BENEFITS AND LIMITATIONS OF RELATIONAL DATABASES

Benefits of RDBMS

1) **Getting at Persistent Data**

- Database is used for keeping large amounts of persistent data.
- Most computer architectures have two areas of memory: a fast volatile “main memory” and a larger but slower “backing store.”
- Main memory is both limited in space and loses all data when you lose power or something bad happens to the operating system.
- Therefore, to save data, we write it to a backing store, which commonly is a disk.
- The backing store can be organized in different ways. It can be a file in the file system of the operating system(Student, Employee file). For most enterprise applications, however, the backing store is a database. The database allows more flexibility than a file system in storing large amounts of data.

Benefits of RDBMS contd..

2) Concurrency

- Enterprise applications have many people working on the same body of data at once, possibly modifying that data.
- Most of the time they are working on different areas of that data, but occasionally they operate on the same bit of data.
- As a result, we have to worry about coordinating these interactions to avoid issues like double booking of hotel rooms.
- Concurrency is very difficult to achieve, with all sorts of errors that can trap even the most careful programmers.
- Since enterprise applications can have lots of users and other systems all working concurrently, there's a lot of room for bad things to happen.
- Relational databases help handle this by controlling all access to their data through transactions.
- The transactional mechanism has worked well .Transactions also play a role in error handling. With transactions, you can make a change, and if an error occurs during the processing of the change you can roll back the transaction to previous stable state.

Benefits of RDBMS contd..

3) Integration

- Enterprise applications requires multiple applications, written by different teams, to collaborate in order to get things done.
- Applications often need to use the same data and updates made through one application have to be visible to others.
- A common way to do this is shared database integration where multiple applications store their data in a single database.
- Using a single database allows all the applications to use each others' data easily, while the database's concurrency control handles multiple applications in the same way as it handles multiple users in a single application

Limitations of Relational databases

- Challenges of traditional relational databases in handling unstructured and semi-structured data: Relational databases are not well-suited for handling unstructured and semi-structured data, which is increasingly common in modern applications and systems.

- Scalability and performance issues with big data and real-time applications: Relational databases can struggle with the scalability and performance demands of big data and real-time applications, leading to potential bottlenecks and performance issues.

NON-RELATIONAL DATABASES

Definition

History

Need for NoSQL

Key features

Difference between SQL and NoSQL

What is NoSQL database?

- NoSQL databases, also known as "not only SQL" are non-tabular databases and store data differently than relational tables.
- The term “NoSQL” originally referred to “non-SQL” or “non-relational” databases.

  but the term has evolved and now known as “not only SQL,” as NoSQL databases have

  expanded and come in a variety of types based on their data models.
- A type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data.
- Unlike traditional relational databases that use tables with pre-defined schemas to store data, NoSQL databases use flexible data models that can adapt to changes in data structures and are capable of scaling horizontally to handle growing amounts of data.
- They provide flexible schemas and scale easily with large amounts of data and high user loads.
- They are generally classified into four main categories: document, key-value, wide-column, and graph.

History of NoSQL databases?

I) Decrease in storage cost

- NoSQL databases emerged in the late 2000s as the cost of storage dramatically decreased.
- No need to create a complex, difficult-to-manage data model in order to avoid data duplication.
- Developers (rather than storage) were becoming the primary cost of software development, so NoSQL databases optimized for developer productivity.
- As storage costs rapidly decreased, the amount of data that applications needed to store and query increased.This data came in all shapes and sizes — structured, semi- structured, and polymorphic — and defining the schema in advance became nearly impossible.
- NoSQL databases allow developers to store huge amounts of unstructured data, giving them a lot of flexibility.

History of NoSQL databases contd..

II) Popularity of Agile Software Development

- Software developers realized the need to rapidly adapt to changing requirements. They needed the ability to iterate quickly and make changes throughout their software stack — all the way down to the database.
- NoSQL databases gave them this flexibility.

III) Popularity of Cloud Computing

- Developers started using public clouds to host their applications and data.
- They wanted the ability to distribute data across multiple servers and regions to make their applications resilient, to scale out instead of scale up, and to intelligently geo-place their data.
- Some NoSQL databases like MongoDB provide these capabilities.

History of NoSQL databases contd..

Pls Note : Scale up vs Scale
out Scale up(Vertical
scaling)
Upgrading the hardware, to handle higher
computing load. Scale out(Horizontal scaling)
Moving to a distributed architecture and adding more computers to solve our
problem.

Why NoSQL databases? / Need for NoSQL

Here are five reasons why organizations may choose to use a NoSQL database:

1. Scalability:
   - NoSQL databases are designed to scale horizontally, meaning that they can handle large amounts of data and user traffic by adding more commodity hardware.
   - This makes it easier to handle the increasing demands of modern applications, without having to make significant changes to the underlying infrastructure.
2. Performance:
   - NoSQL databases are optimized for handling large volumes of data, which means that they can deliver faster performance compared to traditional relational databases.
   - This is particularly important for applications that require real-time data access and low latency.
3. Flexibility:
   - NoSQL databases are schema-less, meaning that they can handle unstructured and semi-structured data in addition to structured data.
   - This flexibility makes it easier to store and manage a wide variety of data types, including documents, graphs, and key-value pairs.

