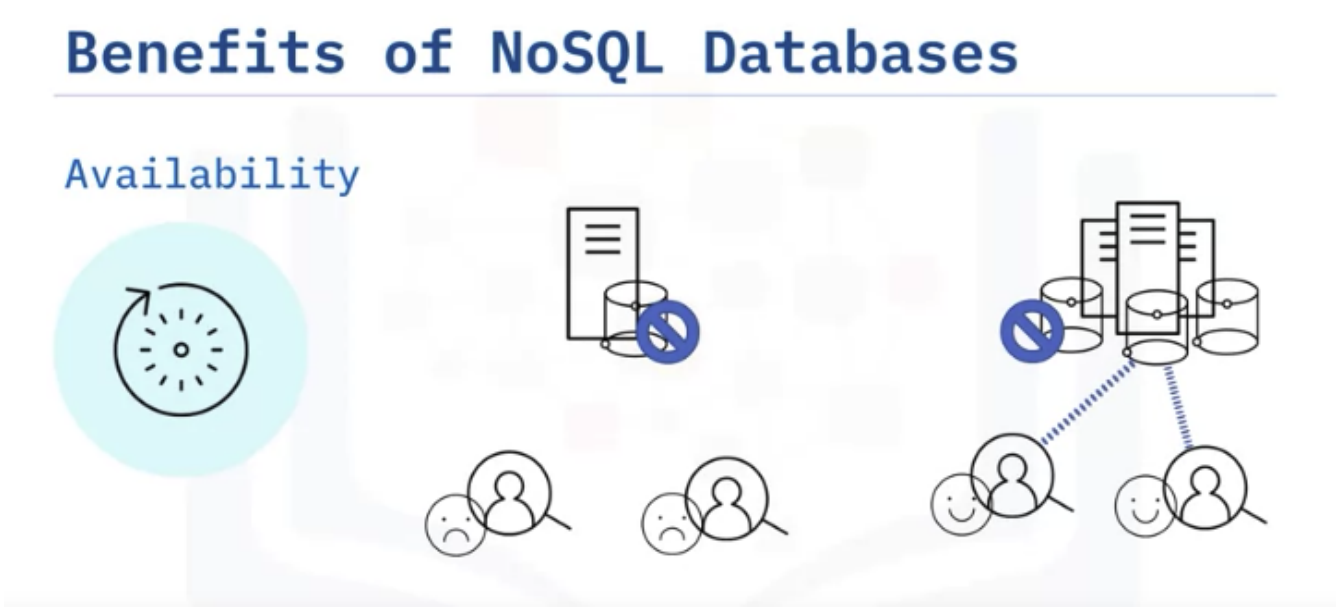
- First, the most common reason to employ a NoSQL database is for **scalability**, particularly the ability to horizontally scale across clusters of servers, racks, and possibly even data centers.
    - The **elasticity** of scaling both up and down to meet the varying demands of applications is key.
    - NoSQL databases are well suited to meet the large data size and huge number of concurrent users that “Big Data” applications exhibit.

Performance



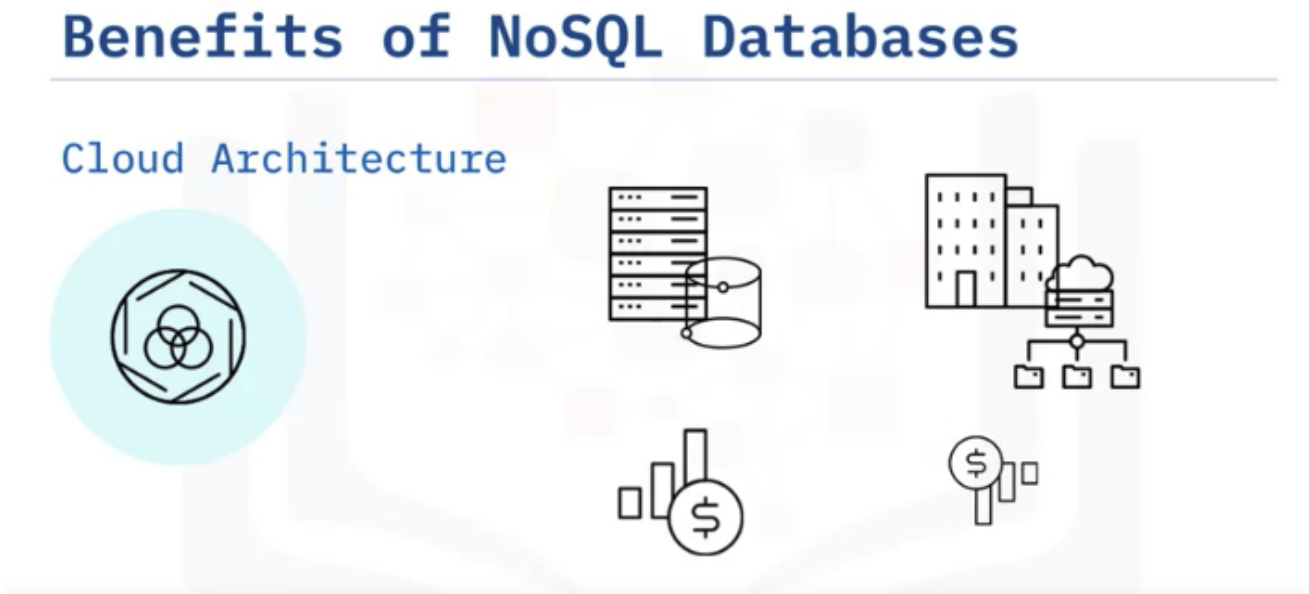
- The second point of performance goes hand-in-hand with scalability.
- The need to deliver fast response times even with large data sets and high concurrency is a must for modern applications, and the ability of NoSQL databases to leverage the resources of large clusters of servers makes them ideal for fast performance in these circumstances.

Availability



- High Availability is an obvious requirement for a database, and having a database run on a cluster of servers with multiple copies of the data makes for a more resilient solution than a single server solution.
    - Historically, large databases have run on expensive machines or mainframes.

Cloud Architecture



- Modern enterprises are employing cloud architectures to support their applications, and the distributed data nature of NoSQL databases means that they can be deployed and operated on clusters of servers in cloud architectures, thereby massively reducing cost.

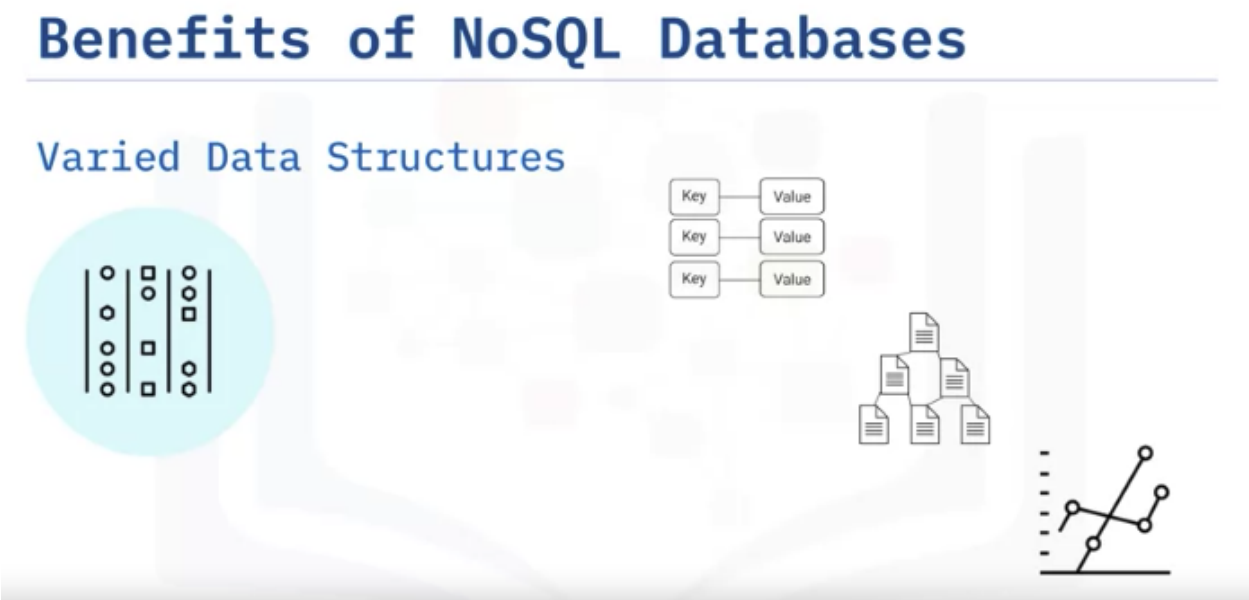
Cost

Cost is important for any technology venture, and it is common to hear of NoSQL adopters cutting significant costs vs. their existing databases... and still be able to get the same or better performance and functionality.

Flexible Schema

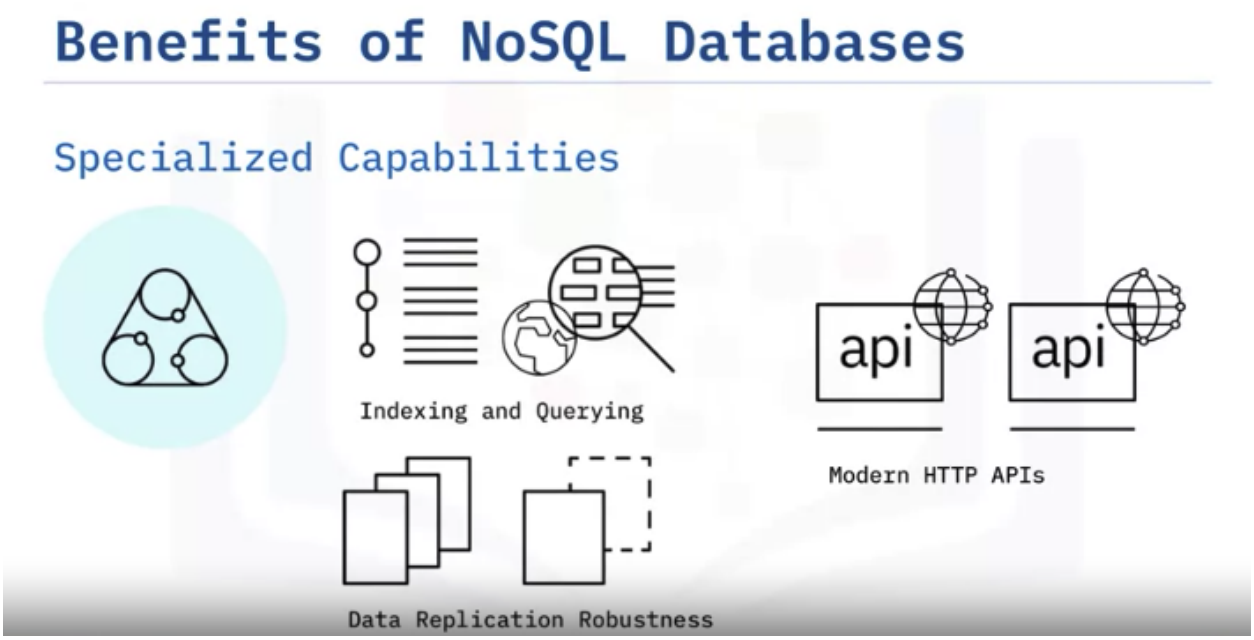
- Flexible schema and intuitive data structures are key features that developers love when wanting to build applications efficiently.
- Most NoSQL databases allow for having flexible schemas, which means that one can build new features into applications quickly and without any database locking or downtime.

Varied Data Structures



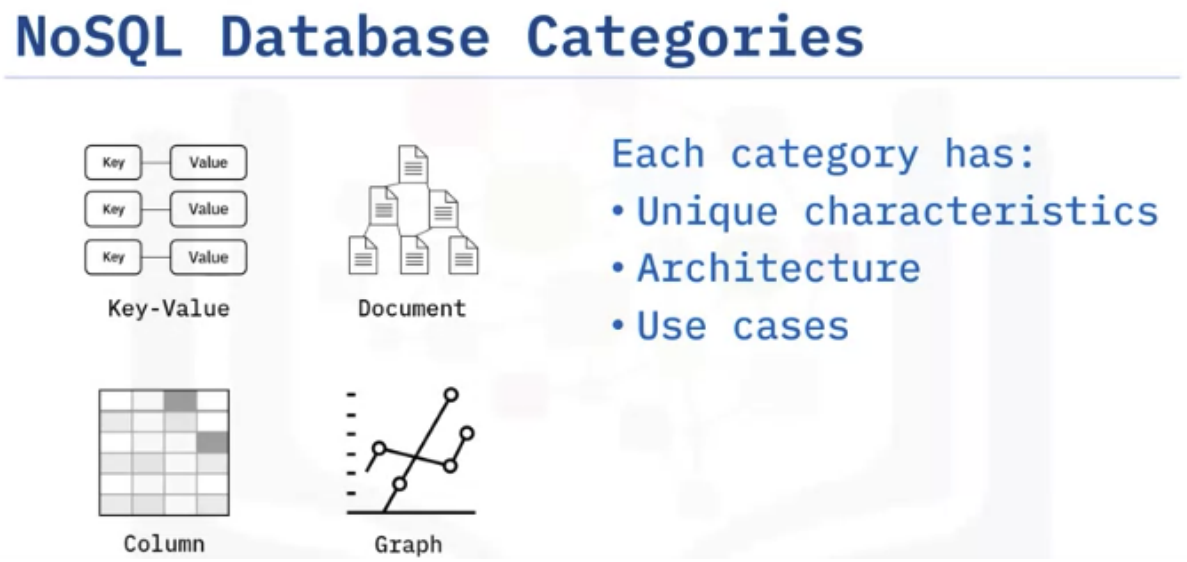
- NoSQL databases also have varied data structures which often are more eloquent for solving development needs than the rows and columns of relational datastores.
- Examples include key- value stores for quick lookup, document stores for storing de-normalized intuitive information, and graph databases for associative data sets.

