Deciding the right database : NoSQL vs SQL

Relevel
by Unacademy

Concept to be covered in class:

What is NoSQL?

- How Does a NoSQL (non relational) Database Work?
- Types of NoSQL Databases

- NoSQL vs.RDBMS
- Brief discussion on Vertical Scaling, Horizontal Scaling, Sharding and Partitioning
- Why is NoSQL a good fit in this Project!
- Popular choices for NoSQL DB

- Intro to MongoDB
- MongoDB Features
- MongoDB Example
- Key components of MongoDB Architecture
- Why use MongoDB?

What is NoSQL?

NoSQL is an approach to database management that can incorporate a wide variety of data models, for instance key-value, document, columnar and graph formats. A NoSQL database essentially means that it is non-relational, distributed, flexible and scalable. Additional NoSQL database features include the lack of a database schema.

NoSQL database stands for “Not Only SQL” or “Not SQL.” Though a better term would be “NoREL”, NoSQL caught on. Carl Strozz introduced the NoSQL concept in 1998.
Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.

Brief History of NoSQL Databases

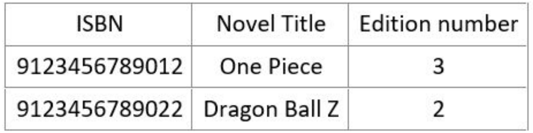
- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebook open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

How Does a NoSQL (non relational) Database Work?

NoSQL databases use a variety of data models for handling data. These databases are optimized specifically for applications that require large data volume, low latency, and flexible data models, which are achieved by giving lesser importance to some of the data consistency restrictions.
Consider the example of modeling the schema for a simple book database:

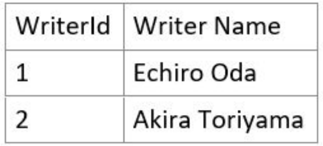
In a relational database, a book record is often disassembled (or “normalized”) and stored in separate tables, and relationships are defined by primary and foreign key constraints.

In this example, the *Novels table* has columns for *ISBN (International Standard Book Number)* which is a 13 digit number unique for every book*, Novel Title, and Edition Number*,



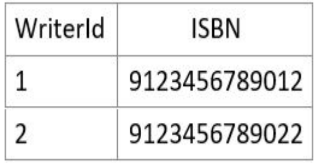
ISBN	Novel Title	Edition number
9123456789012	One Piece	3
9123456789022	Dragon Ball Z	2

the *Writers table* has columns for *WriterID and Writer Name*,



WriterId	Writer Name
1	Echiro Oda
2	Akira Toriyama

and finally the *Writer-ISBN table* has columns for *WriterID and ISBN*.



WriterId	ISBN
1	9123456789012
2	9123456789022

The relational model is designed to enable the database to enforce referential integrity between tables in the database, normalized to reduce the redundancy, and generally optimized for storage.

In a NoSQL database, a book record is usually stored as a *JSON document*. *For each novel, the item, ISBN, Novel Title, Edition Number, Writer Name, and WriterID are stored as attributes in a single document*.

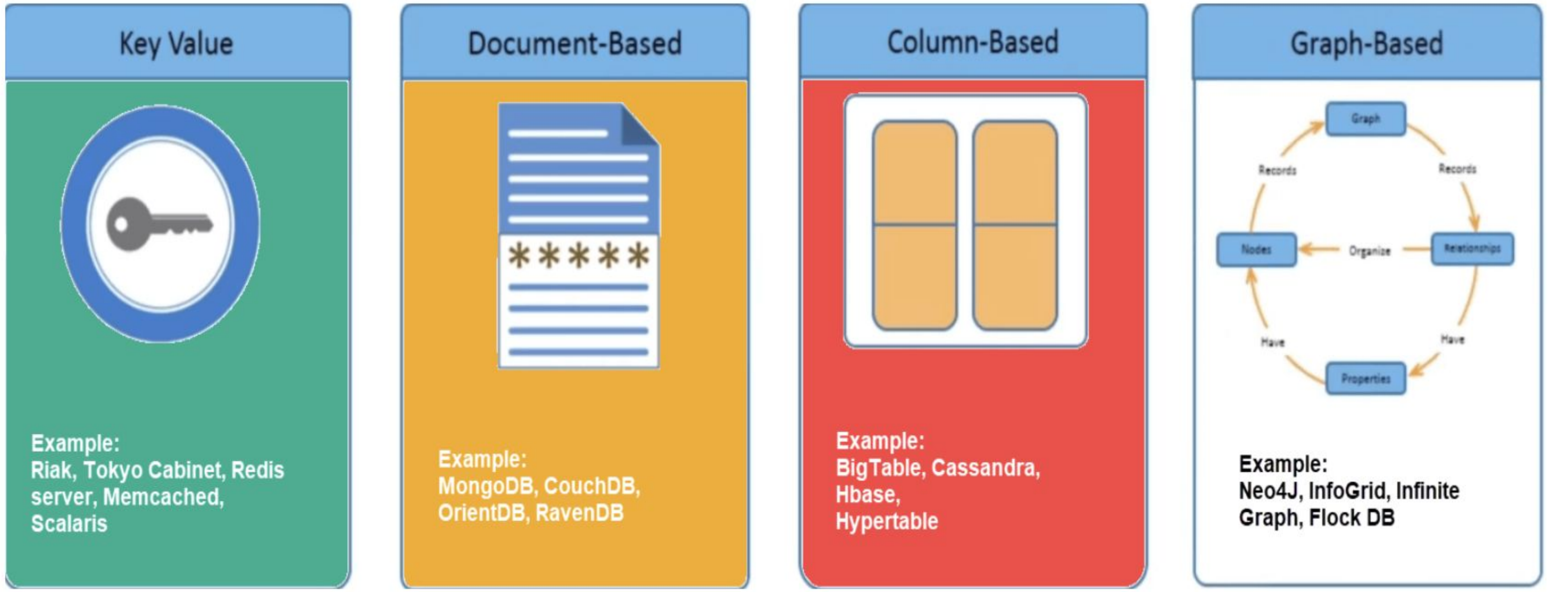
```
{
     " id": {
     "writerId": "1"
     },
     "isbn": "9123456789012",
     "novelTitle": "One Piece",
     "editionNumber": "3",
     "writerName": "Echiro Oda",
}
{
     "_id": {
     "writerId": "2"
     },
     "isbn": "9123456789022",
     "novelTitle": "Dragon Ball Z",
     "editionNumber": "2",
     "writerName": "Akira Toriyama",
}
```