Why NoSQL databases? / Need for NoSQL

4. Cost-effectiveness:

Because NoSQL databases are designed to run on commodity hardware, they are often more cost- effective than traditional relational databases, which can require expensive hardware and software licenses.

5. Availability:

- NoSQL databases are designed to handle high levels of traffic and data throughput, which means that they can provide high availability and fault tolerance.
- This is particularly important for mission-critical applications that require constant uptime and minimal downtime.

Key Features of NoSQL databases

1. Flexible data models: NOSQL databases support various data models, such as document, key-value, wide-column, and graph, providing flexibility in data representation.

2. Horizontal scalability: NOSQL databases can scale horizontally by adding more servers or nodes to the database infrastructure, allowing them to handle increasing data volumes and user loads.

3. Distributed architectures: NOSQL databases are designed with distributed architectures, enabling them to distribute data across multiple nodes or servers for improved performance and fault tolerance.

Key Features of NoSQL databases

4. High-performance real-time data processing: NOSQL databases are optimized for high-performance real-time data processing, making them suitable for applications that require fast data retrieval and processing.

Difference between SQL and NoSQL

Characteristic	SQL Databases	NoSQL Databases
Category	Relational Databases	Non-relational or distributed databases
Data Storage Model	Tables with fixed rows and columns	Document: JSON documents, Key-value: key-value pairs, Wide-column: tables with rows and dynamic columns, Graph: nodes and edges
Type of data	Structured	Structured as well as unstructured data
Development History	Developed in the 1970s with a focus on reducing data duplication	Developed in the late 2000s with a focus on scaling and allowing for rapid application change driven by agile and DevOps practices.
Examples	Oracle, MySQL, Microsoft SQL Server, and PostgreSQL	Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Wide-column: Cassandra and HBase, Graph: Neo4j and Amazon Neptune

Difference between SQL and NoSQL

Characteristic	SQL Databases	NoSQL Databases
Primary Purpose	General purpose	Document: general purpose, Key- value: large amounts of data with simple lookup queries, Wide- column: large amounts of data with predictable query patterns, Graph: analyzing and traversing relationships between connected data
Schemas	Rigid	Flexible
Scaling	Vertical (scale-up with a larger server) Excess of load can be managed by increasing the CPU, SSD, RAM,, etc., on the server.	Horizontal (scale-out across commodity servers) Addition of more servers to ease the load management
Multi-Record ACID Transactions	Supported	Most do not support multi-record ACID transactions. However, some — like MongoDB — do.

Difference between SQL and NoSQL

Characteristic	SQL Databases	NoSQL Databases
Joins	Typically required	Typically not required
Data to Object Mapping	Requires ORM (object-relational mapping)	Many do not require ORMs. MongoDB documents map directly to data structures in most popular programming languages.

Thank you

ACID

- It ensures transaction management in databases.
- HighlightS the importance of ensuring data integrity, consistency, and reliability. It Ensures DB transactions as single unit.

ACID PRINCIPLES

1. **Atomicity:** Transactions are treated as indivisible units, ensuring that either all changes are committed or none at all.
2. **Consistency:** Transactions bring the database from one valid state to another, maintaining data integrity and adhering to defined constraints.
3. **Isolation:** Concurrent execution of transactions occurs in a way that each transaction appears to be executed in isolation, preventing interference.
4. **Durability:** Once a transaction is committed, its effects are permanent, and changes persist even in the event of system failures

BASE MODELS

- **Basically Available:** The system remains operational and responsive even in the face of failures, prioritizing availability.
- **Soft state:** Acknowledges that the system's state can change over time,

so even during times without input there may be changes going on due to ‘eventual consistency,’ thus the state of the system is always ‘soft.’

- **Eventually Consistent:** The system will *eventually* become consistent once it stops receiving input.

At the transaction level, however, consistency is not ensured.The data will propagate to everywhere it should sooner or later, but the system will continue to receive input and is not checking the consistency of every transaction before it moves onto the next one.

ACID vs BASE

Aspect	ACID	BASE
Consistency Model	Strong consistency	Eventual consistency
Priority	Data integrity and consistency	Availability and partition tolerance
Use Cases	Financial applications, systems requiring data integrity	Distributed systems, high-availability requirements
Database Type	Traditional relational databases (e.g., MySQL, PostgreSQL)	NoSQL databases (e.g., Cassandra), distributed systems
Trade-offs	May sacrifice availability for strong consistency	May sacrifice immediate consistency for availability and partition tolerance

Advantages of NoSQL

1) Flexible data models

NoSQL databases typically have very flexible schemas. A flexible schema allows you to easily make changes to your database as requirements change.You can iterate quickly and continuously integrate new application features to provide value to your users faster.