Specialized Capabilities



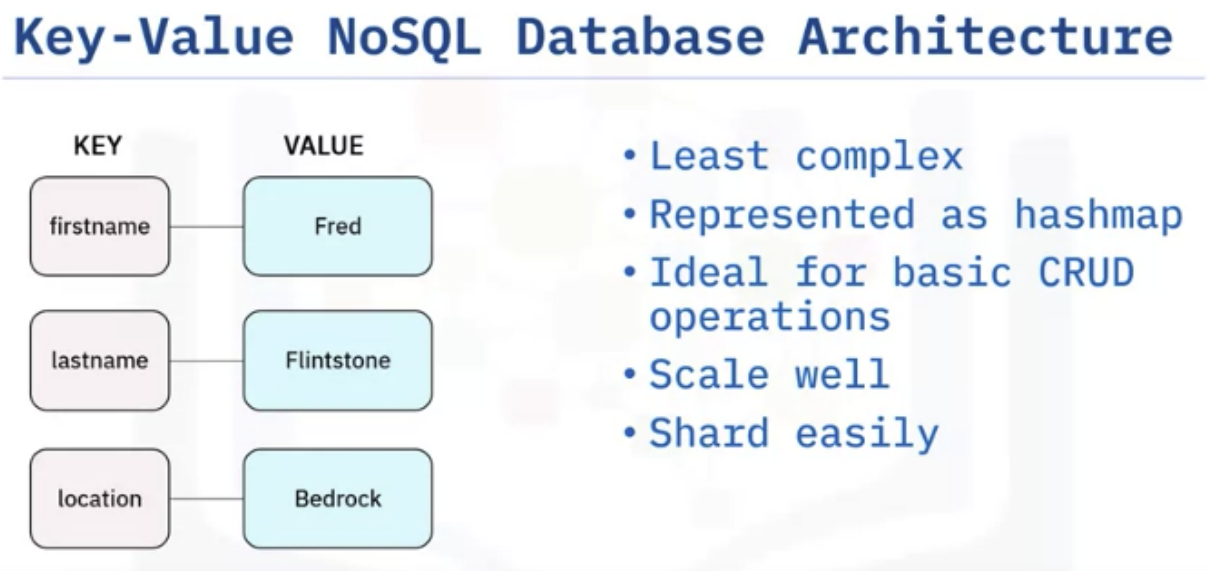
- There are also various specialized capabilities that certain NoSQL providers offer that attract end users.
    - Examples include specific indexing and querying capabilities such as geospatial search, data replication robustness, and modern HTTP API's.

NoSQL Database Categories

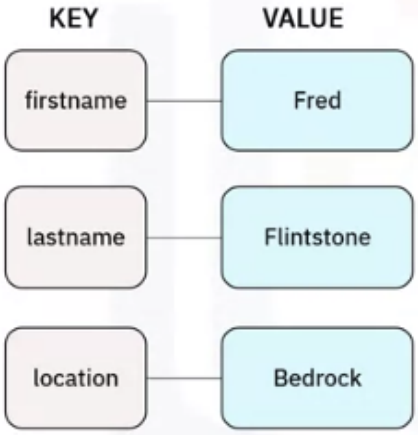


Key-Value

Architecture



Key-Value NoSQL Database Architecture



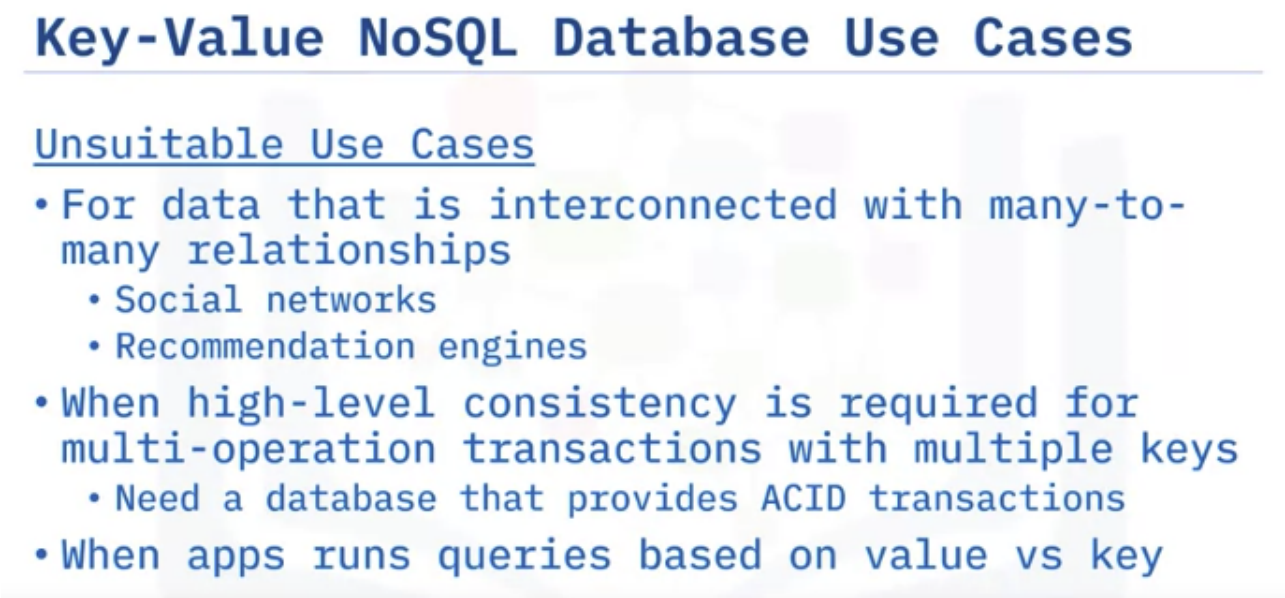
- Not intended for complex queries
- Atomic for single key operations only
- Value blobs are opaque to database
  - Less flexible data indexing and querying

UseCases (Suitable and Not)

Key-Value NoSQL Database Use Cases

Suitable Use Cases

- For quick basic CRUD operations on non-interconnected data
  - E.g. Storing and retrieving session information for web applications
- Storing in-app user profiles and preferences
- Shopping cart data for online stores



- When based on **value** (Consider using 'Document' category)

Providers of Key/Vaue DB's

- Some examples of the more popular implementations of key-value NoSQL databases are: Amazon DynamoDB, Oracle NoSQL Database, Redis, Aerospike, Riak KV, MemcacheDB, and Project Voldemort

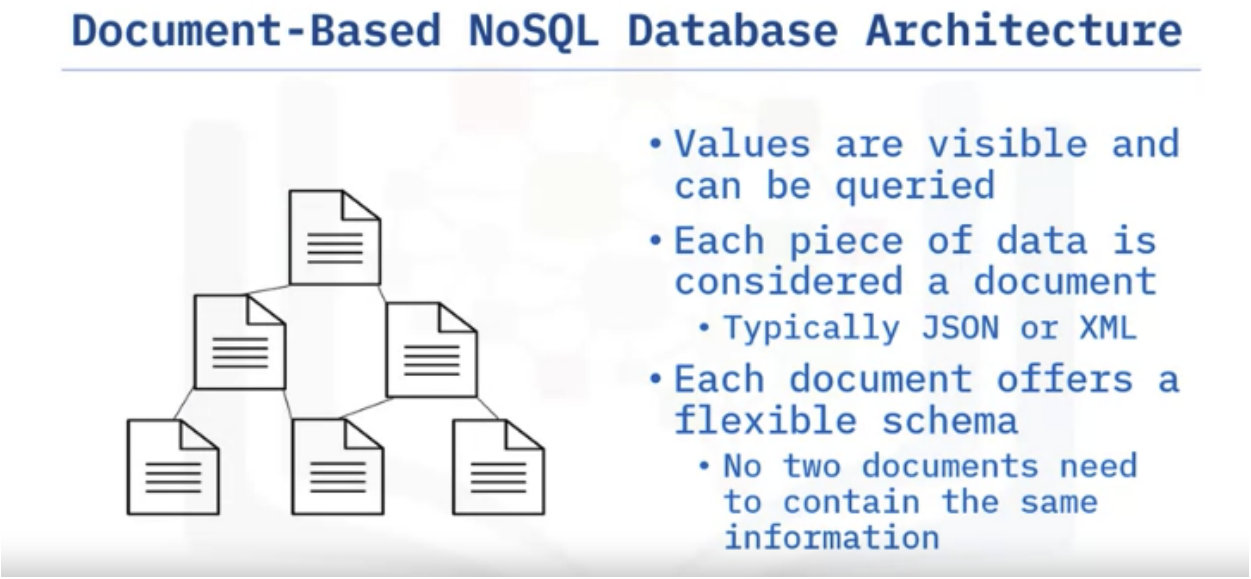
Summary

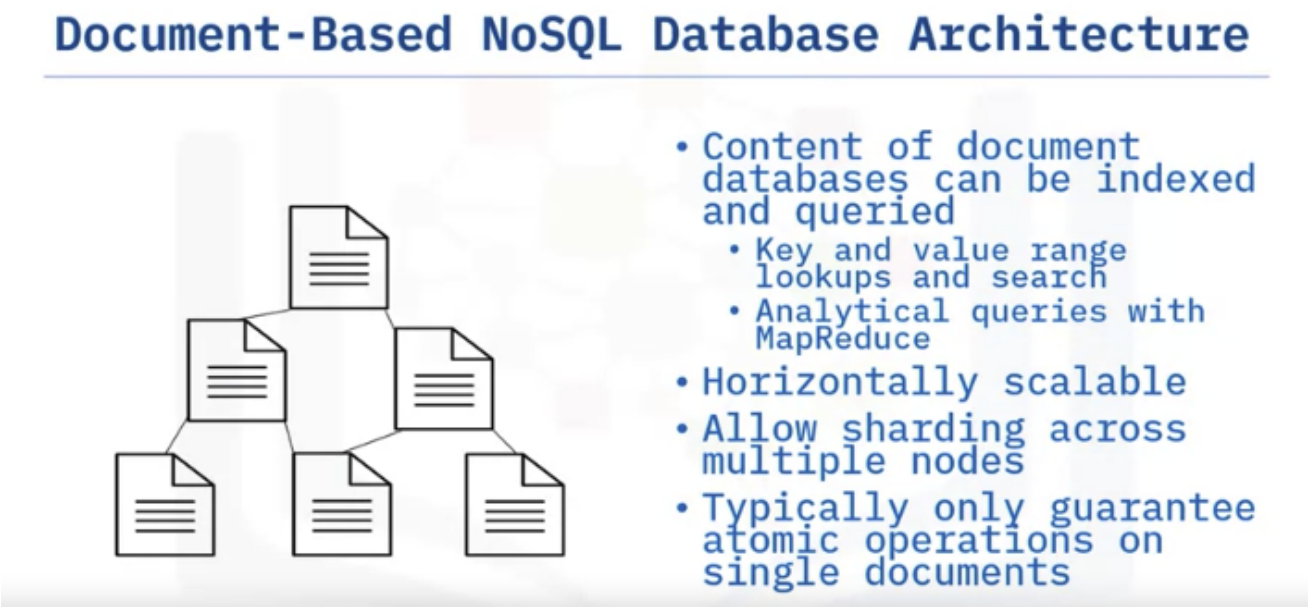
In this video, you learned that:

- The four main categories of NoSQL database are Key-Value, Document, Column, and Graph
- The Key-Value database architecture is the least complex; data is stored with a key and corresponding value blob and is represented by a hashmap
- The primary use cases for Key-Value databases are for quick CRUD operations; for example, storing and retrieving session information, storing in-app user profiles, and storing shopping cart data

Document

Architecture

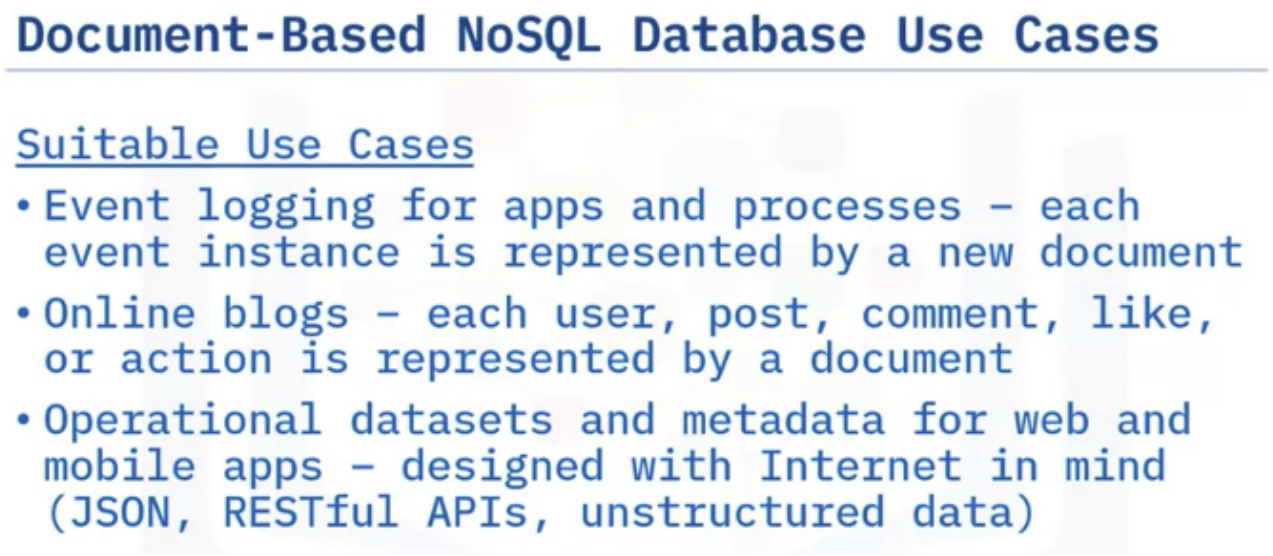


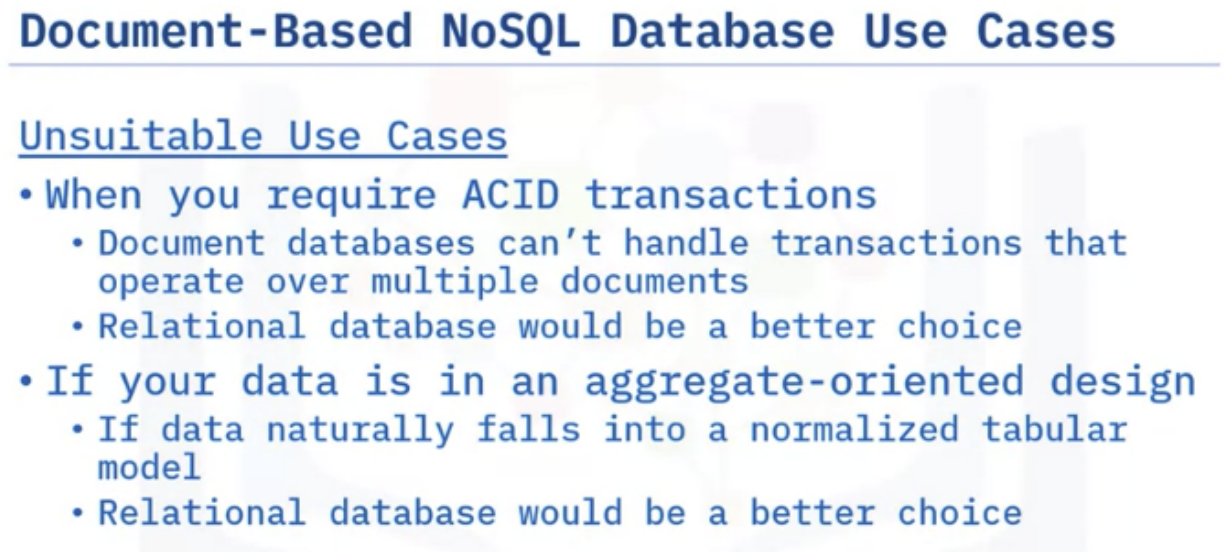


- Document databases typically offer the ability to index and query the contents of the documents, offering key and value range lookups and search ability, or perhaps analytical queries via paradigms like MapReduce.