In this model, data is optimized for intuitive development, easiness in accessing and horizontal scalability(We will address this hot word in some time!).

Types of NoSQL Databases

NoSQL Databases are primarily categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented. Every category has its unique attributes and limitations. Users should select the database based on their most important product requirements.
Types of NoSQL Databases:

- Key-value Pair Based
- Column-oriented
- Graphs based
- Document-oriented



Key Value

Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris

Document-Based

Example:
MongoDB, CouchDB, OrientDB, RavenDB

Column-Based

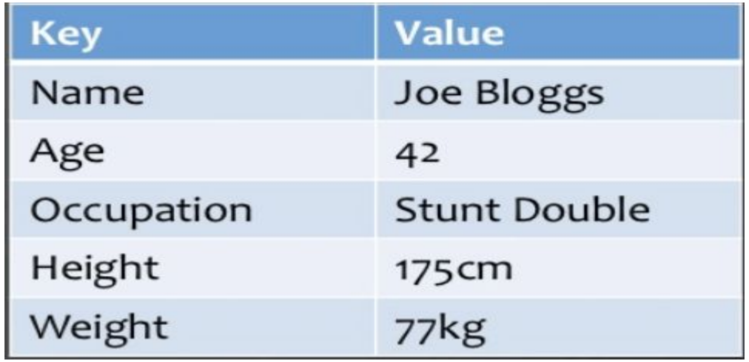
Example:
BigTable, Cassandra, Hbase, Hypertable

Graph-Based

Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

Key Value Pair Based

- Data is stored in key/value pairs.
- It is designed in such a way to handle large amount of data and heavy load.
- What is a Key-value pair storage database? It's a database that stores data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.

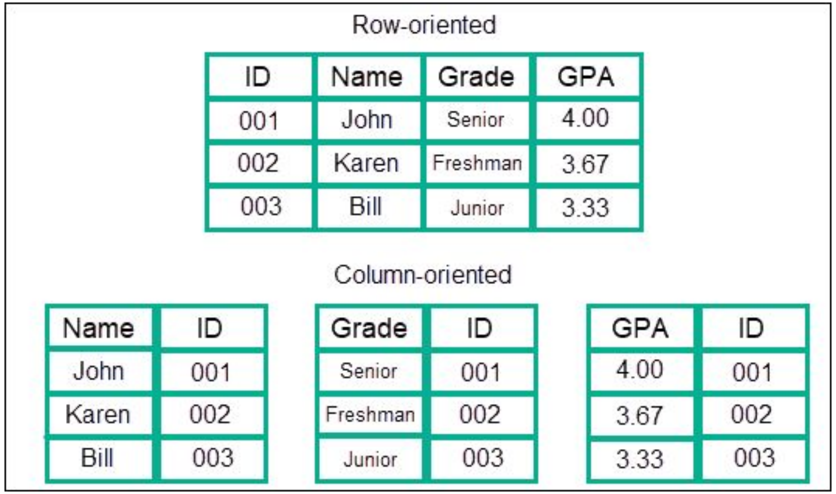


Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

- Use Cases: They work best for shopping cart contents where the shopping list is variable across shoppers, gaming, ad tech, and IoT lend themselves particularly well to the key-value data model.
- Challenges: Key-value stores are less suitable for applications requiring frequent updates or for complex queries involving specific data values, or multiple unique keys and relationships between them.
- Examples: Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases.

Column-Based

- What is Column-Based database? It is a SQL DB but data is organised in columns rather than rows.This essentially makes them function the same way that tables work in relational databases.