2) Horizontal scaling

Most SQL databases require you to scale-up vertically (migrate to a larger, more expensive server) when we exceed the capacity requirements of current server. Conversely, most NoSQL databases allow to scale-out horizontally, meaning we can add cheaper commodity servers whenever needed.

3)Fast queries

Queries in NoSQL databases can be faster than SQL databases. Data in SQL databases is typically normalized, so queries for a single object or entity require you to join data from multiple tables.

As our tables grow in size, the joins can become expensive. However, data in NoSQL databases is typically stored in a way that is optimized for queries.The rule of thumb when we use MongoDB is data that is accessed together should be stored together. Queries typically do not require joins, so the queries are very fast.

4) Easy for developers

Some NoSQL databases like MongoDB map their data structures to those of popular programming languages. This mapping allows developers to store their data in the same way that they use it in their application code. This mapping can allow developers to write less code, leading to faster development time and fewer bugs.

Disadvantages of NoSQL

1) They don’t support ACID (atomicity, consistency, isolation, durability) transactions across multiple documents.
2) Since data models in NoSQL databases are typically optimized for queries and not for reducing data duplication, NoSQL databases can be larger than SQL databases.

Storage is currently so cheap that most consider this a minor drawback, and some NoSQL databases also support compression to reduce the storage footprint.

Types of NoSQL Databases

Document type databases: Here, the key gets paired with a compound data structure, i.e., document. MongoDB is an example of such type.
Key-Value stores: Here, each unstructured data is stored with a key for recognizing it.
Graph stores: In this type of database, data is stored mostly for networked data. It helps to relate data based on some existing data.
Wide-column stores:This type of data stores large data sets Cassandra (Used by Facebook), HBase are examples of such type.

Types of NoSQL Databases

1) A Document Data Model is different than other data models because it stores data in JSON, or XML documents.
In this data model, we can move documents under one document .
Also, any particular elements can be indexed to run queries faster.
Often documents are stored and retrieved in such a way that it becomes close to the data objects which are used in many applications so that very less translations are required to use data in applications.
JSON is a native language that is often used to store and query data too.

So in the document data model, each document has a key-value pair below is an example for the same.

```
{ "Name" : "Yashodhra",
"Address" : "Near Patel Nagar",
"Email" : "yahoo123@yahoo.com",
"Contact" : "12345" }
```

Working of Document Data Model:

It works as a semi-structured data model in which the records and data associated with them are stored in a single document which means this data model is not completely unstructured.
The main thing is that data here is stored in a document. Features:
Document Type Model: As data is stored in documents rather than tables or graphs, it becomes easy to map things in many programming languages.
Flexible Schema: Overall schema is very much flexible. Not all documents in a collection need to have the same fields.
Distributed and Resilient: Document data models are very much dispersed which is the reason behind horizontal scaling and distribution of data.
Manageable Query Language: These data models are the ones in which query language allows the developers to perform CRUD (Create Read Update Destroy) operations on the data model.

Examples

Amazon
DocumentDB
MongoDB
Cosmos DB
ArangoDB
Couchbase
Server CouchDB

Advantages:

Schema-less: These are very good in retaining existing data at massive volumes because there are absolutely no restrictions in the format and the structure of data storage.

Faster creation of document and maintenance: It is very simple to create a document and apart from this maintenance requires is almost nothing.

Open formats: It has a very simple build process that uses XML, JSON, and its other forms.

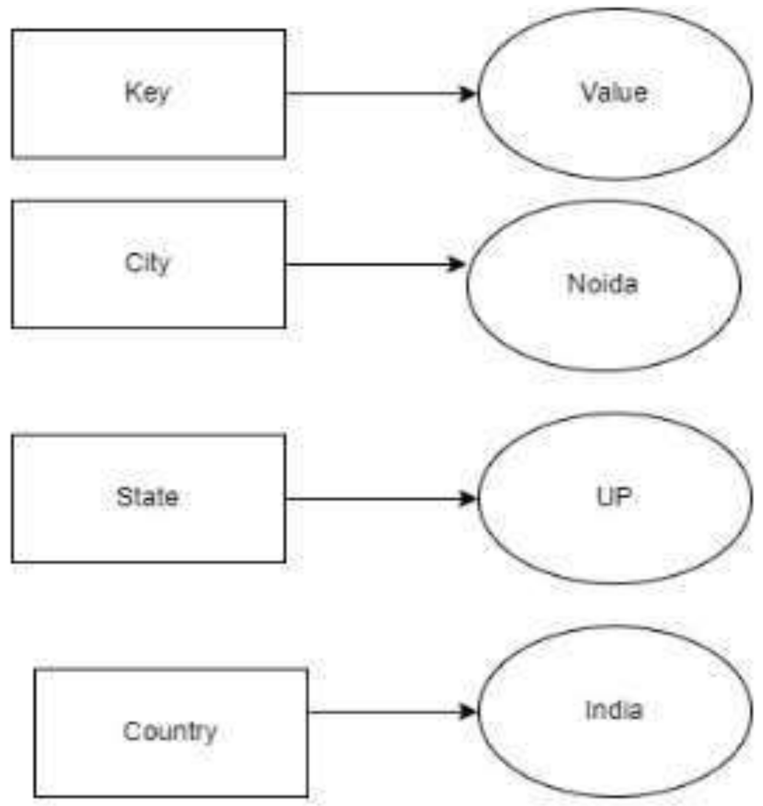
Built-in versioning: It has built-in versioning which means as the documents grow in size there might be a chance they can grow in complexity.Versioning decreases conflicts.