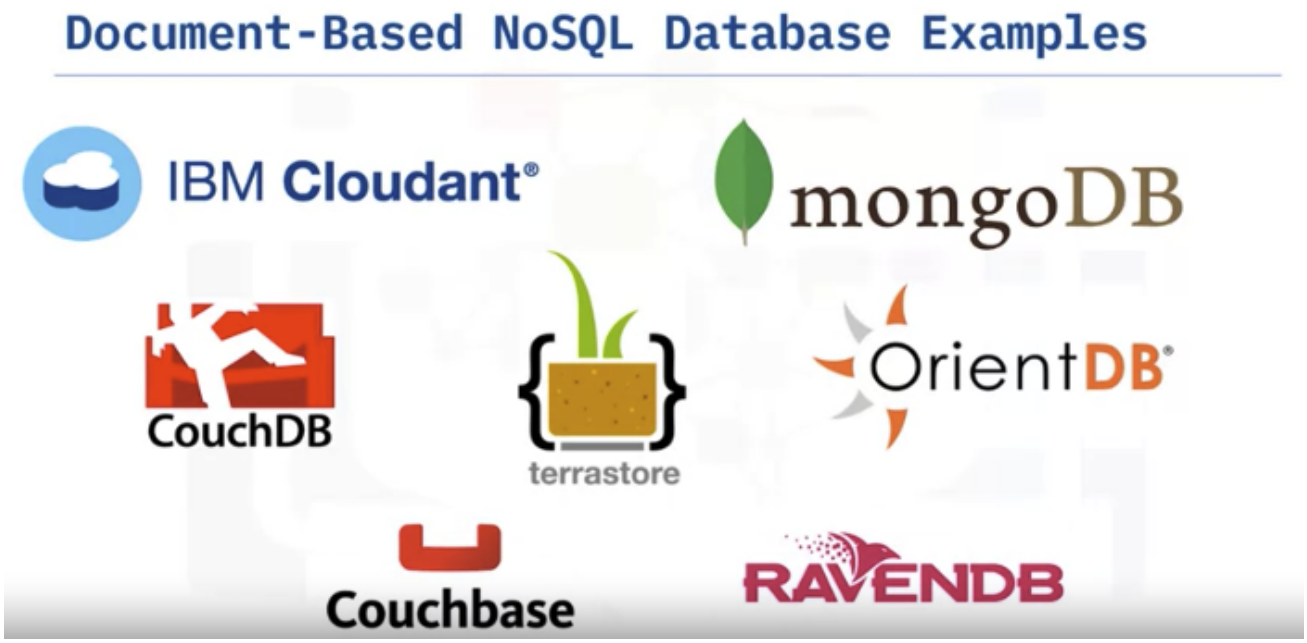
- Document databases are **horizontally scalable** and allow for sharding across multiple nodes, typically sharded by some unique key in the document.
- Document stores also typically only guarantee atomic transactions on single document operations.

UseCases (Suitable and Not)

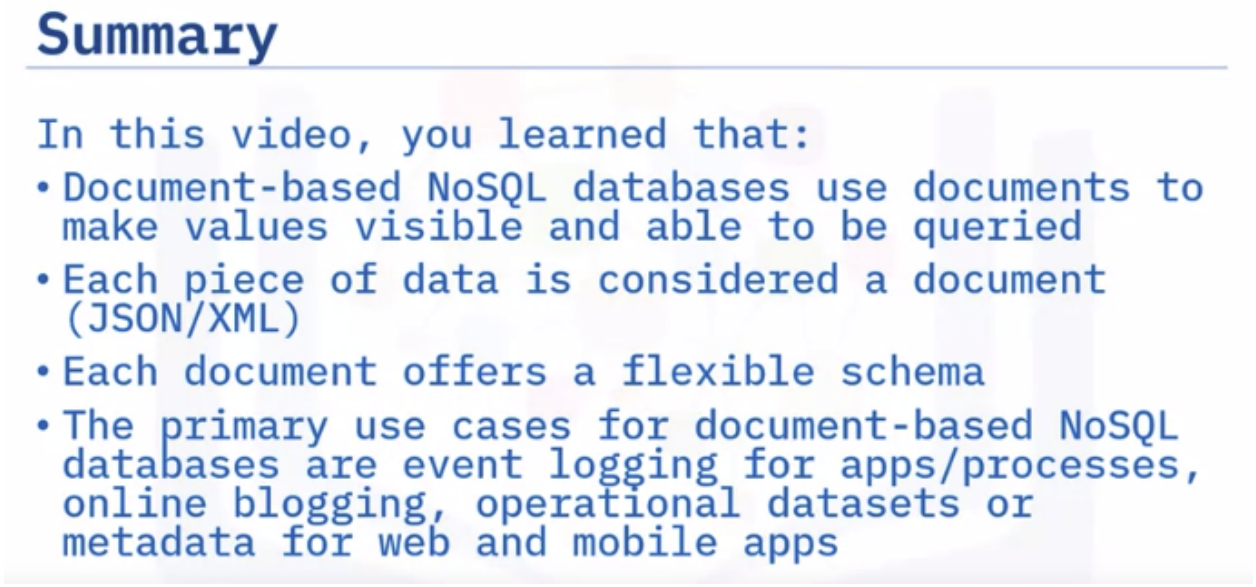




Document Based NoSQL Examples



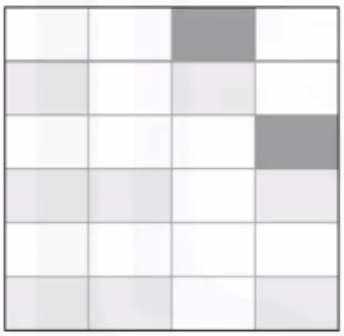
Summary



Column

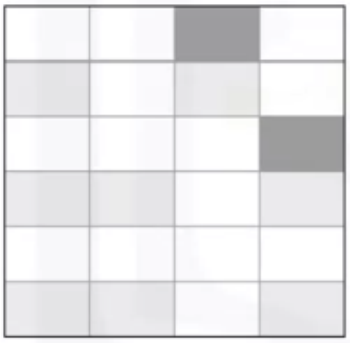
Architecture

Column-Based NoSQL Database Architecture



- Spawned from Google's 'Bigtable'
- a.k.a. Bigtable clones or Columnar or Wide-Column databases
- Store data in columns or groups of columns

Column-Based NoSQL Database Architecture



- Column 'families' are several rows, with unique keys, belonging to one or more columns
  - Grouped in families as often accessed together
- Rows in a column family are not required to share the same columns
  - Can share all, a subset, or none
  - Columns can be added to any number of rows, or not

UseCases (Suitable and Not)

Column-Based NoSQL Database Use Cases

Suitable Use Cases

- Great for large amounts of sparse data
- Column databases can handle being deployed across clusters of nodes
- Column databases can be used for event logging and blogs
- Counters are a unique use case for column databases
- Columns can have a TTL parameter, making them useful for data with an expiration value

Column-Based NoSQL Database Use Cases

Unsuitable Use Cases

- For traditional ACID transactions
  - Reads and writes are only atomic at the row level
- In early development, query patterns may change and require numerous changes to column-based databases
  - Can be costly and can slow down the production timeline

Column-Based NoSQL Database Examples

Summary

Summary

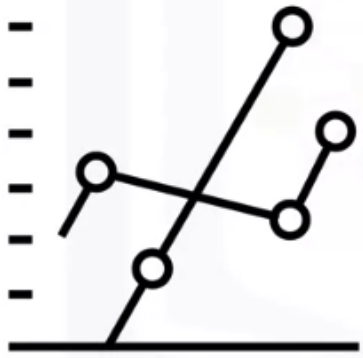
In this video, you learned that:

- Column-based databases were spawned from the architecture of Google's Bigtable storage system
- Column-based databases store data in columns or groups of columns
- Column 'families' are several rows, with unique keys, belonging to one or more columns
- The primary use cases for Column-based databases are event logging, blogs, counters, and data with expiration values

Graph

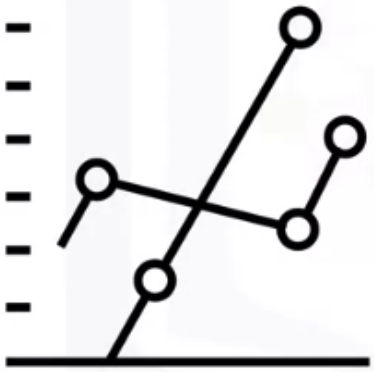
Architecture

Graph NoSQL Database Architecture



- Graph databases store information in entities (or nodes), and relationships (or edges
- Graph databases are impressive when your data set resembles a graph-like data structure

Graph NoSQL Database Architecture



- Graph databases do not shard well
  - Traversing a graph with nodes split across multiple servers can become difficult and hurt performance
- Graph databases are ACID transaction compliant
  - Unlike other NoSQL databases discussed

UseCases (Suitable and Not)

Graph NoSQL Database Use Cases

Suitable Use Cases

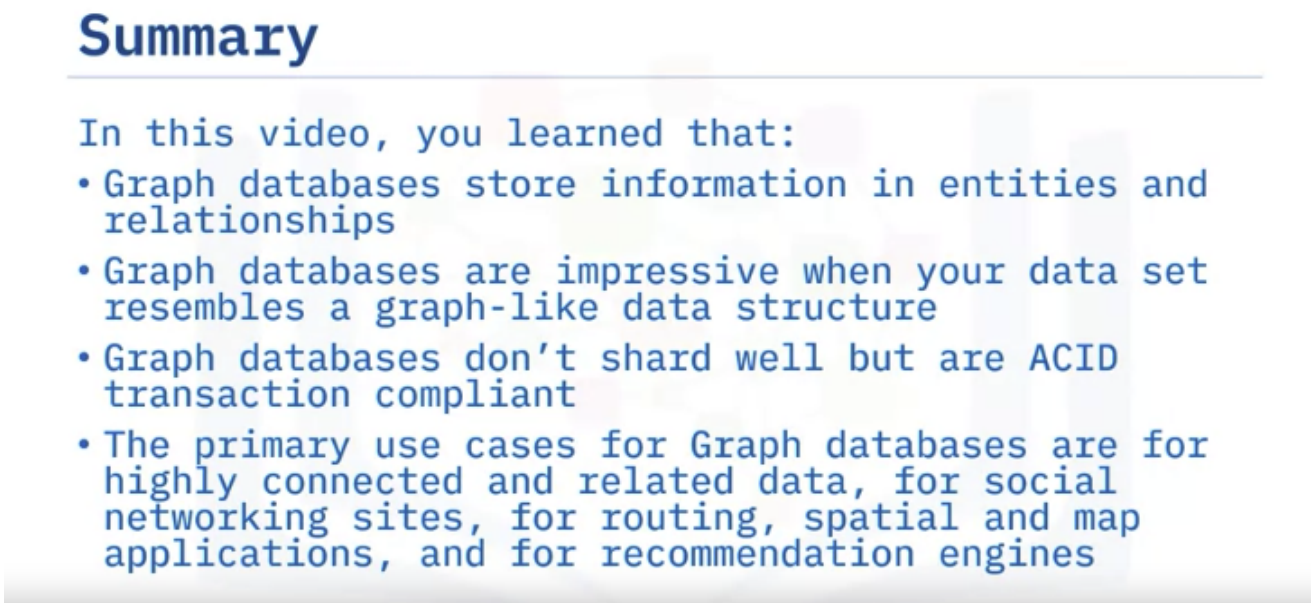
- For highly connected and related data
- Social networking
- Routing, spatial, and map apps
- Recommendation engines

Graph NoSQL Database Use Cases

Unsuitable Use Cases

- When looking for advantages offered by other NoSQL database categories
- When an application needs to scale horizontally
  - You will quickly reach the limitations associated with these types of data stores
- When trying to update all or a subset of nodes with a given parameter
  - These types of operations can prove to be difficult and non-trivial

Summary



Database Deployment Options

This reading introduces you to the concept of **Database-as-a-Service (or DBaaS**) and enables you to differentiate the available database hosting options.

The major consideration when choosing the best database for your applications and organization is around where to host it and how it is being managed. It is important to understand all of the components to make sure you end up with the simplest and most cost-effective option.

In a traditional in-house **do-it-yourself scenario**, you will own the setup of the underlying hardware and operating system, installation and configuration of the chosen database management system, overall administration including patching and support, and of course how the application data is designed.

This contrasts a little bit with hosted database solutions. A hosted database solution means the provider is choosing what hardware your database runs on, and they are provisioning it for you. But it is your responsibility to provide and install the software, perform the administrative tasks, and do the design. So, at the end of the day, the hosting provider are handing the administrative keys over to you and it's really up to your team of database administrators to keep things scaling and running smoothly.

In comparison, **using a fully managed database-as-a-service (or DBaaS)** is really meant to eliminate the complexity and risk of doing it all in-house, and help development teams get to market faster, scale more smoothly and massively, and provide better performance and availability for end users.

The only concern for users of a fully managed DBaaS is the design and development of their product. Guaranteed uptime, availability, and scalability are all the result of a fully managed DBaaS.