Row-oriented

ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

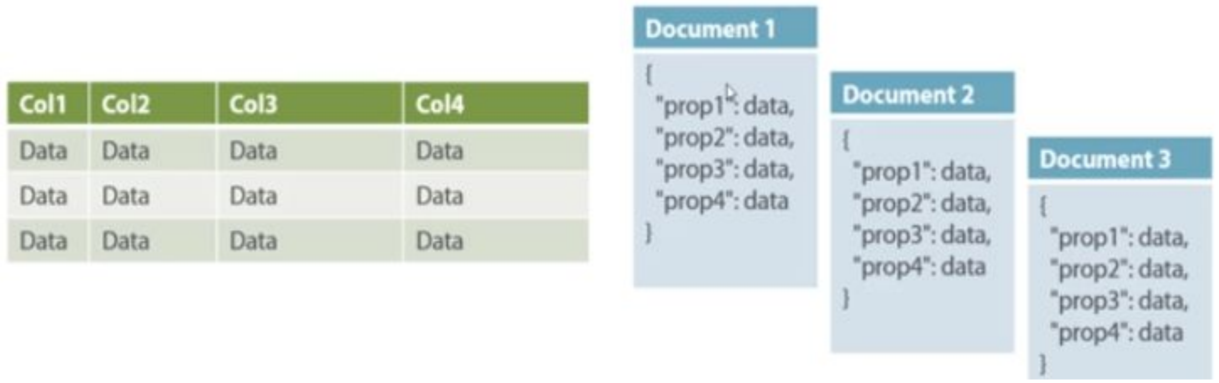
GPA	ID
4.00	001
3.67	002
3.33	003

- When querying, columnar storage lets you skip over all the non-relevant data very quickly.

- Use Cases: For aggregation queries (queries where you only need to lookup subsets of your total data) could be done in a blink of an eye compared to row-oriented DBs.
- Challenges: Writing new data could take more time. If you’re inserting a new record into a row-oriented database, you can simply write that in one operation. But if you’re inserting a new record to a columnar database, you need to write to each column one by one.
- Examples : Cassandra and Apache Hadoop Hbase.

Document-Based

- What is a Document-Oriented NoSQL DB? Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats.
- In the below diagram can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. You do not require to define which make it flexible.



Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1
```
{
 "prop1": data,
 "prop2": data,
 "prop3": data,
 "prop4": data
}
```

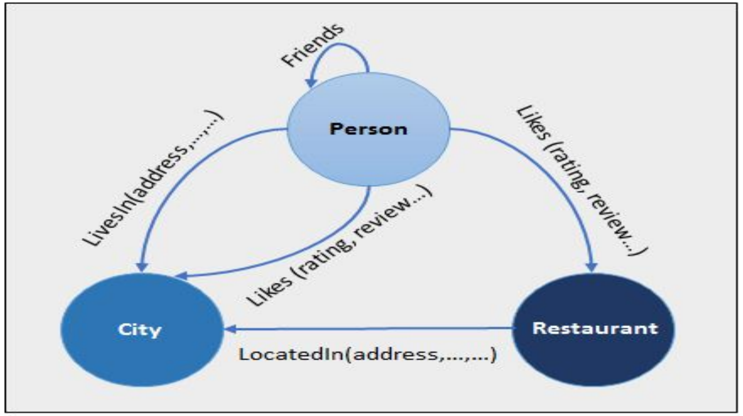
Document 2
```
{
 "prop1": data,
 "prop2": data,
 "prop3": data,
 "prop4": data
}
```

Document 3
```
{
 "prop1": data,
 "prop2": data,
 "prop3": data,
 "prop4": data
}
```

- Use Cases: The document type is mostly used for CMS systems, blogging platforms, real-time analytics & e-commerce applications.
- Challenges: It should not be ideally used for complex transactions which require multiple operations or queries against varying aggregate structures.
- Examples: Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB, are popular Document originated DBMS systems.