Disadvantages

Weak Atomicity: It lacks in supporting multi-document ACID transactions. A change in the document data model involving two collections will require us to run two separate queries i.e. one for each collection.This is where it breaks atomicity requirements.
Consistency Check Limitations: One can search the collections and documents that are not connected to an author collection but doing this might create a problem in the performance of database performance.
Security: Nowadays many web applications lack security which in turn results in the leakage of sensitive data. So it becomes a point of concern, one must pay attention to web app vulnerabilities.

2) Key-Value Data Model in NoSQL

- Also known as key value stores, are NoSQL database types where data is stored as key value pairs and optimized for reading and writing that data.
- The data is fetched by a unique key or a number of unique keys to retrieve the associated value with each key.
- The values can be simple data types like strings and numbers or complex objects.
- The unique key can be anything. Most of the time, it is an id field, since that's the unique field in all the documents.

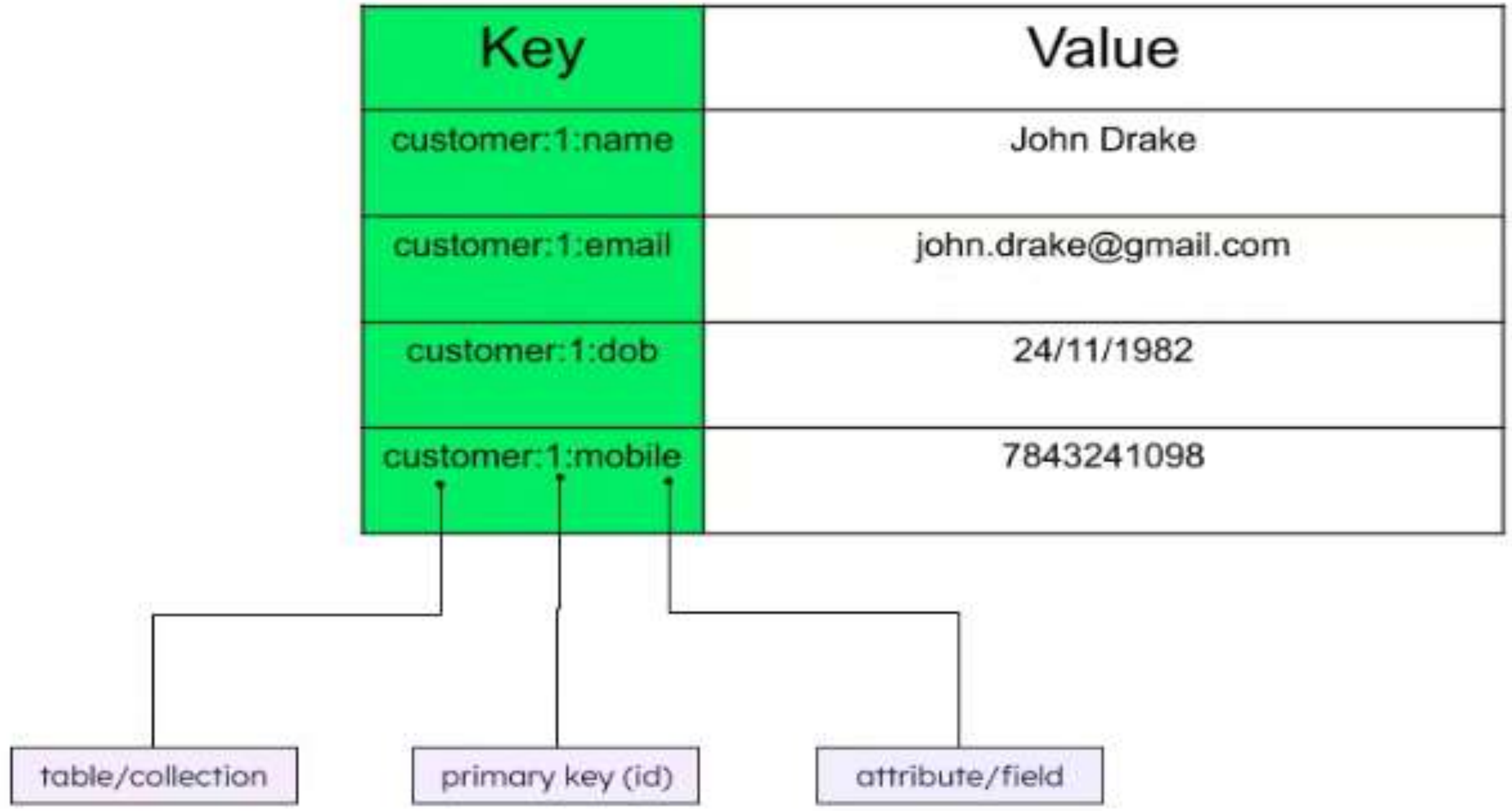


How do key-value databases work?