You mitigate risk by offloading database administration and data layer management issues from your development team. And you ensure that your developers need only concern themselves with what really matters – developing better applications for your customers.

Summary and Highlights (Section I - Basics of NoSQL)

- NoSQL means Not only SQL.
- NoSQL databases have their roots in the open source community.
- NoSQL database implementations are technically different from each other.
- There are several benefits of adopting NoSQL databases including storing and retrieving session information, and event logging for apps.
- The four main categories of NoSQL database are Key-Value, Document, Wide Column, and Graph.
- Key-Value NoSQL databases are the least complex architecturally.
- Document-based NoSQL databases use documents to make values visible for queries.
    - In document-based NoSQL databases, each piece of data is considered a document, which is typically stored in either JSON or XML format.

- Column-based databases spawned from the architecture of Google's Bigtable storage system.
- The primary use cases for column-based NoSQL databases are event logging and blogs, counters, and data with expiration values.
- Graph databases store information in entities (or nodes) and relationships (or edges).

Practice Quiz: Basics of NoSQL

Congratulations! You passed!

Grade received 100% **To pass** 60% or higher

Go to next item

1. What are the four main categories of NoSQL databases? **1 / 1 point**

- Key-Value, Relational, Column, Graph
- (•) Key-Value, Document, Column, Graph
- Key-Value, Document, Common, Graph
- Key-Value, Distributed, Column, Graph

Correct
The four major categories of NoSQL databases all have unique characteristics that make them great fits for different types of applications or parts of applications.

2. What is the name of the fully managed service model that many NoSQL databases now leverage? **1 / 1 point**

- Network-as-a-service (NaaS)
- (•) Database-as-a-service (DBaaS)
- Platform-as-a-service (PaaS)
- Software-as-a-service (SaaS)

Correct
Database-as-a-service (DbaaS) is the fully managed service model that many NoSQL databases now leverage.

3. What is the most common trait among all NoSQL databases? **1 / 1 point**

- ○ They are all relational
- ○ They are all open source
- ○ They are all document-based
- ◉ They are all non-relational

Correct
The most common trait among all NoSQL databases is that they are non-relational in nature.

4. Which document formats are typically used for documents stored in a document-based NoSQL database? **1 / 1 point**

- ◉ XML and JSON format
- ○ CSV and JSON format
- ○ XML and CSV format
- ○ PDF and JSON format

Correct
In document-based NoSQL databases, documents are typically stored in XML or JSON format.

5. Which category of NoSQL databases is it recommended not to shard data on? **1 / 1 point**

- ◉ Graph NoSQL databases
- ○ Column-based NoSQL databases
- ○ Document-based NoSQL databases
- ○ Key-Value NoSQL databases

Correct
Sharding a Graph NoSQL database is not recommended since traversing a graph with nodes split across multiple servers can become difficult and hurt performance.

Graded Quiz: Basics of NoSQL

Congratulations! You passed!

Grade received 100%

Latest Submission Grade 100%

To pass 60% or higher

Retake the assignment in **7h 59m**

Go to next item

1. What common trait does the NoSQL family of databases share? **1 / 1 point**

- ○ Exclusion of SQL
- ○ Tabular style
- ◉ Non-relational
- ○ Technology

Correct
This is a family of databases that vary widely in style and technology, but which all share a common trait in that they are non-relational in nature (they are not a standard row and column relational database management system).

2. What may be the most common reason to use a NoSQL database? **1 / 1 point**

- ○ Availability
- ◉ Scalability
- ○ Security
- ○ Consistency

Correct
The elasticity of scaling both up and down to meet the varying demands of applications is key.

3. What makes Key-Value NoSQL databases powerful for basic CRUD operations? **1 / 1 point**

- ○ They shard easily across nodes.
- ○ The value blob is opaque.
- ◉ They are represented as a hashmap.
- ○ They are atomic for key operations.

Correct
Because Key-Value stores are represented as a hashmap, they are powerful for basic Create-Read-Update-Delete operations.

4. Which use case would be a poor choice for a document type NoSQL database? **1 / 1 point**

- ○ Online blogging
- ○ Operational data sets for a web app
- ○ Event logging for a process
- ◉ Aggregate-oriented design

Correct
If the data naturally falls into a normalized tabular model, then a relational database may be a better choice.

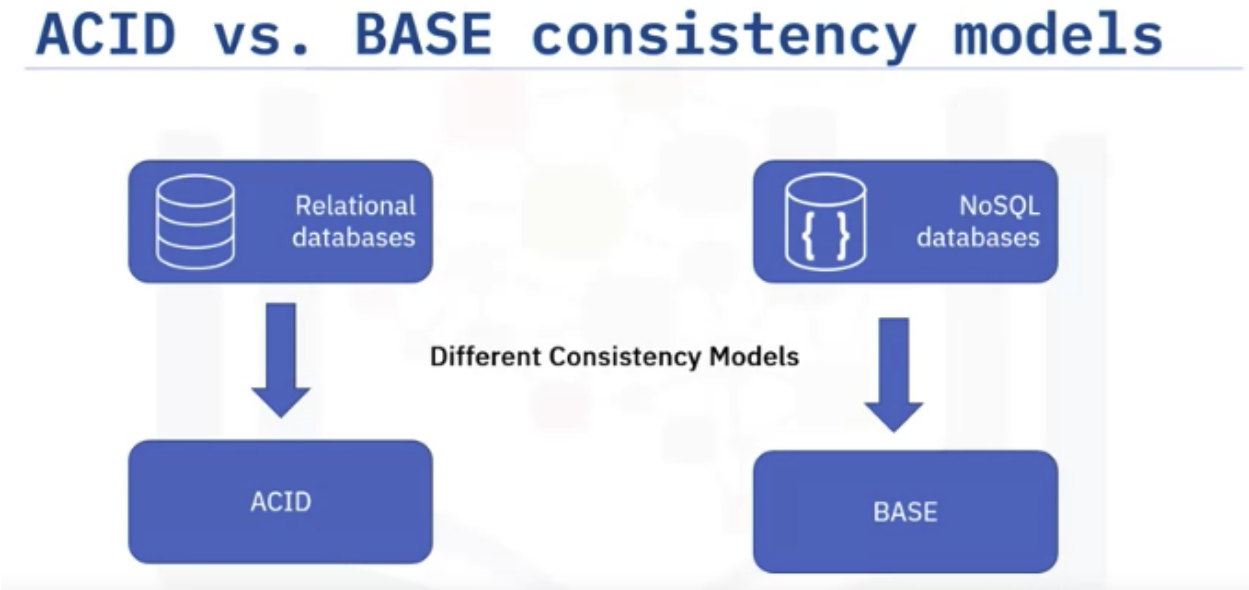
5. What is a characteristic of column-based NoSQL databases? **1 / 1 point**

- ○ Rows in column families share a common key or identifier.
- ○ Rows in column families are required to share the same columns.
- ◉ Rows in column families can share all, a subset, or none of the columns.
- ○ Columns are grouped together in families because they are often accessed individually.

Correct
They can share all, a subset, or none of the columns, and columns can be added to any number of rows.

Working w/Distributed Data (Section II : Wk1)

ACID vs BASE

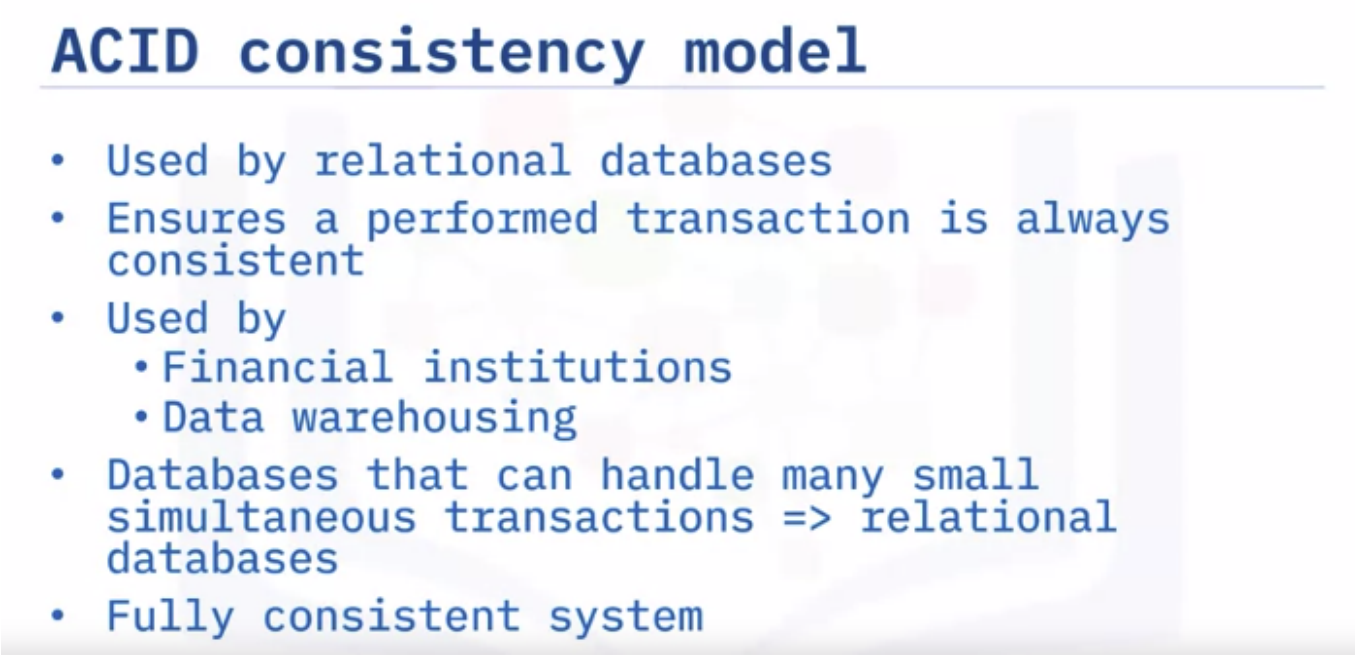


- Consistency Models
  - The two most common consistency models that are known and in use nowadays are: **ACID** and **BASE**.
  - Relational databases use the ACID model
  - NoSQL databases use the BASE model.
- Although **ACID** and **BASE** models are often seen as enemies or one versus the other, the truth is that both come with their advantages and disadvantages.

ACID Definition

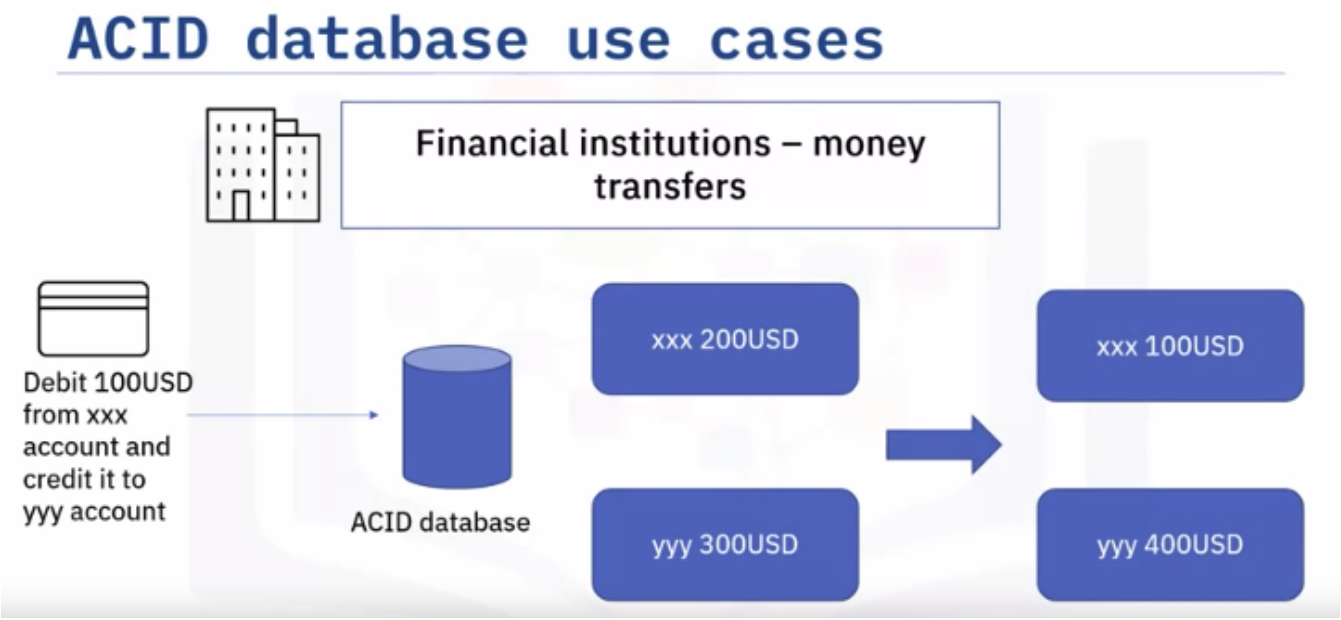
- The ACID acronym stands for:
  - **Atomic** – All operations in a transaction succeed or every operation is rolled back.
  - **Consistent** – On the completion of a transaction, the structural integrity of the data in the database is not compromised.
  - **Isolated** – Transactions cannot compromise the integrity of other transactions by interacting with them while they are still in progress.
  - **Durable** – The data related to the completed transaction will persist even in the case of network or power outages.
    - If a transaction fails, it will not impact the already changed data.

ACID Consistency Model



- The ACID consistency model ensures that a performed transaction is always consistent. This makes it a good fit for businesses that deal with online transaction processing, such as financial institutions, or data warehousing types of applications. These organizations need database systems that can handle many small simultaneous transactions, like relational database can. An ACID system provides a consistent model you can count on for the structural integrity of your data.