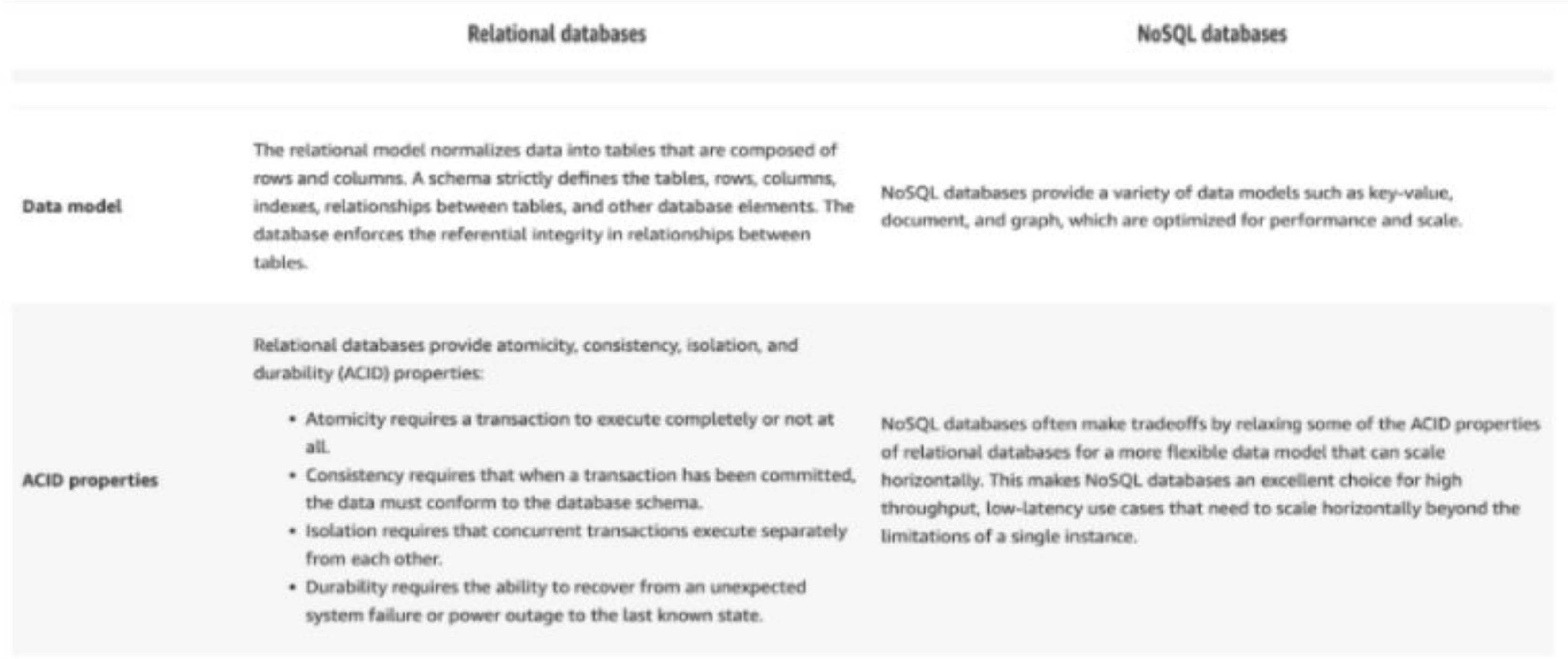
Graph-Based

- What is a Graph based database? A graph type database stores entities/nodes as well the relations amongst those entities.
- The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.
- Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature. Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.

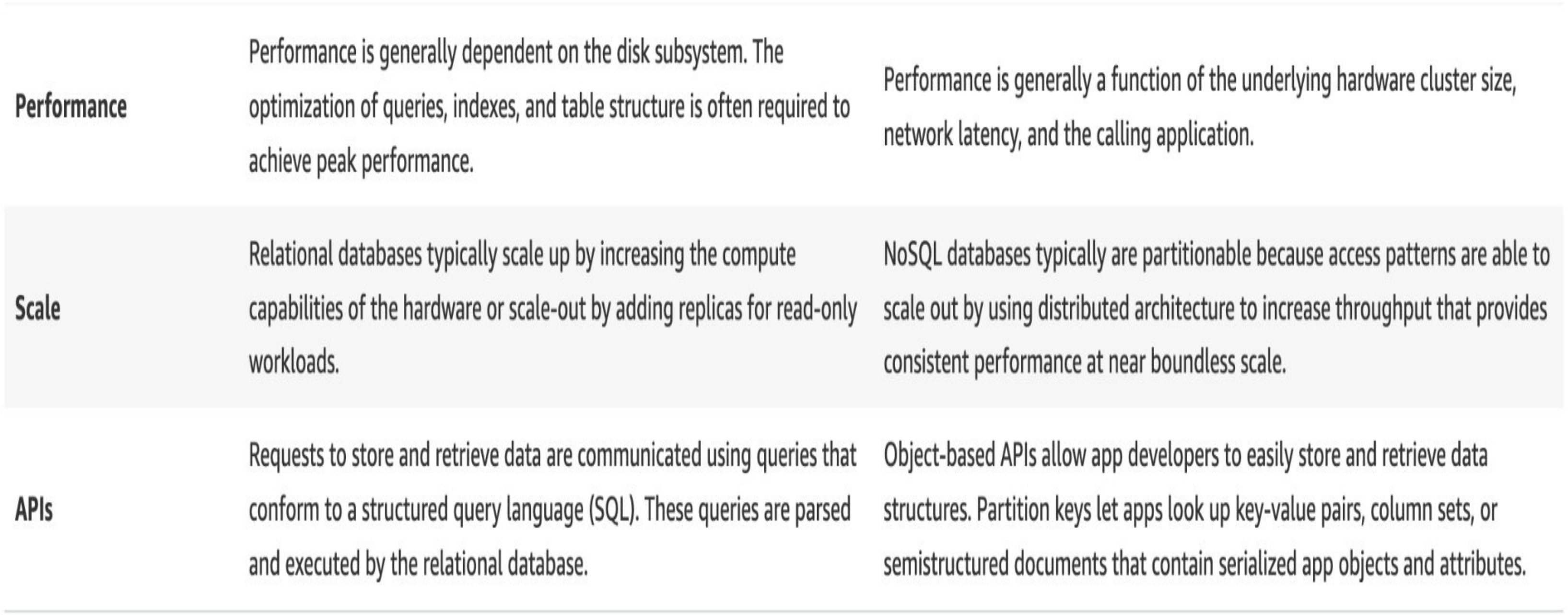

Friends
Person
LivesIn(address,...,...)
Likes (rating, review...)
Likes (rating, review...)
City
Restaurant
LocatedIn(address,...,...)