- A key value database associates a value (which can be anything from a number or simple string to a complex object) with a unique identifier (key), which is used to keep track of the object.
- In its simplest form, a key value store is like a dictionary/array/map object as it exists in most programming paradigms but is managed by a database management system (DBMS).
- There are three main operations performed by a key value database:
  1. put(key, value): insert or update a value into the database.
  2. get(key): read a value from the database.
  3. delete(key): delete a value from the database.

How do key-value databases work?

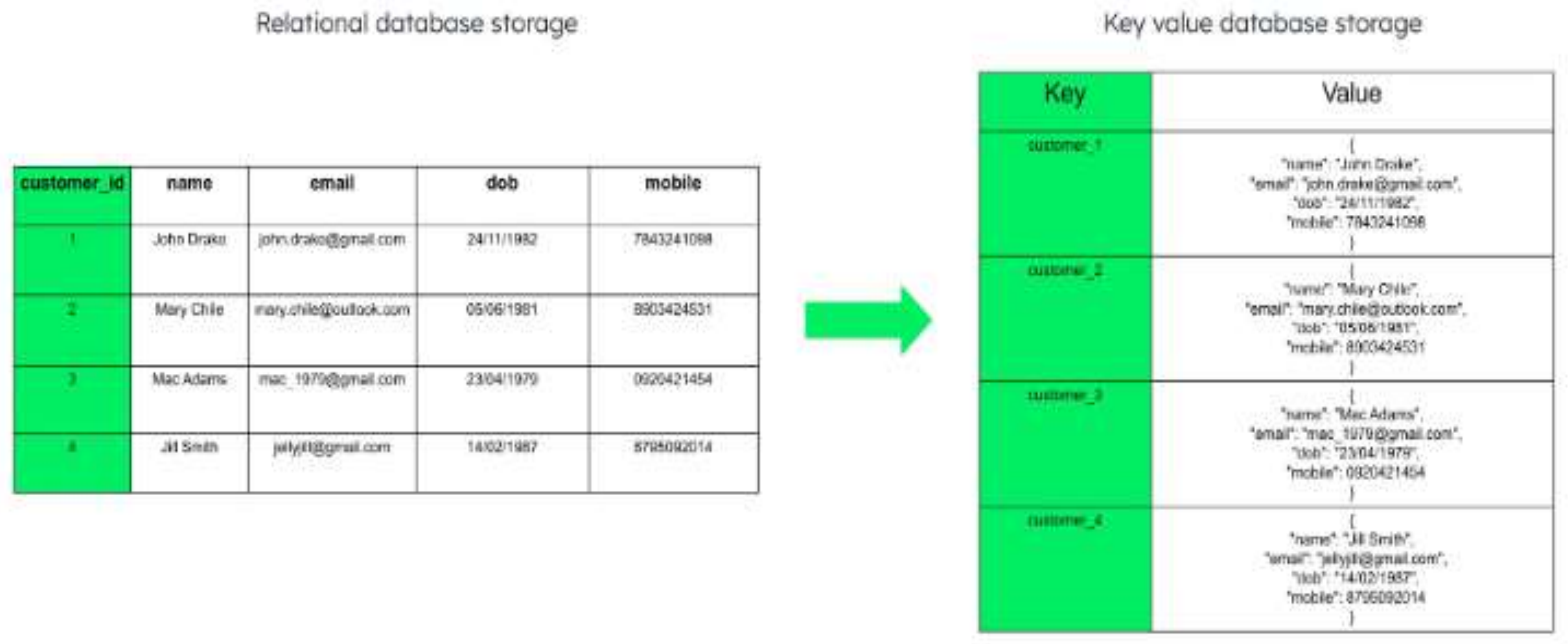
Key-value database storage



Key	Value
customer:1:name	John Drake
customer:1:email	john.drake@gmail.com
customer:1:dob	24/11/1982
customer:1:mobile	7843241098

key-value database schema

- Unlike relational databases, key value data stores do not follow a specific schema, making them flexible.
- This makes them a good choice for unstructured and semi structured data.
- Storing and retrieving data becomes much simpler when it is done in a key value store when compared to that of relational database table structure.
- In the below image, due to the JSON structure, all the data is grouped together.
- We can store anything from simple types to complex types as values. While fetching the data, you can retrieve the individual records by iterating through the JSON data.



customer_id	name	email	dob	mobile
1	John Drake	john.drake@gmail.com	24/11/1982	7843241098
2	Mary Chile	mary.chile@outlook.com	05/06/1981	8903424531
3	Mac Adams	mac_1979@gmail.com	23/04/1979	0920421454
4	Jill Smith	jellyjill@gmail.com	14/02/1987	8795092014

Key	Value
customer_1	{ "name": "John Drake", "email": "john.drake@gmail.com", "dob": "24/11/1982", "mobile": 7843241098 }
customer_2	{ "name": "Mary Chile", "email": "mary.chile@outlook.com", "dob": "05/06/1981", "mobile": 8903424531 }
customer_3	{ "name": "Mac Adams", "email": "mac_1979@gmail.com", "dob": "23/04/1979", "mobile": 0920421454 }
customer_4	{ "name": "Jill Smith", "email": "jellyjill@gmail.com", "dob": "14/02/1987", "mobile": 8795092014 }

When to use a key-value database?

