Think and Answer

ACID properties are indeed relevant in the context of distributed computing. Let's break down what ACID stands for:

1. **Atomicity**: In a distributed system, atomicity ensures that a transaction is treated as a single, indivisible unit. Either all the operations within the transaction succeed, or none of them do. This prevents partial updates and maintains data consistency.

2. **Consistency**: Consistency guarantees that the database remains in a valid state after a transaction. In distributed systems, maintaining consistency across multiple nodes can be challenging due to network delays and failures. Techniques like two-phase commit (2PC) or consensus algorithms (e.g., Paxos, Raft) help achieve distributed consistency.

3. **Isolation**: Isolation ensures that concurrent transactions do not interfere with each other. In distributed databases, isolation levels (e.g., read committed, serializable) define how transactions interact. Implementing isolation correctly is crucial to prevent anomalies like dirty reads or lost updates.

4. **Durability**: Durability ensures that once a transaction is committed, its effects persist even in the face of system failures. Distributed systems achieve durability by replicating data across multiple nodes and ensuring that updates are safely stored on stable storage.

In summary, ACID properties play a vital role in maintaining data integrity and reliability, even in distributed environments.

The **CAP theorem**, also known as Brewer's theorem, is a fundamental concept in distributed computing. It states that in a distributed data store, you can achieve at most two out of three desirable properties:

1. **Consistency**: All clients see the same data at the same time, regardless of which node they connect to. To achieve consistency, data written to one node must be instantly forwarded or replicated to all other nodes before the write is considered successful.

2. **Availability**: Every client request for data receives a response, even if some nodes are down. In other words, all working nodes in the system provide valid responses without exceptions.

3. **Partition Tolerance**: A partition refers to a communication break within the distributed system—a lost or temporarily delayed connection between two nodes. Partition tolerance ensures that the cluster continues to function despite communication breakdowns between nodes.

Here's the catch: You can't have all three properties simultaneously. When designing a distributed system, you must prioritize based on your application's requirements. NoSQL databases, which are horizontally scalable and distributed by design, play a crucial role in achieving these characteristics[1][2][3].

Let's explore schemaless databases and their pros and cons:

1. **Schemaless Databases**:
   - A schemaless database manages information without a predefined blueprint. It doesn't enforce specific fields, tables, or data model structures.
   - Unlike relational databases, which adhere to strict schemas, schemaless databases allow any data type (e.g., key-value, document, in-memory, column-oriented, or graph).

- Popular in real-time data-driven industries like finance, gaming, and social media.
- Changes (adding/removing data types, tables, fields) are easier without complex schema migrations[12].

2. **Pros of Schemaless Databases**:
   - **Flexibility**: No fixed data structure; data remains unaltered and accessible.
   - **Scalability**: Easily handle large datasets.
   - **Efficient Migration**: No need for rigid schema planning; adapt as needed.
   - **Unstructured Data**: Ingest unstructured data without constraints.

3. **Cons of Schemaless Databases**:
   - **Query Complexity**: Lack of a universal language for querying non-relational data.
   - **No Predefined Schema**: No existing structure to guide data organization[12].

In summary, schemaless databases offer flexibility and scalability, but querying complexity and lack of a predefined schema are trade-offs to consider. Choose based on your specific use case!

Let's explore the characteristics of schemaless databases:

Flexibility over Data Types:

1. Schemaless databases allow you to store, retrieve, and query any data type. They're perfect for big data analytics and operations powered by unstructured data.
2. Unlike relational databases, which enforce rigid schemas, schemaless databases accept any data type, including those not yet defined. This future-proofs your database as your needs evolve12.

No Predefined Database Schemas:

1. Schemaless databases don't require conforming to a fixed schema throughout the system's life.
2. You're not limited by predefined data types, tables, or structures. Your NoSQL database can adapt as your data-driven operations change and mature3.

Raw Data Preservation:

1. Each item in a schemaless database is saved in its own document with a partial schema.
2. This means every detail is always available, and nothing is stripped to match the current schema. Raw information remains untouched1.

In summary, schemaless databases offer flexibility, adaptability, and the ability to handle both structured and unstructured data. They're commonly used in systems like MongoDB, CouchDB, and Google Cloud Datastore.

NoSQL databases use various data architecture patterns to store and manage data. Let's explore the four common patterns:

Key-Value Store Database:

1. Data is stored as key-value pairs.
2. Keys are typically strings, integers, or characters.
3. Values can be of any type (e.g., JSON, BLOB, strings).
4. Examples: DynamoDB, Berkeley DB.

Column Store Database:

1. Data is stored in individual cells grouped into columns.
2. Columns allow efficient aggregation operations (e.g., SUM, AVERAGE, COUNT).
3. Examples: HBase, Google Bigtable, Cassandra.