- Use Cases: Graph based database mostly used for social networks, logistics, spatial data, recommendation engines, fraud detection, and knowledge graphs.
- Challenges: Graph databases are less useful for operational use cases as they are not efficient at processing high volumes of transactions and are bad at handling queries over the entire database.
- Examples: Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases,
- Amazon Neptune is a fully-managed graph database service

NoSQL vs RDBMS



	Relational databases	NoSQL databases
Data model	The relational model normalizes data into tables that are composed of rows and columns. A schema strictly defines the tables, rows, columns, indexes, relationships between tables, and other database elements. The database enforces the referential integrity in relationships between tables.	NoSQL databases provide a variety of data models such as key-value, document, and graph, which are optimized for performance and scale.
ACID properties	Relational databases provide atomicity, consistency, isolation, and durability (ACID) properties: • Atomicity requires a transaction to execute completely or not at all. • Consistency requires that when a transaction has been committed, the data must conform to the database schema. • Isolation requires that concurrent transactions execute separately from each other. • Durability requires the ability to recover from an unexpected system failure or power outage to the last known state.	NoSQL databases often make tradeoffs by relaxing some of the ACID properties of relational databases for a more flexible data model that can scale horizontally. This makes NoSQL databases an excellent choice for high throughput, low-latency use cases that need to scale horizontally beyond the limitations of a single instance.



Performance	Performance is generally dependent on the disk subsystem. The optimization of queries, indexes, and table structure is often required to achieve peak performance.	Performance is generally a function of the underlying hardware cluster size, network latency, and the calling application.
Scale	Relational databases typically scale up by increasing the compute capabilities of the hardware or scale-out by adding replicas for read-only workloads.	NoSQL databases typically are partitionable because access patterns are able to scale out by using distributed architecture to increase throughput that provides consistent performance at near boundless scale.
APIs	Requests to store and retrieve data are communicated using queries that conform to a structured query language (SQL). These queries are parsed and executed by the relational database.	Object-based APIs allow app developers to easily store and retrieve data structures. Partition keys let apps look up key-value pairs, column sets, or semistructured documents that contain serialized app objects and attributes.

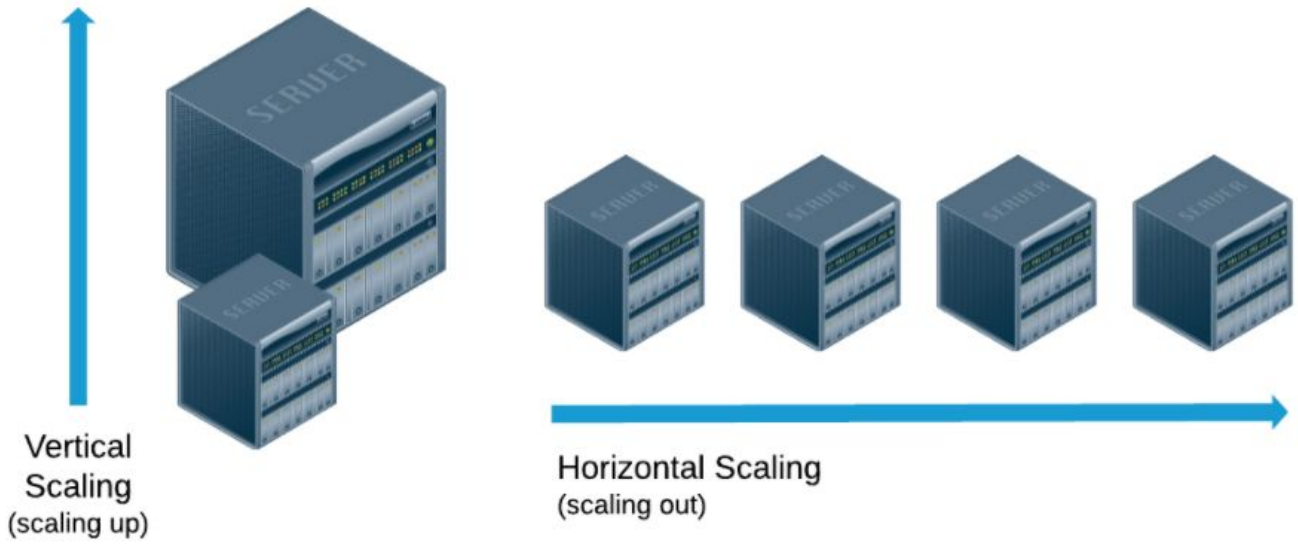
Brief discussion on Vertical Scaling, Horizontal Scaling, Sharding and Partitioning

- What is scalability?
  The scalability of an application can be measured by the number of requests it can effectively support simultaneously. The point at which an application is no longer able to handle additional requests effectively is the limit of its scalability. This limit is reached when a critical hardware resource runs out, requiring different or more machines.
- This is where we need to scale these resources.
- Scaling resources can include any combination of adjustments to CPU and physical memory (different or more machines), hard disk (bigger hard drives, less “live” data, solid state drives), and/or the network bandwidth (multiple network interface controllers, bigger NICs, fiber, etc.).
- Scaling horizontally and scaling vertically are similar in a certain way as they both involve adding computing resources to your infrastructure. There are distinct differences between the two in terms of implementation and performance.