ACID Use Case (Perspective)

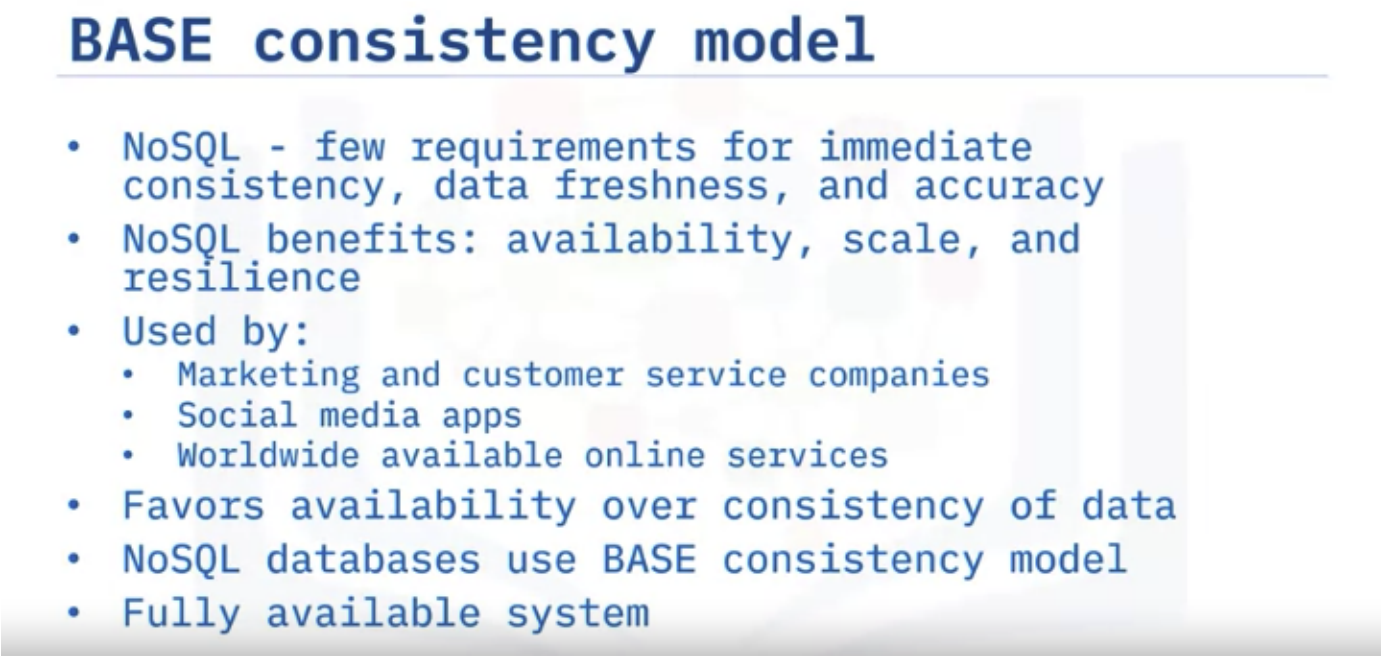


- While there are many possible use cases for ACID databases, one stands out:
    - financial institutions will almost exclusively use ACID databases for their money transfers, since these operations depend on the atomic nature of ACID transactions.
    - An interrupted transaction that is not immediately removed from the database can cause a lot of issues.
        - Money could be debited from one account and, due to an error, never credited to the other, or not credited at all.

BASE Definition

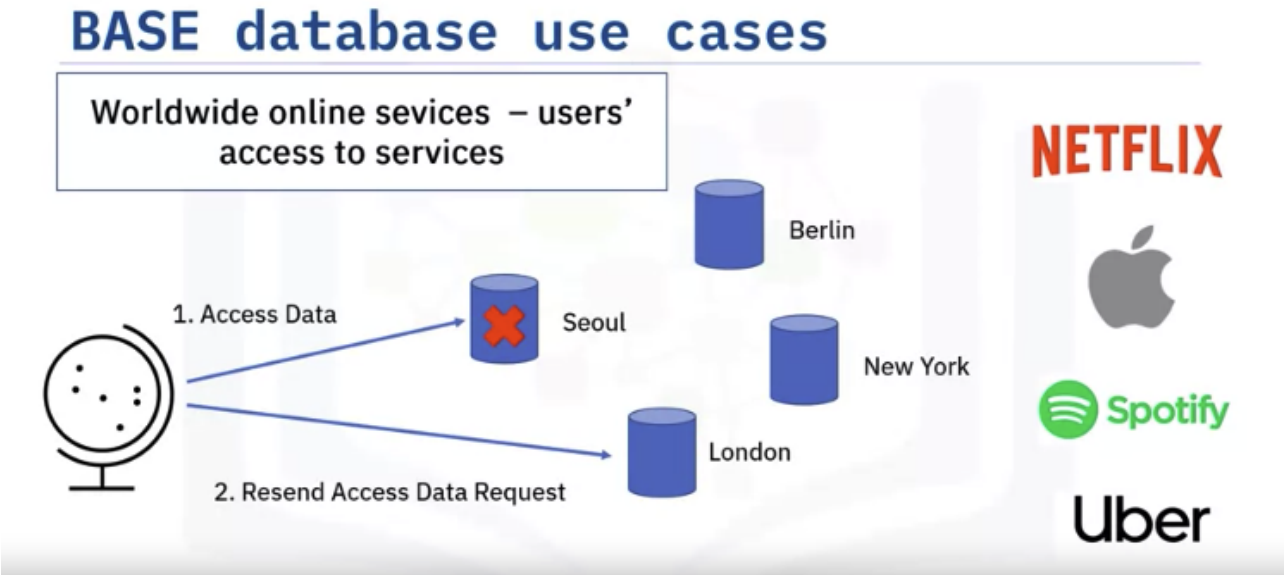
- The BASE acronym stands for:
    - **Basically Available** – Rather than enforcing immediate consistency, BASE-modelled NoSQL databases will ensure availability of data by spreading and replicating it across the nodes of the database cluster.
    - **Soft state** – Due to the lack of immediate consistency, data values may change over time. In the BASE model data, stores don't have to be write-consistent, nor do different replicas have to be mutually consistent all the time.
    - **Eventually consistent** – The fact that the BASE model does not enforce immediate consistency does not mean that it never achieves it.
        - However, until it does, data reads might be inconsistent.

BASE Consistency Model



- In the NoSQL database world, ACID transactions are less fashionable because some databases have loosened the requirements for immediate consistency, data freshness, and accuracy.
- They do this to gain other benefits, such as **availability**, **scale**, and **resilience**.
- The BASE consistency model is used by: Marketing and customer service companies that deal with sentiment analysis and social network research.
    - Social media applications that contain huge amounts of data and need to be available at all times. And, worldwide available online services like Netflix, Spotify, and Uber.
    - A BASE data store values availability over consistency, but it doesn't offer guaranteed consistency of replicated data at write time.
- NoSQL databases use the BASE consistency model.
    - Essentially, the BASE consistency model provides high availability.

BASE Use Cases



- BASE data stores can be, and are, used in a lot of specific use cases.
    - The fame of BASE databases increased because well-known, worldwide online services such as Netflix, Apple, Spotify, and Uber use them in applications like user profile data storage.
    - A common characteristic of these services is that they need to be accessible at all times, no matter which part of the globe you are in.
        - So, if a part of their database cluster becomes unavailable, the system needs to be able to serve the user requests seamlessly.

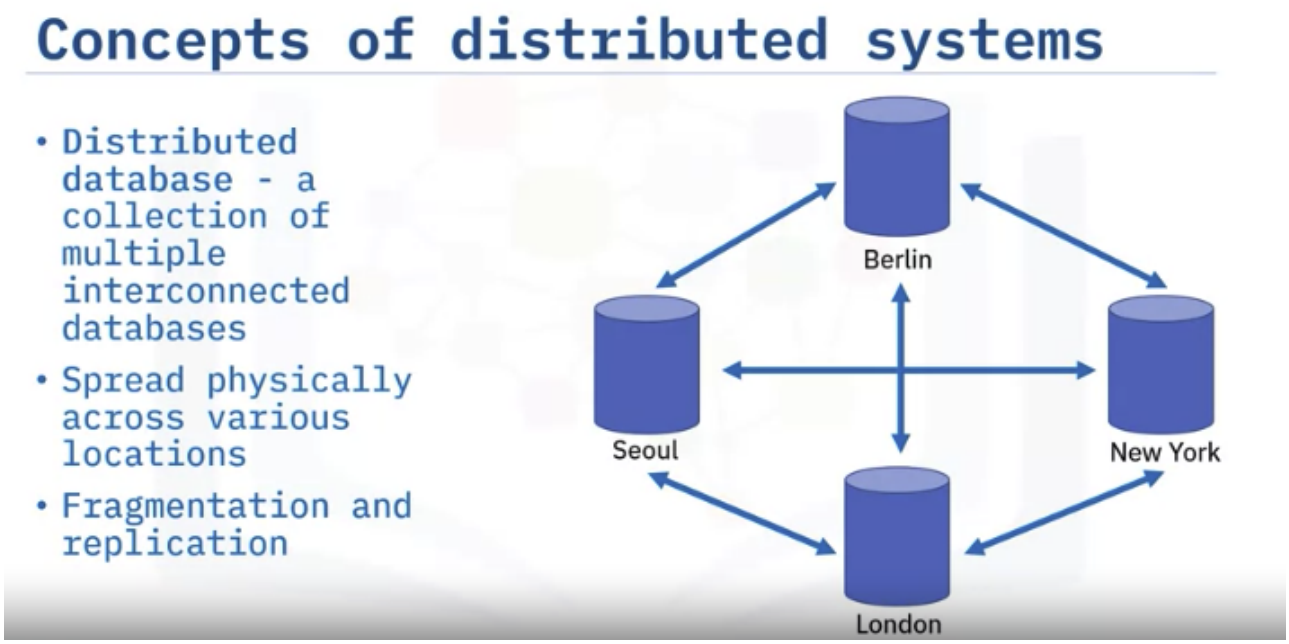
Summary

In this video you have learned that:

- ACID and BASE are database consistency models
- ACID = Atomicity, Consistency, Isolated, Durable
- BASE = Basically Available, Soft state, Eventually consistent
- ACID (RDBMS) is focused on consistency
- BASE (NoSQL) is focused on availability
- Usage selected by case-by-case basis selection

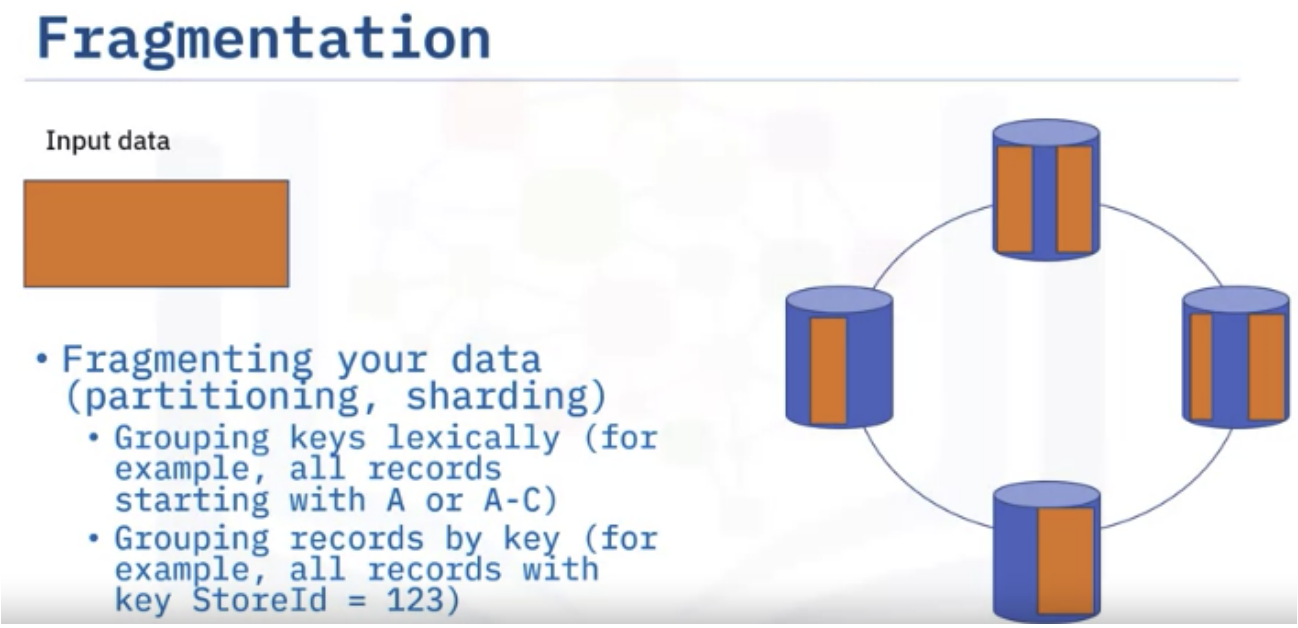
Distributed Databases

Concepts of Distributed Systems



- A distributed database is physically distributed across the data sites by fragmenting and replicating the data.
    - A distributed database follows the **BASE** consistency model

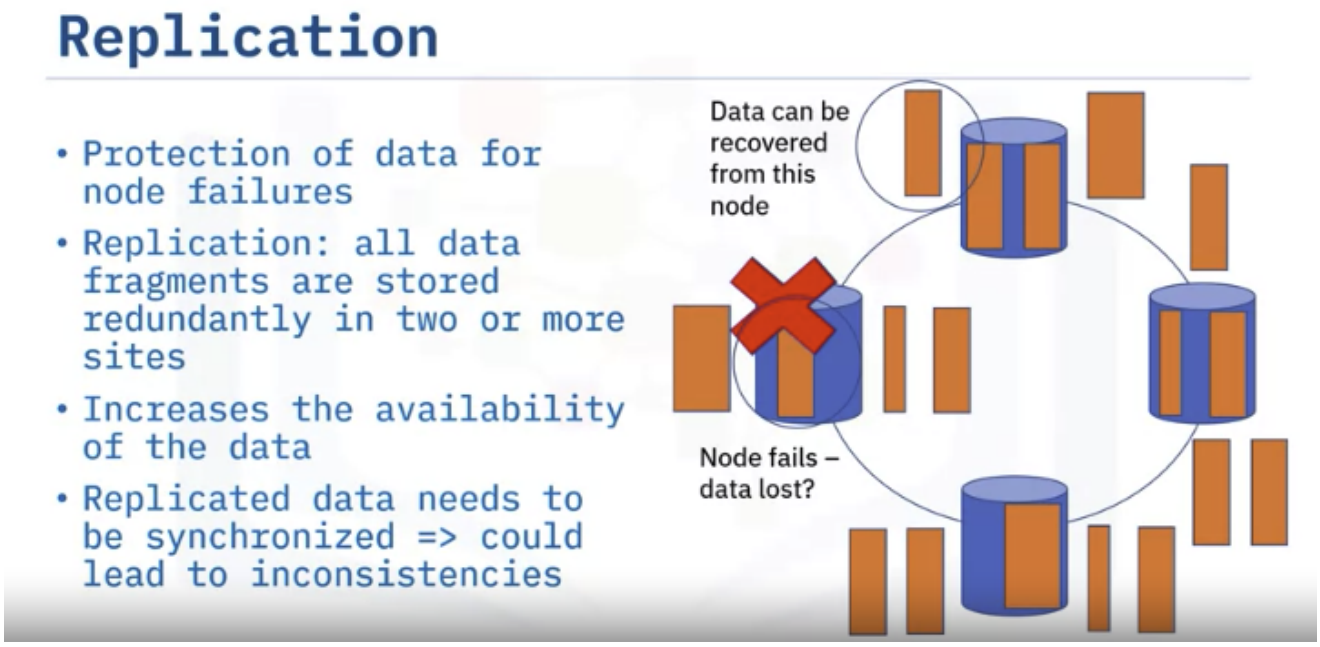
Fragmentation



- To store a large piece of data on all the servers of a distributed system, you need to break your data into smaller pieces.
- This process, known as fragmentation of data, is also called **partitioning** of data, or **sharding** of data, by some NoSQL databases.
- No matter the name, all distributed systems need to expose a way to store a large chunk of data.
    - This process is usually done by the key of the ‘key:value’ record in two ways:
    - By either grouping all keys lexically.
        - For example, all keys that start with A or between A and C can be found on a specific server, or by
    - Grouping all records that have the same key, and placing them on the same server.
        - For example, all transactions from a store, where ‘StoreID’ is the key of the records.
            - In this way, with a query like "give me all sales from a store," all records will be on a single server.