Document Database:

1. Data is stored as key-value pairs, but values are called "documents."
2. Documents can be complex structures (text, arrays, JSON, XML).
3. Commonly used for semi-structured data.
4. Examples: MongoDB, CouchDB.

Graph Databases:

1. Manage data in graphs (connections between objects).
2. Useful for modeling relationships.
3. Examples: Neo4j, Amazon Neptune.

Each pattern has its advantages and limitations, so choose based on your specific requirements!

Simple JSON file

```
{
   "name": "John",
   "age": 30,
   "car": null
}
```

JSON (JavaScript Object Notation) is a lightweight, human-readable format for data interchange. It's commonly used in web APIs, configuration files, and data storage.

Storing **JSON** (JavaScript Object Notation) data in distributed databases involves a few considerations. Let's explore how this works:

MongoDB:

1. MongoDB is a popular NoSQL database that natively supports JSON (or more specifically, BSON) data.
2. It allows you to store hierarchical data structures directly as documents.
3. MongoDB's dynamic schema lets you create records without defining a fixed structure upfront.
4. Data is partitioned across nodes, ensuring scalability and availability1.

PostgreSQL:

1. PostgreSQL, although a relational database, now offers enhanced JSON storage capabilities.
2. It supports both **JSON** and **JSONB** (binary JSON) formats.
3. JSONB allows efficient indexing and querying of JSON data.
4. Postgres provides data constraint functions, ensuring meaningful JSON documents1.

Azure Cosmos DB:

1. Azure Cosmos DB represents resources (like collections or documents) as JSON documents.
2. Its resource model is hierarchical, allowing navigation using hyperlinks2.

NoSQL Databases:

1. NoSQL databases, including MongoDB, store data within a single structure (e.g., JSON documents).
2. They handle large, unstructured datasets without rigid schemas3.

In summary, both MongoDB and PostgreSQL can effectively store JSON data in distributed environments.

Base Properties in DBMS

Pre-requisites: ACID Properties in DBMS

The BASE properties of a database management system are a set of principles that guide the design and operation of modern databases.

The acronym BASE stands for **Basically Available**, **Soft State**, and **Eventual Consistency.**

Basically Available
This property refers to the fact that the database system should always be available to respond to user requests, even if it cannot guarantee immediate access to all data. The database may experience brief periods of unavailability, but it should be designed to minimize downtime and provide quick recovery from failures.

Soft State
This property refers to the fact that the state of the database can change over time, even without any explicit user intervention. This can happen due to the effects of background processes, updates to data, and other factors. The database should be designed to handle this change gracefully, and ensure that it does not lead to data corruption or loss.

Eventual Consistency
This property refers to the eventual consistency of data in the database, despite changes over time. In other words, the database should eventually converge to a consistent state, even if it takes some time for all updates to propagate and be reflected in the data. This is in contrast to the immediate consistency required by traditional ACID-compliant databases.

Uses of BASE Databases
BASE databases are used in modern, highly-available, and scalable systems that handle large amounts of data. Examples of such systems include online shopping websites, social media platforms, and cloud-based services.

Difference between Base Properties and ACID Properties

ACID	BASE
ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee the integrity and consistency of data in a traditional database.	The BASE properties are a more relaxed version of ACID that trade off some consistency guarantees for greater scalability and availability.
The primary difference between the two is that ACID requires immediate consistency,	while BASE only requires eventual consistency.
ACID is better suited to traditional transactional databases.	The BASE is more suitable for use in large-scale, highly-available systems,

The CAP Theorem in DBMS

The CAP theorem, originally introduced as the CAP principle, can be used to explain some of the competing requirements in a distributed system with replication. It is a tool used to make system designers aware of the trade-offs while designing networked shared-data systems.

The three letters in CAP refer to three desirable properties of distributed systems with replicated data: consistency (among replicated copies), availability (of the system for read and write operations) and partition tolerance (in the face of the nodes in the system being partitioned by a network fault).

The CAP theorem states that it is not possible to guarantee all three of the desirable properties – consistency, availability, and partition tolerance at the same time in a distributed system with data replication.

The theorem states that networked shared-data systems can only strongly support two of the following three properties:

Consistency –
Consistency means that the nodes will have the same copies of a replicated data item visible for various transactions. A guarantee that every node in a distributed cluster returns the same, most recent and a successful write. Consistency refers to every client having the same view of the data. There are various types of consistency models. Consistency in CAP refers to sequential consistency, a very strong form of consistency.

Availability –
Availability means that each read or write request for a data item will either be processed successfully or will receive a message that the operation cannot be completed. Every non-failing node returns a response for all the read and write requests in a reasonable amount of time. The key word here is “every”. In simple terms, every node (on either side of a network partition) must be able to respond in a reasonable amount of time.