Horizontal and Vertical Scaling

- Horizontal scaling means scaling by adding more machines to your existing infrastructure of resources (also described as “scaling out”).
- Vertical scaling refers to scaling by adding more power (e.g. CPU, RAM) to an existing machine (also described as “scaling up”).

- What’s the main difference?
  One of the fundamental differences between the two is that horizontal scaling requires breaking a sequential piece of logic into smaller pieces so that they can be executed in parallel across multiple machines. In many respects, vertical scaling is easier because the logic really doesn’t need to change. Rather, you’re just running the same code on higher-spec machines. However, there are many other factors to consider when determining the appropriate approach.


SERVER
Vertical Scaling (scaling up)
Horizontal Scaling (scaling out)

Sharding

In many large-scale applications, data is divided into partitions that can be accessed separately. There are two typical strategies for partitioning data.

Vertical partitioning: it means some columns are moved to new tables. Each table contains the same number of rows but fewer columns.

Horizontal partitioning (often called sharding): it divides a table into multiple smaller tables. Each table is a separate data store, and it contains the same number of columns, but fewer rows.

Sharding is the practice of optimizing database management systems by separating the rows or columns of a larger database table into multiple smaller tables.

Why Is Sharding Used?

Sharding is a common concept in scalable database architectures. By sharding a larger table, you can store the new chunks of data, called logical shards, across multiple nodes to achieve horizontal scalability and improved performance. Also, sharded databases can offer higher levels of availability. In the event of an outage on an unsharded database, the entire application is unusable. With a sharded database, only the portions of the application that relied on the missing chunks of data are unusable. In practice, sharded databases often further mitigate the impact of such outages by replicating backup shards on additional nodes.

When running a database on a single machine, you will eventually reach the limit of the amount of computing resources you can apply to any queries, and you will obviously reach a maximum amount of data with which you can efficiently work. By horizontally scaling out, you can enable a flexible database design that increases performance in two key ways:

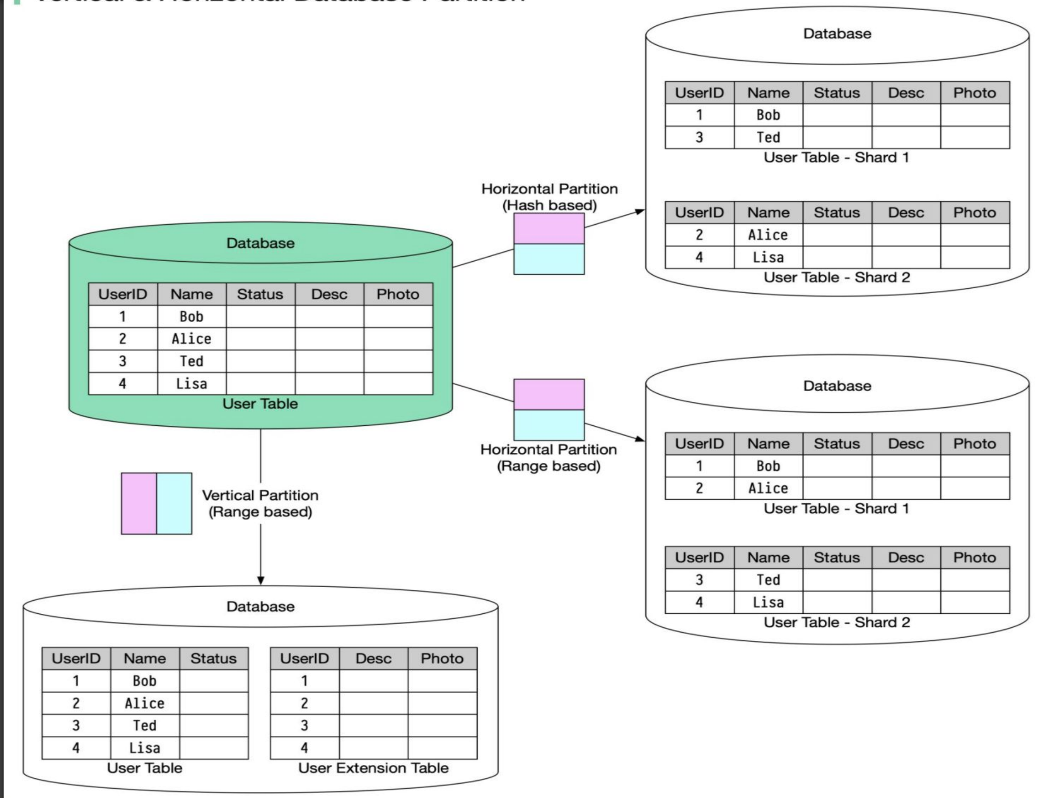
Horizontal and Vertical Sharding

- With massively parallel processing, you can take advantage of all the compute resources across your cluster for every query.
- Because the individual shards are smaller than the logical table as a whole, each machine has to scan fewer rows when responding to a query.