- User session attributes in an online app like finance or gaming, which is referred to as real-time random data access.
- Caching mechanism for repeatedly accessing data or key-based design.
- The application is developed on queries that are based on keys.

Features:

- **Flexible data models**-do not have a fixed schema or table structure. as data evolves, it is easy to move key value stores from one system to another without any changes to the architecture or design
- **No query language**-As there is no fixed structure, key value databases do not need a query language for CRUD operations. You can simply give the key as input and get the values as output.
- **Support for complex data types**-Along with simple types like string or number, key value databases can also support data types like arrays, JSON, date, media files
- **Indexing support for performance**-Advanced key value databases support various types of indexing apart from primary indexing, like secondary indexing, hash indexing, tree-based indexing, and wild card indexing, to improve performance and efficiency of data retrieval.

Advantages:

- It is very easy to use due to the simplicity of the database
- Its response time is fast due to its simplicity
- Key-value store databases are scalable vertically as well as horizontally.

Disadvantages:

- As querying language is not present in key-value databases, transportation of queries from one database to a different database cannot be done.
- If application involves many-to-many relationships or requires joining datasets, key-value databases become cumbersome and inefficient.

Examples:

Couchbase: It permits SQL-style querying and searching for text.
Amazon DynamoDB: The key-value database which is mostly used is Amazon DynamoDB as it is a trusted database used by a large number of users. It can easily handle a large number of requests every day and it also provides various security options.
Riak: It is the database used to develop applications,etc