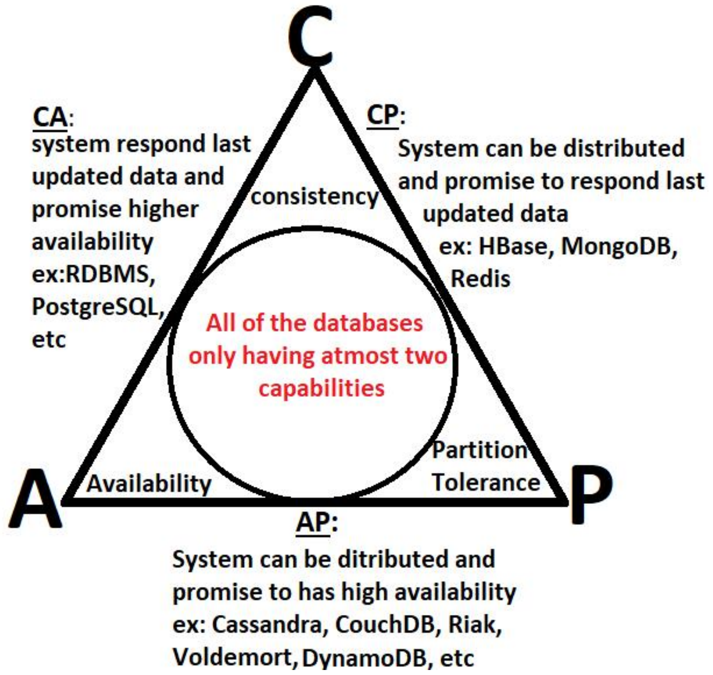
Partition Tolerance –
Partition tolerance means that the system can continue operating even if the network connecting the nodes has a fault that results in two or more partitions, where the nodes in each partition can only communicate among each other. That means, the system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

The use of the word consistency in CAP and its use in ACID do not refer to the same identical concept.

In CAP, the term consistency refers to the consistency of the values in different copies of the same data item in a replicated distributed system. In ACID, it refers to the fact that a transaction will not violate the integrity constraints specified on the database schema.

The CAP theorem states that distributed databases can have at most two of the three properties: consistency, availability, and partition tolerance. As a result, database systems prioritize only two properties at a time.

The following figure represents which database systems prioritize specific properties at a given time:


C
CA:
system respond last updated data and promise higher availability ex:RDBMS, PostgreSQL, etc
CP:
System can be distributed and promise to respond last updated data ex: HBase, MongoDB, Redis
consistency
All of the databases only having atmost two capabilities
A
Availability
Partition Tolerance
P
AP:
System can be ditributed and promise to has high availability ex: Cassandra, CouchDB, Riak, Voldemort, DynamoDB, etc

CAP theorem with databases examples

CA(Consistency and Availability)-

The system prioritizes availability over consistency and can respond with possibly stale data.

Example databases: Cassandra, CouchDB, Riak, Voldemort.

AP(Availability and Partition Tolerance)-

The system prioritizes availability over consistency and can respond with possibly stale data.

The system can be distributed across multiple nodes and is designed to operate reliably even in the face of network partitions.

Example databases: Amazon DynamoDB, Google Cloud Spanner.

CP(Consistency and Partition Tolerance)-

The system prioritizes consistency over availability and responds with the latest updated data.

The system can be distributed across multiple nodes and is designed to operate reliably even in the face of network partitions.

Example databases: Apache HBase, MongoDB, Redis.

It’s important to note that these database systems may have different configurations and settings that can change their behavior with respect to consistency, availability, and partition tolerance. Therefore, the exact behavior of a database system may depend on its configuration and usage.

for example, Neo4j, a graph database, the CAP theorem still applies. Neo4j prioritizes consistency and partition tolerance over availability, which means that in the event of a network partition or failure, Neo4j will sacrifice availability to maintain consistency.

NoSQL Data Architecture Patterns

Architecture Pattern is a logical way of categorizing data that will be stored on the Database. NoSQL is a type of database which helps to perform operations on big data and store it in a valid format. It is widely used because of its flexibility and a wide variety of services.

Architecture Patterns of NoSQL:
The data is stored in NoSQL in any of the following four data architecture patterns.

1. Key-Value Store Database
2. Column Store Database
3. Document Database
4. Graph Database

These are explained as following below.

1. Key-Value Store Database:

This model is one of the most basic models of NoSQL databases. As the name suggests, the data is stored in form of Key-Value Pairs. The key is usually a sequence of strings, integers or characters but can also be a more advanced data type. The value is typically linked or co-related to the key. The key-value pair storage databases generally store data as a hash table where each key is unique. The value can be of any type (JSON, BLOB(Binary Large Object), strings, etc). This type of pattern is usually used in shopping websites or e-commerce applications.