Horizontal sharding is effective when queries tend to return a subset of rows that are often grouped together. For example, queries that filter data based on short date ranges are ideal for horizontal sharding since the date range will necessarily limit querying to only a subset of the servers.

Vertical sharding is effective when queries tend to return only a subset of columns of the data. For example, if some queries request only names, and others request only addresses, then the names and addresses can be sharded onto separate servers.

Vertical & Horizontal Database Partition



Database
Horizontal Partition (Hash based)
Horizontal Partition (Range based)
Vertical Partition (Range based)
Database
Database
Database

UserID	Name	Status	Desc	Photo
1	Bob			
2	Alice			
3	Ted			
4	Lisa			

User Table

UserID	Name	Status	Desc	Photo
1	Bob			
3	Ted			

User Table - Shard 1

UserID	Name	Status	Desc	Photo
2	Alice			
4	Lisa			

User Table - Shard 2

UserID	Name	Status	Desc	Photo
1	Bob			
2	Alice			

User Table - Shard 1

UserID	Name	Status	Desc	Photo
3	Ted			
4	Lisa			

User Table - Shard 2

UserID	Name	Status
1	Bob	
2	Alice	
3	Ted	
4	Lisa	

User Table

UserID	Desc	Photo
1		
2		
3		
4		

User Extension Table

Why is NoSQL a good fit in this Project!

NoSQL is becoming increasingly popular in today's times. It is the go-to choice for a database for applications which are focused on handling massive scale. To appreciate the usefulness of NoSQL DBs, let's first understand the limitations of relational DBs.

1. Maintenance Problem

   The maintenance of the relational database becomes difficult over time due to the increase in the data. Developers and programmers have to spend a lot of time maintaining the database.

2. Lack of Scalability

   While using the relational database over multiple servers, its structure changes and becomes difficult to handle, especially when the quantity of the data is large. Due to this, the data is not scalable on different physical storage servers. Ultimately, its performance is affected i.e. lack of availability of data and load time etc. As the database becomes larger or more distributed with a greater number of servers, this will have negative effects like latency and availability issues affecting overall performance.

Note : The above point in no way means that relational databases are completely incompetent on handling scale. They are various optimisations and improvisations on managing increased loads and scale. The point is that they are not inherently accustomed to handle scale as well as the NoSQL databases.

1. Fixed Schema
   Arguably, this is the biggest concern with relational databases which makes NoSQL a winner for handling flexible, non-structured data. This kind of input is the most common form of input when we are handling real-time applications which are serving a considerable amount of traffic.

Why NoSQL?

- Simple to implement than using RDBMS.
- Offers a flexible schema design which can easily be altered without downtime or service disruption

- Provisional for managing and handling scale. Supports Horizontal Partitioning/Sharding.
- Excels at distributed database and multi-data center operations

Popular choices for NoSQL DB

1. MongoDB
   MongoDB is the most widely used document-based database. It stores the documents in JSON objects According to the website stackshare.io, more than 3400 companies are using MongoDB in their tech stack. Uber, Google, eBay, Nokia, Coinbase are some of them.

   When to use MongoDB?
   - In case you are planning to integrate hundreds of different data sources, the document-based model of MongoDB will be a great fit as it will provide a single unified view of the data.
   - When you are expecting a lot of reads and write operations from your application but you do not care much about some of the data being lost in the server crash.
   - You can use it to store clickstream data and use it for the customer behavioral analysis.

Cassandra

Cassandra is an open-source, distributed database system that was initially built by Facebook (and motivated by Google's Big Table). It is widely available and quite scalable. It can handle petabytes of information and thousands of concurrent requests per second.
Again, according to stackshare.io, more than 400 companies are using Cassandra in their tech stack. Facebook, Instagram, Netflix, Spotify, Coursera are some of them.

When to use Cassandra?

- When your use case requires more writing operations than reading ones.
- In situations where you need more availability than consistency. For example, you can use it for social network websites but cannot use it for banking purposes.
- You require less number of joins and aggregations in your queries to the database.
- Health trackers, weather data, tracking of orders, and time series data are some good use cases where you can use Cassandra databases.

3 ElasticSearch

This is also an open-source, distributed NoSQL database system. It is highly scalable and consistent. It can easily analyze, store, and search huge volumes of data.
More than 3000 companies are using Elasticsearch in their tech stack, including Slack, Udemy, Medium, and Stackoverflow.

When to use ElasticSearch?

- If your use case requires a full-text search, Elasticsearch will be the best fit.
- If your use case involves chatbots where these bots resolve most of the queries, such as when a person types something there are high chances of spelling mistakes. You can make use of the in-built fuzzy matching practices of the ElasticSearch
- ElasticSearch is useful in storing logs data and analyzing it.

4. Amazon DynamoDB

It is a key-value pair based distributed database system created by Amazon and is highly scalable. But unfortunately, it is not open-source. It can easily handle 10 trillion requests per day so you can see why!
More than 700 companies are using DynamoDB in their tech stack including Snapchat, Lyft, and Samsung.

When to use DynamoDB?