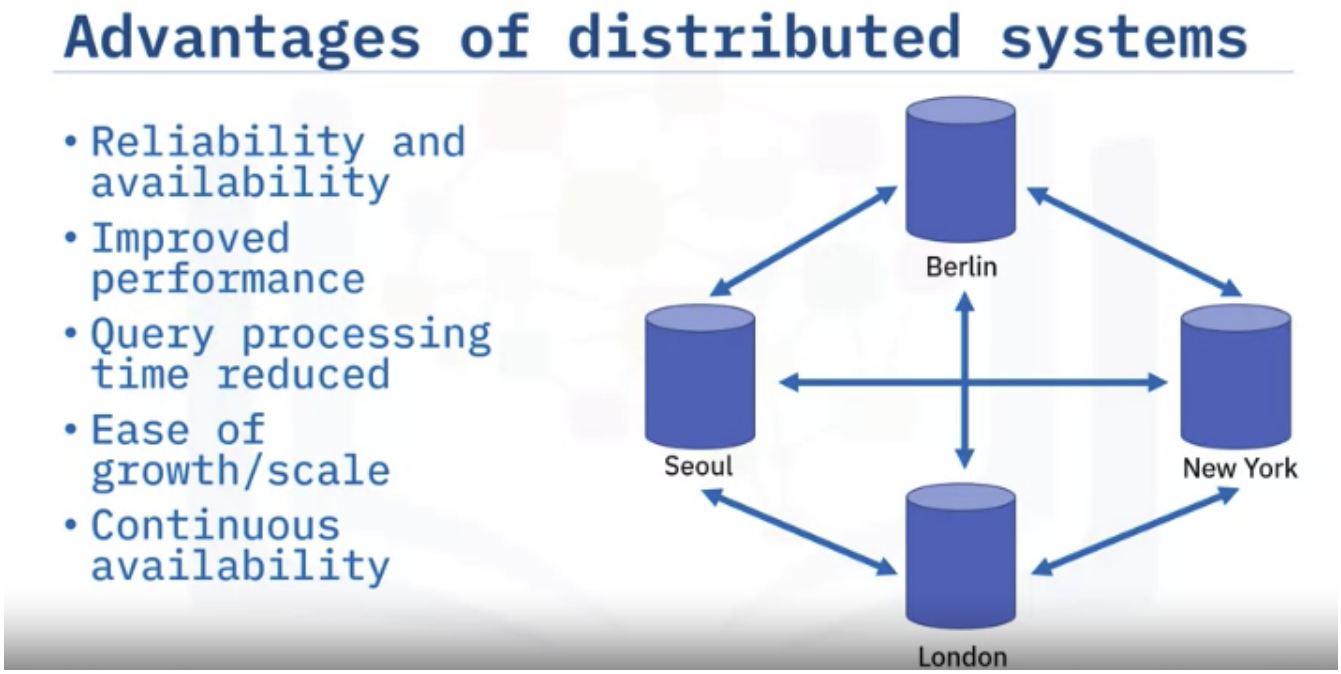
Replication

- Data is now distributed to all the cluster's nodes, but how do we make sure that if a node fails, we don't lose all the data in that node?
    - This is done through … **replication**

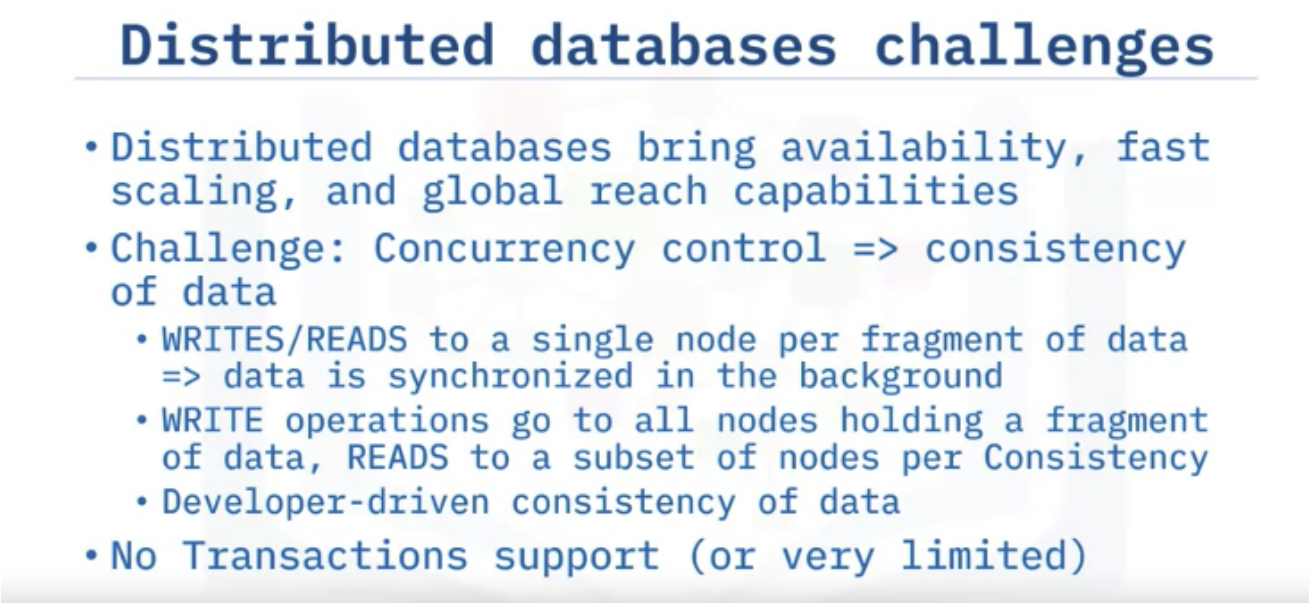


- This is done through replication, where all fragments (or **partitions**, or **shards**) of your data are <u>stored redundantly in two or more sites</u>.
    - Hence, in replication, systems maintain copies of data.
- Replication increases the availability of data in different sites. If one node fails, that piece of data can be retrieved from another node.
    - However, it has certain disadvantages as well.
    - <u>Data needs to be constantly synchronize</u>d.
        - Any change made at one site needs to be replicated to every site where that related data is stored, or else it will lead to inconsistency.

Advantages of Distributed Systems

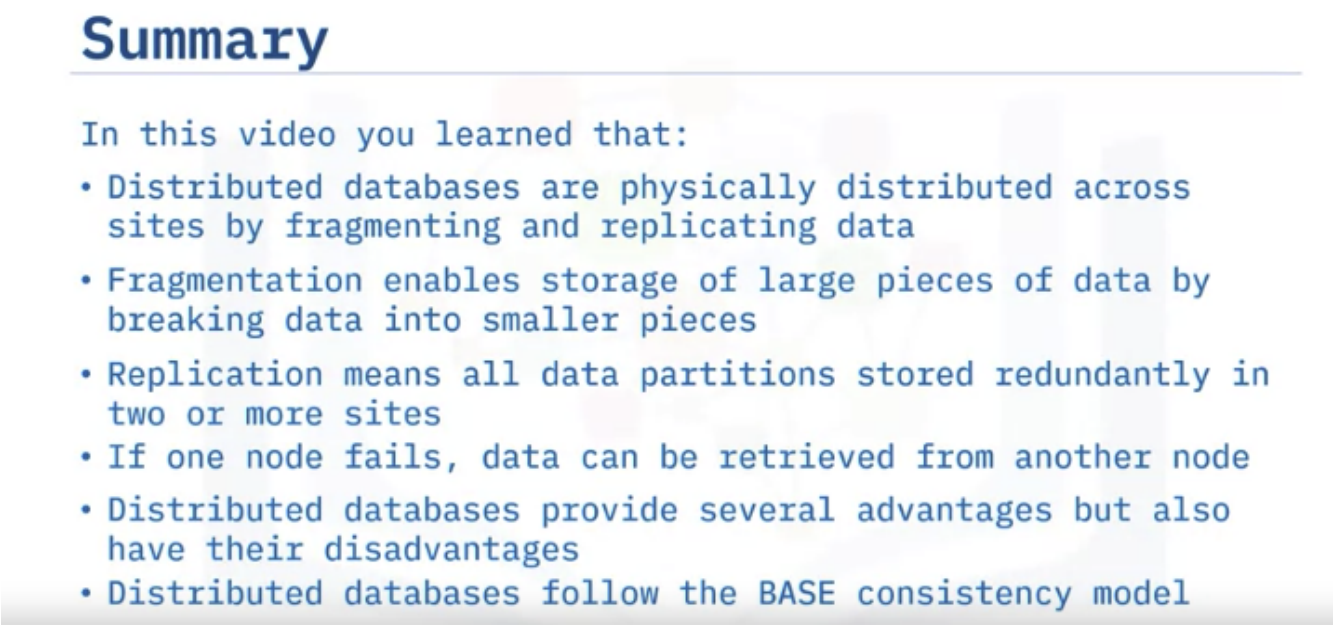


- They allow more reliability and availability.
- The data is replicated at multiple sites.
    - If the local server is unavailable, the data can be retrieved from another available server.
- Another advantage of distributed databases is improved performance, especially for high volumes of data.
    - Query processing time is reduced, which also helps improve performance.
    - You can easily grow (or scale) to increase your system capacity, just by adding new servers to the cluster.
- Distributed systems also provide continuous operation with no more reliance on the central site.
- Distributed databases solve a lot of the technological issues for today's application services, like availability, fast scaling, and global reach



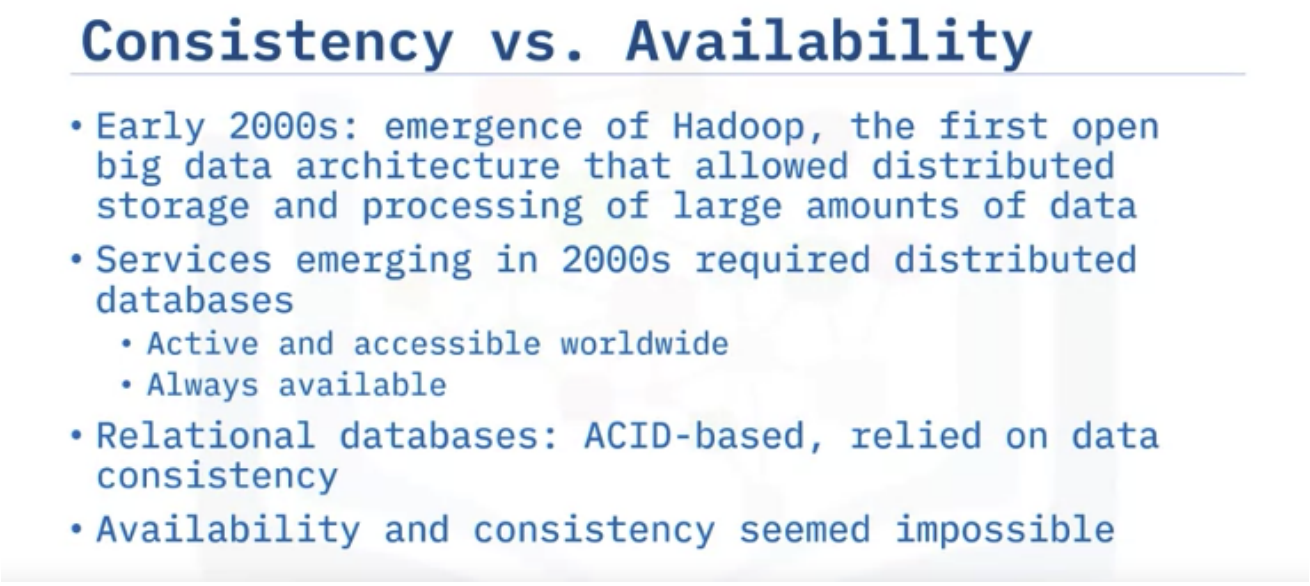
- One is **concurrency control**.
    - Because the same piece of data is stored in multiple locations, if you modify (update or delete) your data, how can data synchronization be secured?
- To solve this issue, some distributed databases direct operations for a certain fragment of data to only one node, leaving the cluster to synchronize with the other nodes.
- Others WRITE to all nodes holding that particular fragment of data and read from as many nodes as required per CONSISTENCY.
- In both cases the developer can control the consistency of the operation, or how many nodes need to answer for a certain operation to be considered successful.
- Due to concurrency control issues, distributed databases, by design, don't really support Transactions (or provide a very limited version of them).