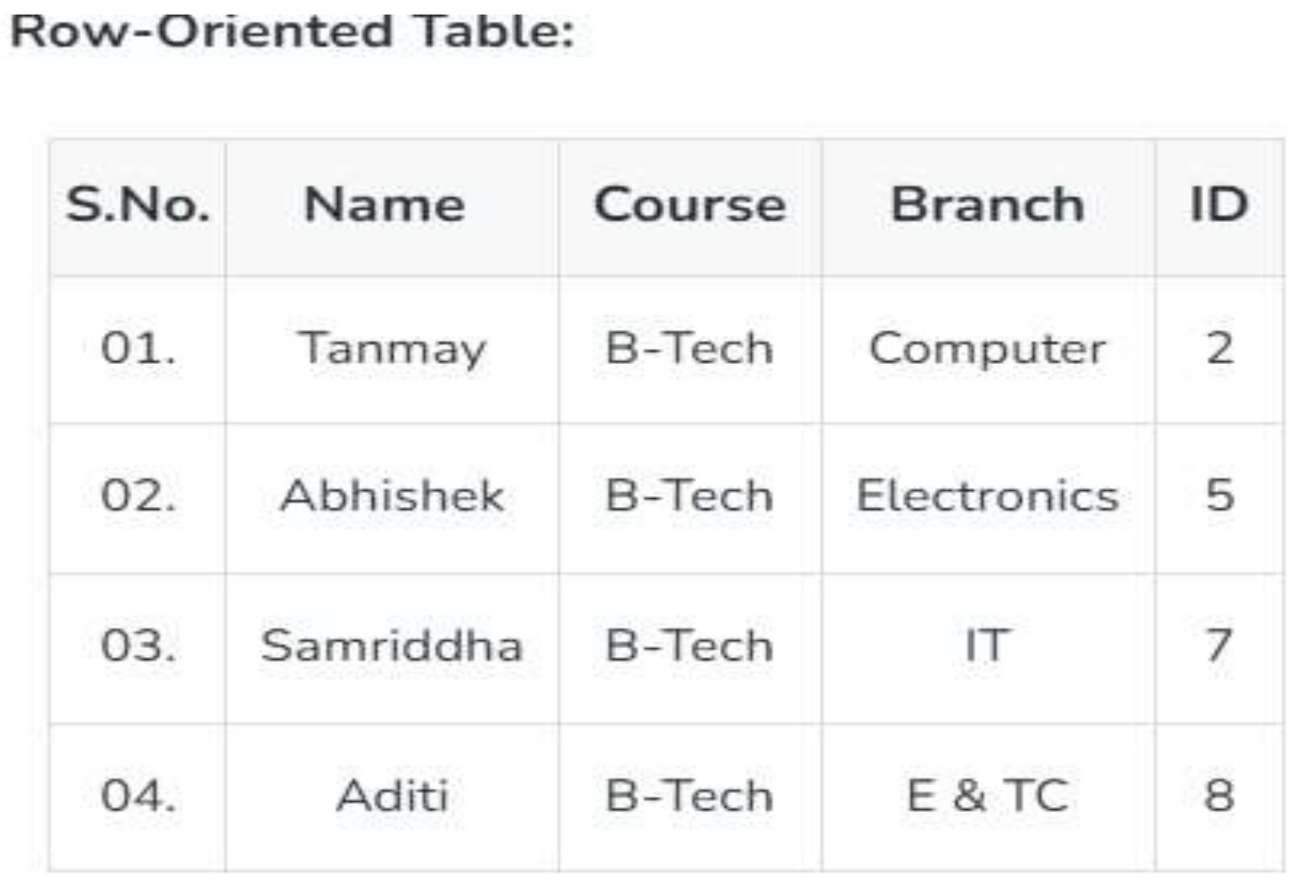
3) Columnar Database

- The relational database stores data in rows and also reads the data row by row.
- Column store is organized as a set of columns. So if someone wants to run analytics on a small number of columns, one can read those columns directly without consuming memory with the unwanted data.
- It is responsible for speeding up the time required to return a particular query.
- It also is responsible for greatly improving the disk I/O performance.
- It is helpful in data analytics and data warehousing and to effectively read and write data.
- Examples for Columnar Database are Apache Cassandra, SAP Hana, Monet DB, etc

Working of Columnar Data Model:

- In Columnar Data Model instead of organizing information into rows, it does in columns. This makes them function the same way that tables work in relational databases. This type of data model is much more flexible obviously because it is a type of NoSQL database.

Examples:

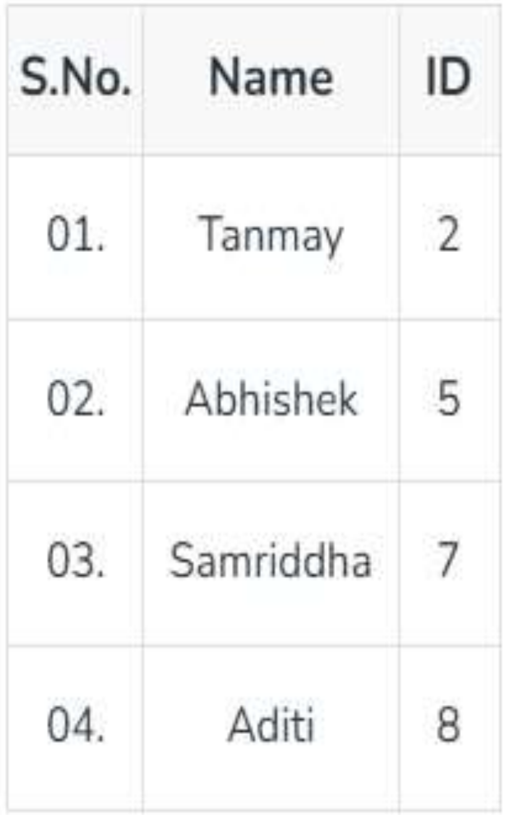


Row-Oriented Table:

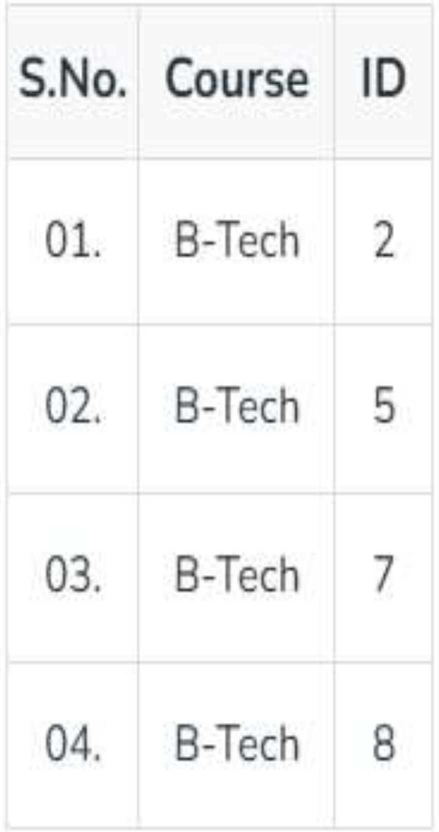
S.No.	Name	Course	Branch	ID
01.	Tanmay	B-Tech	Computer	2
02.	Abhishek	B-Tech	Electronics	5
03.	Samriddha	B-Tech	IT	7
04.	Aditi	B-Tech	E & TC	8

Examples:

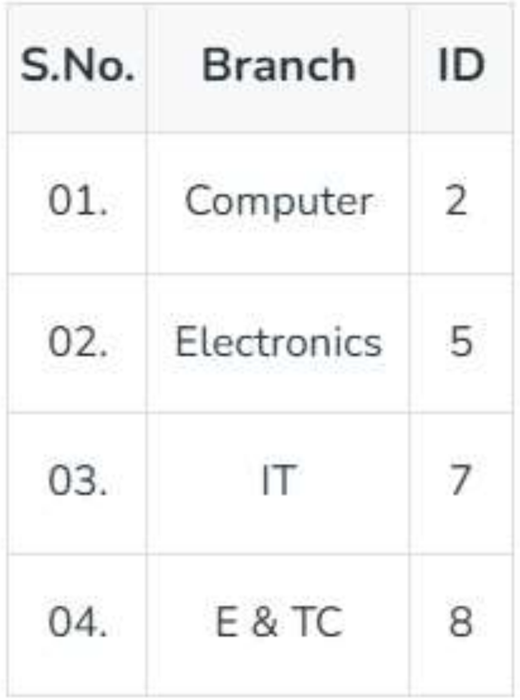
Column – Oriented Table:



S.No.	Name	ID
01.	Tanmay	2
02.	Abhishek	5
03.	Samriddha	7
04.	Aditi	8



S.No.	Course	ID
01.	B-Tech	2
02.	B-Tech	5
03.	B-Tech	7
04.	B-Tech	8



S.No.	Branch	ID
01.	Computer	2
02.	Electronics	5
03.	IT	7
04.	E & TC	8

When to use the Columnar Database:

- Queries that involve only a few columns.
- Compression but column-wise only.
- Clustering queries against a huge amount of data.

Advantages:

- For applications that are related to big data the column-oriented databases have greater attention.
- The data in the columnar database has a highly compressible nature and has different operations like (AVG), (MIN, MAX), which are permitted by the compression.
- Efficiency and Speed: The speed of Analytical queries that are performed is faster in columnar databases.
- Self-indexing: Another benefit of a column-based DBMS is self-indexing, which uses less disk space than a relational database management system containing the same data.

Limitations:

1. For loading incremental data, traditional databases are more relevant as compared to column-oriented databases.
2. For Online transaction processing (OLTP) applications, Row oriented databases are more appropriate than columnar databases.

4) Graph Databases

A graph database is a type of NoSQL database that is designed to handle data with complex relationships and interconnections.

In a graph database, data is stored as nodes and edges, where nodes represent entities and edges represent the relationships between those entities.

1. Graph databases are particularly well-suited for applications that require complex queries, such as social networks, recommendation engines, and fraud detection systems. They can also be used for other types of applications, such as supply chain management, network and infrastructure management, and bioinformatics.
2. Main advantages of graph databases is their ability to handle and represent relationships between entities which often cannot be easily represented in a traditional relational database.

3. Another advantage of graph databases is their flexibility. Graph databases can handle data with changing structures and can be adapted to new use cases without requiring significant changes to the database schema.

4. Graph databases may not be suitable for all applications. For example, they may not be the best choice for applications that require simple queries or that deal primarily with data that can be easily represented in a traditional relational database.

In graph theory, we have terms like Nodes, edges, and properties.

- **Nodes:** These are the instances of data that represent objects which is to be tracked.
- **Edges:** As we already know edges represent relationships between nodes.
- **Properties:** It represents information associated with nodes.

The below image represents Nodes with properties from relationships represented by edges.

- In these data models, the nodes which are connected together are connected physically and the physical connection among them is also taken as a piece of data.
- Connecting data in this way becomes easy to query a relationship.
- This data model reads the relationship from storage directly instead of calculating and querying the connection.

Examples:

JanusGraph: These are very helpful in big data analytics. It is a scalable graph database system which is open source too.

Neo4j: It stands for Network Exploration and Optimization 4 Java. As the name suggests this graph database is written in Java with native graph storage and processing.

Advantages of Graph Data Model :

- No joins are required since relationships is already specified.
- **Efficient data modeling:** Graph databases allow for efficient data modeling by representing data as nodes and edges. This allows for more flexible and scalable data modeling than traditional relational databases.
- **Flexible relationships:** Graph databases are designed to handle complex relationships and interconnections between data elements.Well-suited for applications that require deep and complex queries, such as social networks, recommendation engines, and fraud detection systems.
- **High performance:** Graph databases are optimized for handling large and complex datasets
- **Scalability:** Graph databases can be easily scaled horizontally, allowing additional servers to be added to the cluster to handle increased data volume or traffic.

Disadvantages of Graph Data Model :

- No standard query language: Since the language depends on the platform that is used so there is no certain standard query language.
- Limited use cases: Graph databases are not suitable for all applications. They may not be the best choice for applications that require simple queries or that deal primarily with data that can be easily represented in a traditional relational database.
- They are inappropriate for transactional data.
- Often for complex relationships speed becomes slower in searching.

Applications of Graph Data Model:

Fraud detection system, Digital asset management, Network management.
Example of Graph Database:

- **Recommendation engines in E commerce** use graph databases to provide customers with accurate recommendations, updates about new products thus increasing sales and satisfying the customer’s desires.
- **Social media companies** use graph databases to find the “friends of friends” or products that the user’s friends like and send suggestions accordingly to user.

Thank you