- In case you are looking for a database that can handle simple key-value queries but those queries are very large in number.
- In case you are working with OLTP workload like online ticket booking or banking where the data needs to be highly consistent

5. HBase

It is also an open-source highly scalable distributive database system. HBase was written in JAVA and runs on top of the Hadoop Distributed File System (HDFS).
More than 70 companies are using Hbase in their tech stack, such as Hike, Pinterest, and HubSpot.

When to use HBase?

- You should have at least petabytes of data to be processed. If your data volume is small, then you will not get the desired results
- If your use case requires random and real-time access to the data, then HBase will be the appropriate option.
- If you want to easily store real-time messages for billions of people

Intro to MongoDB

MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. MongoDB is a database which came into light around the mid-2000s.

MongoDB Features

1. Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.

1. The rows (or documents as called in MongoDB) doesn’t need to have a schema defined beforehand. Instead, the fields can be created on the fly.

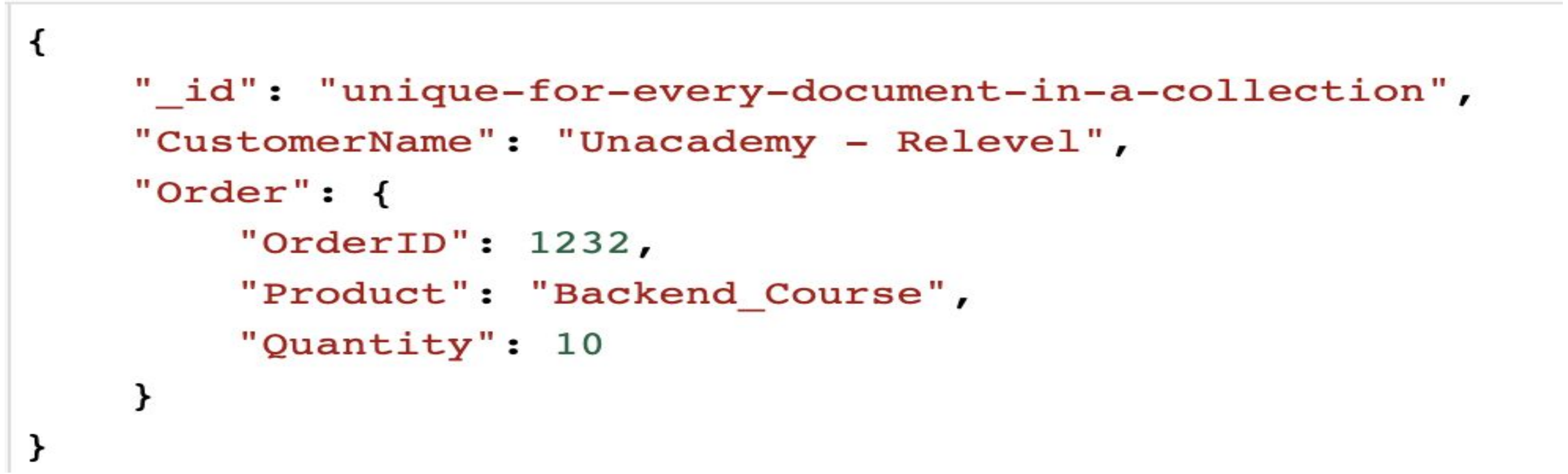
1. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.

1. Scalability – The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database

MongoDB Example

The below example shows how a document can be modeled in MongoDB.

1. The _id field is added by MongoDB to uniquely identify the document in the collection.
2. What you can note is that the Order Data (OrderID, Product, and Quantity) which in RDBMS will normally be stored in a separate table, while in MongoDB it is actually stored as an embedded document in the collection itself. This is one of the key differences in how data is modeled in MongoDB.



```
{
    "_id": "unique-for-every-document-in-a-collection",
    "CustomerName": "Unacademy - Relevel",
    "Order": {
        "OrderID": 1232,
        "Product": "Backend_Course",
        "Quantity": 10
    }
}
```

Key Components of MongoDB Architecture

Below are a few of the common terms used in MongoDB

1. **_id** – This is a field required in every MongoDB document. The _id field represents a unique value in the MongoDB document. The _id field is like the document's primary key. If you create a new document without an _id field, MongoDB will automatically create the field. Mongo DB will add a 24 digit unique identifier to each document in the collection.

2. Collection – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL. A collection exists within a single database. As seen from the introduction, collections don't enforce any sort of structure.

3. Cursor – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.

4. Database – This is a container for collections like in RDBMS where it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.

5. Document – A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.

6. Field – A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.The following diagram shows an example of Fields with Key value pairs. So in the example below OrderID and 1232 is one of the key value pairs defined in the document.

7. JSON – This is known as Javascript Object Notation. This is a human-readable, plain text format for expressing structured data. JSON is currently supported in many programming languages.

Just a quick note on the key difference between the _id field and a normal collection field. The _id field is used to uniquely identify the documents in a collection and is automatically added by MongoDB when the collection is created.

Why Use MongoDB?

Below are the few of the reasons as to why one should start using MongoDB

1. Document-oriented – Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situation and requirements.
2. Ad hoc queries – MongoDB supports searching by field, range queries, and regular expression searches. Queries can be made to return specific fields within documents.
3. Indexing – Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.
4. Replication – MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.
5. Load balancing – MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

THANK YOU