Summary

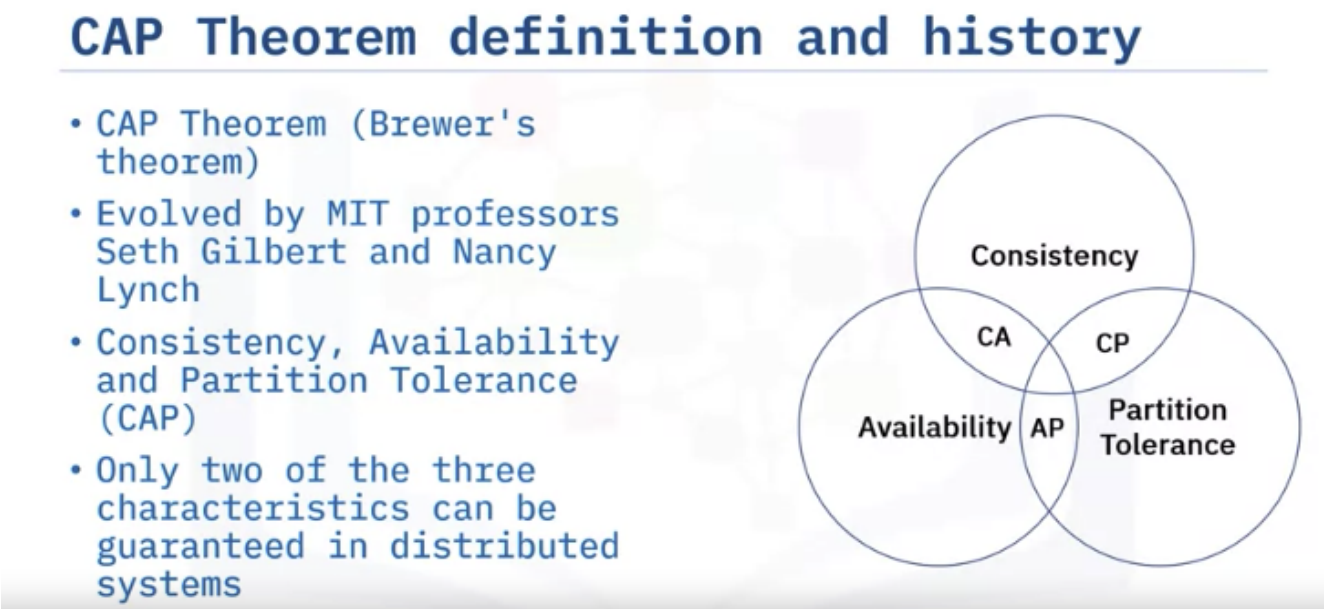


CAP Theorem

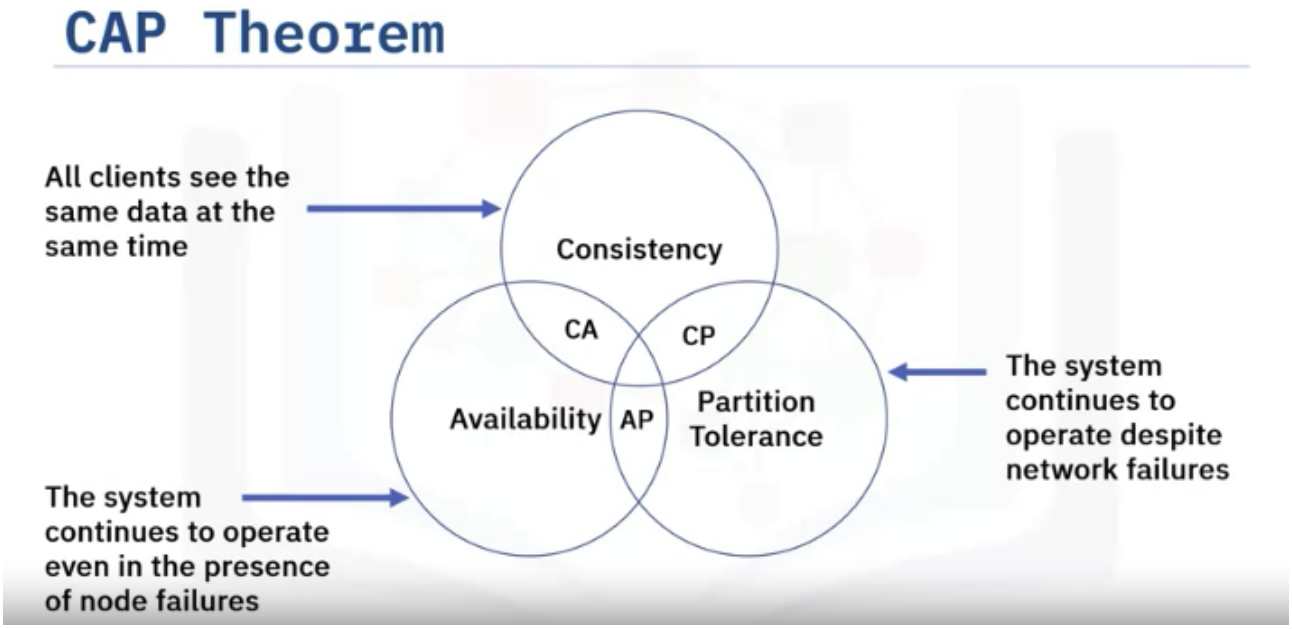
Consistency vs Availability



CAP Theorem Defined

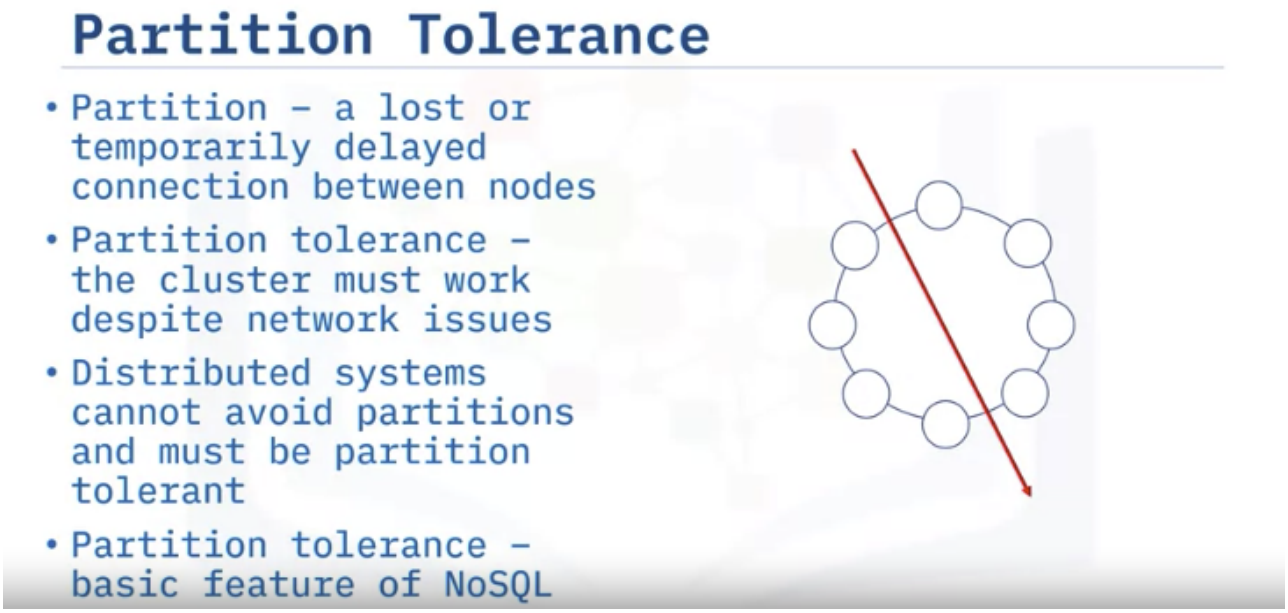


- For relational databases that rely so much on the **consistency of data**, the new concept of availability while having a distributed system seemed impossible, and this was proven by the CAP Theorem.
- The CAP Theorem is also called Brewer's Theorem, because it was first advanced by Professor Eric A. Brewer during a talk he gave on distributed computing in the year 2000.
- Two years later it was revised by MIT professors Seth Gilbert and Nancy Lynch, and there have been many other contributors since.
    - The theorem states that there are three essential system requirements necessary for the successful design, implementation, and deployment of applications in distributed systems.
    - These are **Consistency**, **Availability**, and **Partition Tolerance**, or CAP.
        - A distributed system can guarantee delivery of only two of these three desired characteristics.



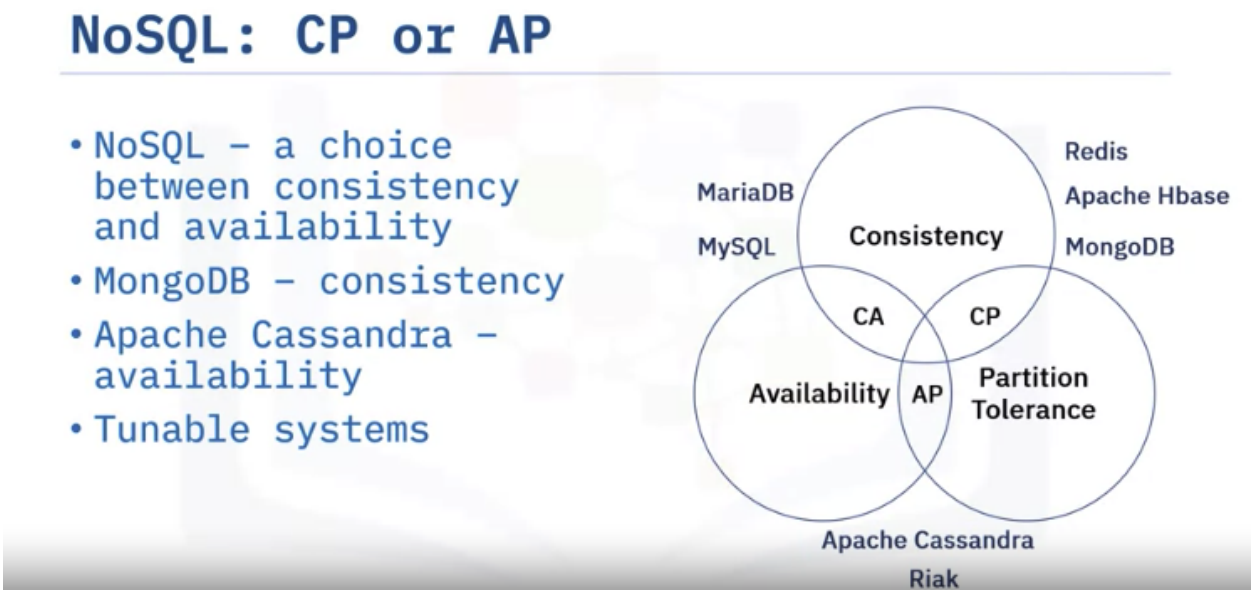
- **Consistency** refers to whether a system operates fully or not.
    - Do all nodes within a cluster see all the data they are supposed to?
- **Availability** means just as it sounds.
    - Does each request get a response outside of failure or success?
- **Partition Tolerance** represents the fact that a given system continues to operate even under circumstances of data loss or network failure

Deeper Dive Partition Tolerance



- A **partition** is a communications break within a distributed system—a lost or temporarily delayed connection between nodes.
- **Partition tolerance** means that the cluster must continue to work despite any number of communication breakdowns between nodes in the system.
    - In distributed systems, partitions can't be avoided.
- Therefore, partition tolerance becomes a basic feature of native distributed systems such as NoSQL.
- In a cluster with eight distributed nodes, a network partition could occur, and communication will be broken between all the nodes.
    - In our case, instead of one 8-node cluster we will have two smaller 4-node clusters available.
    - Consistency between the two clusters will be achieved when network communication is re-established.
- Partition tolerance has become more of a necessity than an option in distributed systems.
    - It is made possible by sufficiently replicating records across combinations of nodes and networks.

AP or CP



- For such systems as NoSQL, since partition tolerance is mandatory, a system can be either
    - Consistent and Partition Tolerant (CP) or
    - Available and Partition Tolerant (AP).
- Existing NoSQL systems, like **MongoDB** or **Cassandra**, can be classified using CAP Theorem.
    - For example, **MongoDB** chooses consistency as the primary design driver of the solution, and
    - **Apache** Cassandra chooses availability.
- This doesn't mean that MongoDB cannot be available, or that Cassandra cannot become fully consistent.
    - It means that these solutions first ensure that they are consistent (in the case of MongoDB) or available (in the case of Cassandra) and the rest is tunable.

Summary

Summary

In this video you have learned that:

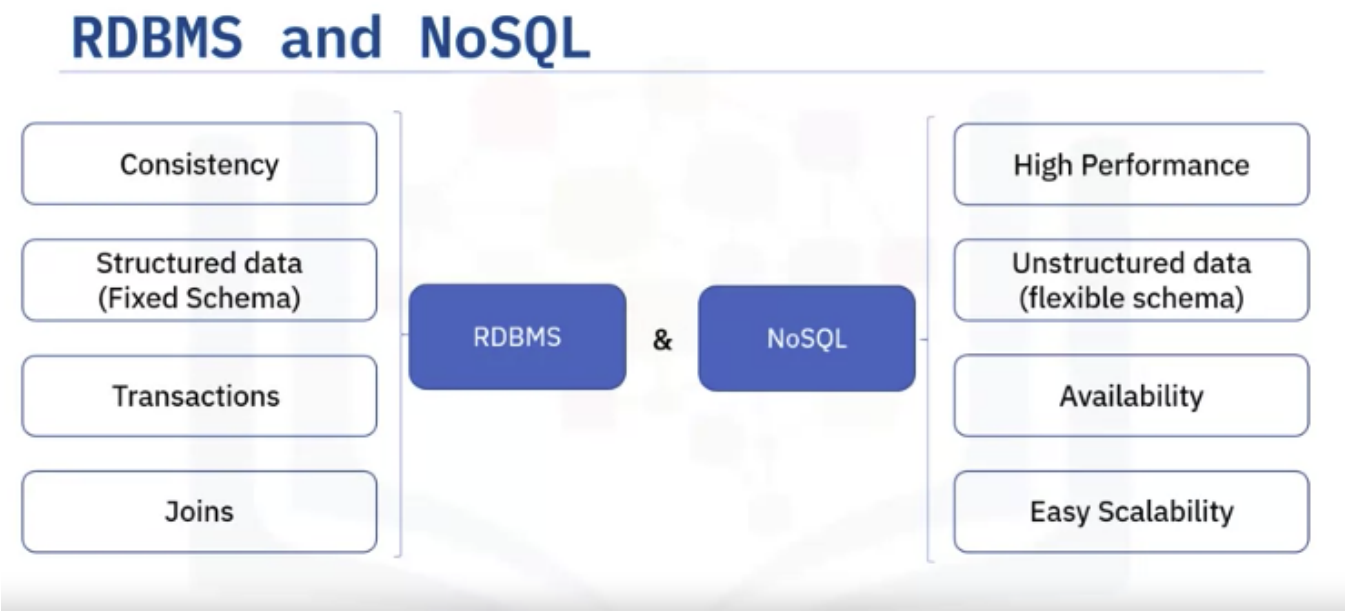
- CAP theorem can be used to classify NoSQL databases
- NoSQL databases choose between availability and consistency
- Partition Tolerance is a basic feature of NoSQL databases

Challenges in Migrating from RDBMS to NoSQL Databases

RDBMS to NoSQL: Overview

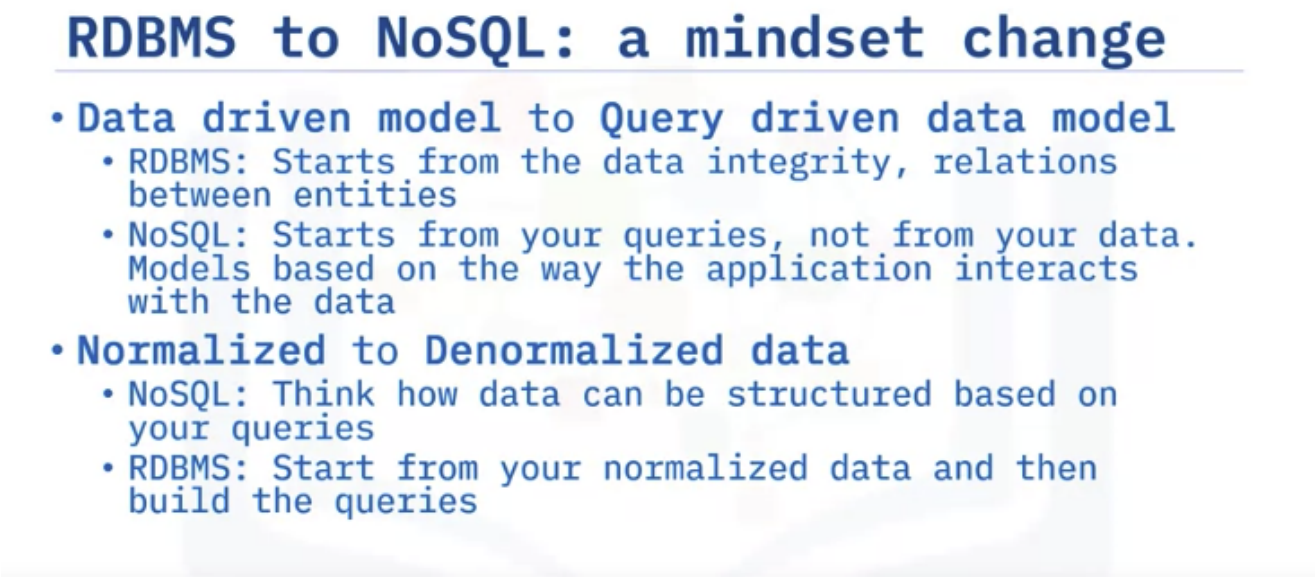
- NoSQL not a de facto replacement of RDBMS
- RDBMS and NoSQL cater to different use cases
- RDBMS to NoSQL should be done based on careful case-by-case analysis (need for performance? flexibility?)

Picking (Characteristics of Each)

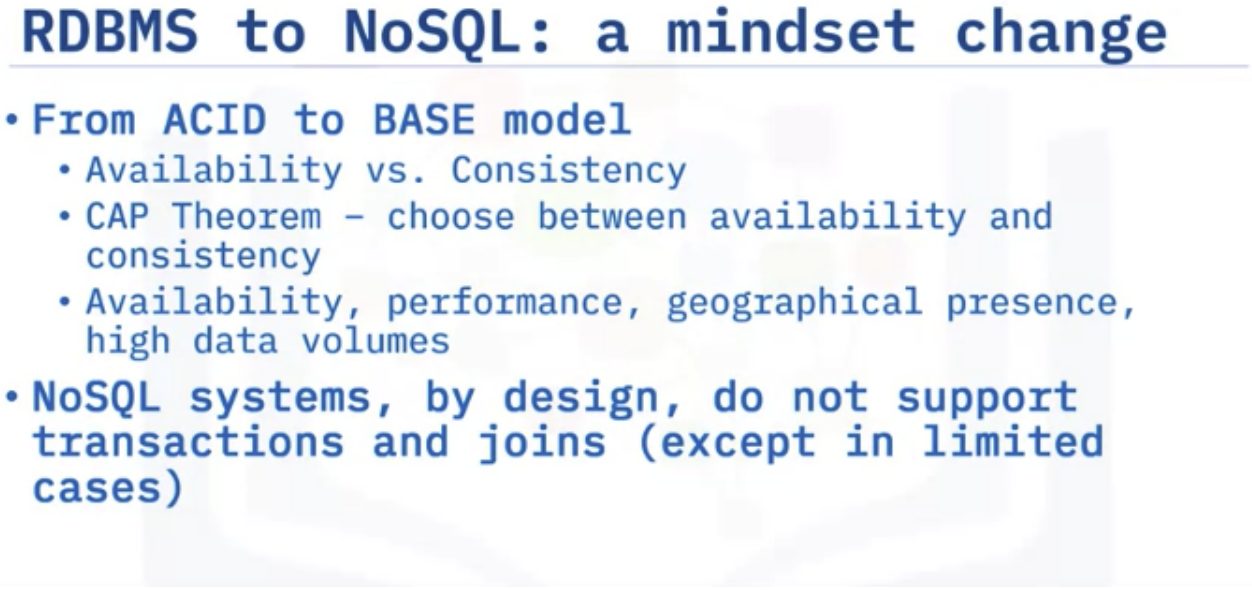


- There might be cases in which both relational and NoSQL databases will be needed.
    - If you have too much data and need performance, and need to scale fast,
    - But at the same time, you also need transactions support, and complex joins on your data,
        - then you might think of a combined solution.

Mindset Change

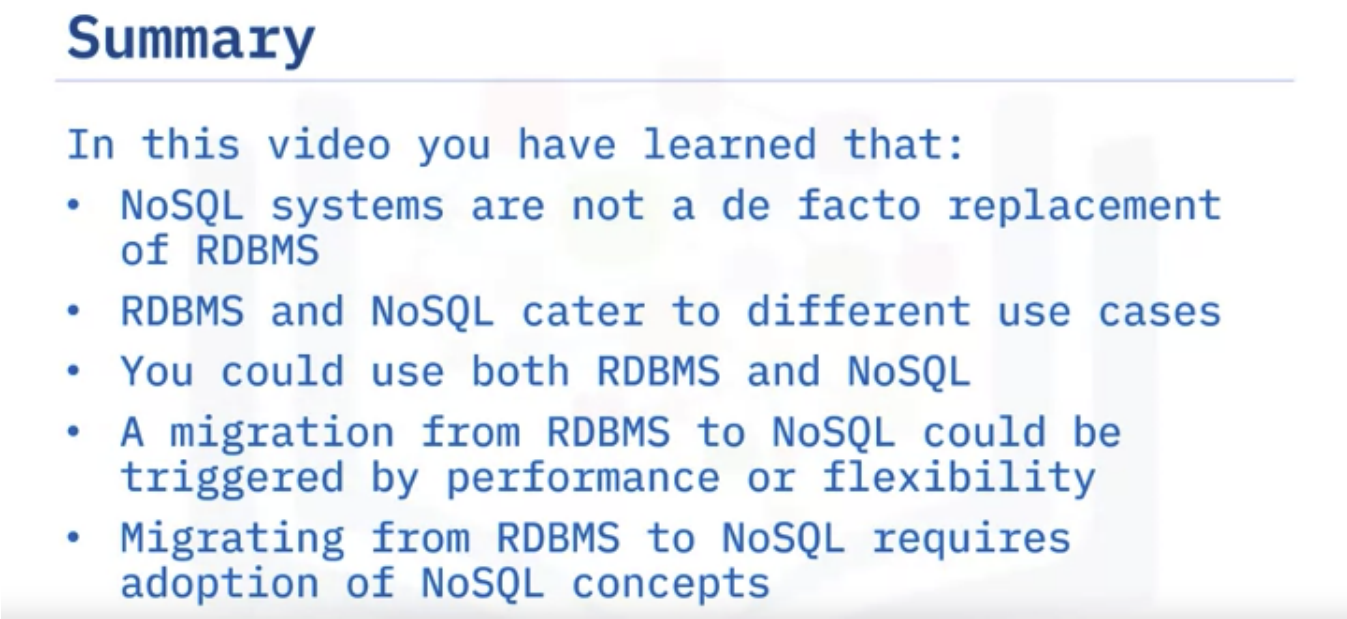


- When embracing the NoSQL world and coming from a relational one, there are a few things you should pay attention to:
    - Like the fact that in NoSQL your data model is driven by your queries, not the data itself.
        - In a relational world the solution design starts from the data, the entities, and their relationship.
    - In NoSQL it's not the data that drives your data model or schema.
        - It's the way your application accesses the data and the queries you are going to make.
    - In NoSQL, models should be based on how the app interacts with the data, rather than how the model can be stored as rows in one or more tables.
    - Another factor to pay attention to is the fact that in RDBMS data is normalized, while in NoSQL it is denormalized.
    - With NoSQL, starting from your query means that you will structure your data on disk accordingly.
        - Thus, you may need to store the same data in different models just to answer the question.
        - This will lead to data denormalization.
    - While data in RDBMS is normalized, you need to be open to the situation in which you start with your queries instead of data



- Remember our earlier **ACID** vs. **BASE** video?
    - When migrating from relational to NoSQL databases, you need to understand that sometimes services require availability more than consistency.
    - When both availability and performance are needed (thus distributed systems),
        - consistency cannot be ensured.
- Remember CAP Theorem?
    - Many of today's online services value availability more than consistency.
    - Because of this, they look for systems that can provide it, taking into consideration the amount of data they are dealing with, and their geographical presence.
- One last thing to know when dealing with NoSQL databases:
    - These are not designed to support transactions, or joins, or complex processing, except in limited cases.
    - You need to consider this when moving from RDBMS to NoSQL.

Summary



- NoSQL systems are not a de facto replacement of RDBMS.
- RDBMS and NoSQL cater to different use cases.
- Your solution could use both RDBMS and NoSQL.
- A migration from RDBMS to NoSQL could be triggered by requirements of performance driven by data volume, or flexibility in the schema or system scalability.
- Migrating from RDBMS to NoSQL requires adoption of NoSQL concepts

Summary and Highlights

- ACID stands for Atomicity, Consistency, Isolated, Durable.
- BASE stands for Basic Availability, Soft-state, Eventual Consistency.
- ACID and BASE are the consistency models used in relational and NoSQL databases.
- Distributed databases are physically distributed across data sites by fragmenting and replicating the data.
- Fragmentation enables an organization to store a large piece of data across all the servers of a distributed system by breaking the data into smaller pieces.
- You can use the CAP Theorem to classify NoSQL databases.

- Partition Tolerance is a basic feature of NoSQL databases.
- NoSQL systems are not a de facto replacement of RDBMS.
- RDBMS and NoSQL cater to different use cases, which means that your solution could use both RDBMS and NoSQL.

Practice Quiz: Working with Distributed Data

Congratulations! You passed!

Grade received 100% **To pass** 50% or higher

Go to next item

1. What is one way that a distributed NoSQL database usually shards data? **1 / 1 point**

- ○ By grouping all keys numerically
- ○ By grouping all records that have the same data on the same server
- ○ By distributing all records that share the same key across multiple servers
- ◉ By grouping all keys lexically

Correct
All distributed systems need to expose a way to store a large chunk of data. For example, all keys that start with A can be found on a specific server.

2. Which requirement would prompt you to consider choosing RDBMS over NoSQL? **1 / 1 point**

- ○ Flexible schema
- ○ Easy scalability
- ◉ Complicated joins
- ○ High availability

Correct
NoSQL systems, by design, do not support transactions and joins (except in limited cases).

3. What requirement does the BASE model support? **1 / 1 point**

- (●) Eventual consistency
- ( ) Isolated consistency
- ( ) Immediate consistency
- ( ) Atomic consistency

Correct
A BASE data store values availability over consistency, but it doesn't offer guaranteed consistency of replicated data at write time.

4. What is the best way to describe the Partition Tolerance characteristic of the CAP Theorem? **1 / 1 point**

- ( ) The cluster must not continue to work during communication breakdowns between nodes in the system.
- (●) The system continues to operate despite data loss or network failures.
- ( ) The system continues to operate even in the presence of node failures.
- ( ) All nodes within a cluster see all the data they are supposed to at the same time.

Correct
Partition Tolerance means that a given system continues to operate even under circumstances of data loss or network failure. The cluster must continue to work despite any number of communication breakdowns between nodes in the system.

Graded Quiz: Working with Distributed Data

Congratulations! You passed!

Grade received 100%

Latest Submission Grade 100%

To pass 75% or higher

Go to next item

1. Which industry will almost exclusively use ACID databases? **1 / 1 point**

- (•) Financial institutions
- ( ) Online social networks
- ( ) Ecommerce
- ( ) Marketing consultants

Correct
Financial institutions will almost exclusively use ACID databases for money transfers because these operations depend on the atomic nature of ACID transactions.

2. What is data sharding? **1 / 1 point**

- ( ) Replicating data across multiple nodes
- ( ) Retrieving data from a failed node
- ( ) Deleting all data and schema
- (•) Fragmenting data into smaller pieces

Correct
This process, which breaks data into smaller pieces for storage in a distributed system, is also called partitioning of data by some NoSQL databases.

3. How many of the desired characteristics of the CAP Theorem can a distributed system guarantee? **1 / 1 point**

- ○ None
- ◉ Two
- ○ Three
- ○ One

Correct
A distributed system can guarantee delivery of only two of the three desired characteristics necessary for the successful design, implementation, and deployment of applications.

4. What drives your data model in NoSQL? **1 / 1 point**

- ◉ Your queries and the way the app accesses the data
- ○ How the data is stored in one or more tables
- ○ The choice between availability and consistency
- ○ How the data is denormalized

Correct
In NoSQL, the way your application accesses the data for the queries you are going to make is the driver. Models should be based on how the app interacts with the data rather than how the model can be stored as rows in one or more tables.