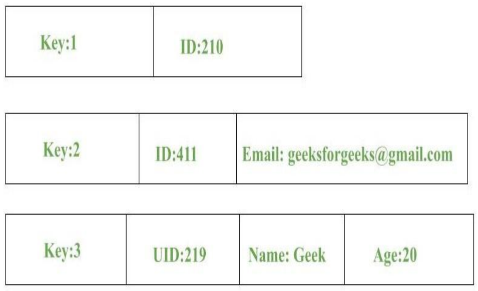
Advantages:

Can handle large amounts of data and heavy load,
Easy retrieval of data by keys.

Limitations:

Complex queries may attempt to involve multiple key-value pairs which may delay performance.
Data can be involving many-to-many relationships which may collide.

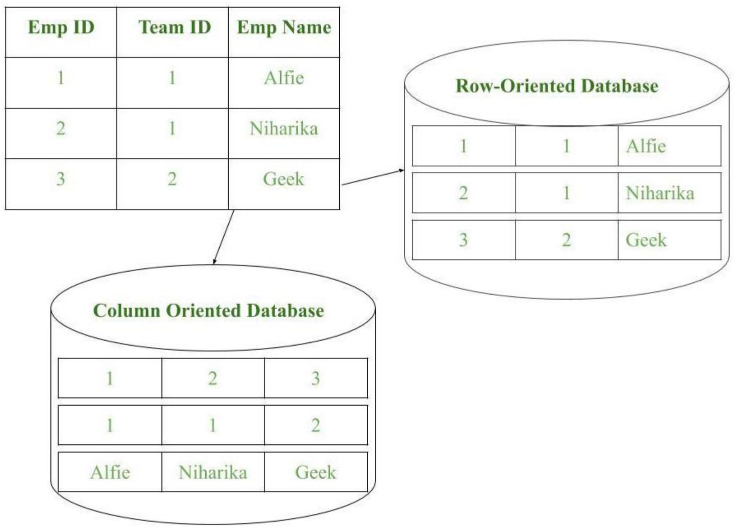
Examples:
DynamoDB
Berkeley DB



2. Column Store Database:

Rather than storing data in relational tuples, the data is stored in individual cells which are further grouped into columns. Column-oriented databases work only on columns. They store large amounts of data into columns together. Format and titles of the columns can diverge from one row to other. Every column is treated separately. But still, each individual column may contain multiple other columns like traditional databases.

Basically, columns are mode of storage in this type.



Advantages:

Data is readily available
Queries like SUM, AVERAGE, COUNT can be easily performed on columns.

Examples:

HBase
Bigtable by Google
Cassandra

3. Document Database:

The document database fetches and accumulates data in form of key-value pairs but here, the values are called as Documents. Document can be stated as a complex data structure. Document here can be a form of text, arrays, strings, JSON, XML or any such format. The use of nested documents is also very common. It is very effective as most of the data created is usually in form of JSONs and is unstructured.

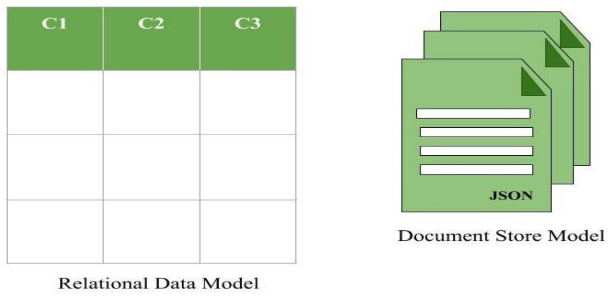


Figure – Document Store Model in form of JSON documents
Advantages:

This type of format is very useful and apt for semi-structured data.
Storage retrieval and managing of documents is easy.
Limitations:

Handling multiple documents is challenging
Aggregation operations may not work accurately.
Examples:

MongoDB
CouchDB

4. Graph Databases:

Clearly, this architecture pattern deals with the storage and management of data in graphs. Graphs are basically structures that depict connections between two or more objects in some data. The objects or entities are called as nodes and are joined together by relationships called Edges. Each edge has a unique identifier. Each node serves as a point of contact for the graph. This pattern is very commonly used in social networks where there are a large number of entities and each entity has one or many characteristics which are connected by edges. The relational database pattern has tables that are loosely connected, whereas graphs are often very strong and rigid in nature.

Advantages:

Fastest traversal because of connections.

Spatial data can be easily handled.

Limitations:
Wrong connections may lead to infinite loops.

Examples:
Neo4J
FlockDB(Used by Twitter)

Why is metadata important for a data set?
Metadata provides context for data. It provides clarity on what data represents, its type, origin, and structure. Without metadata, data can often be challenging to interpret, especially when the datasets are complex or extensive.

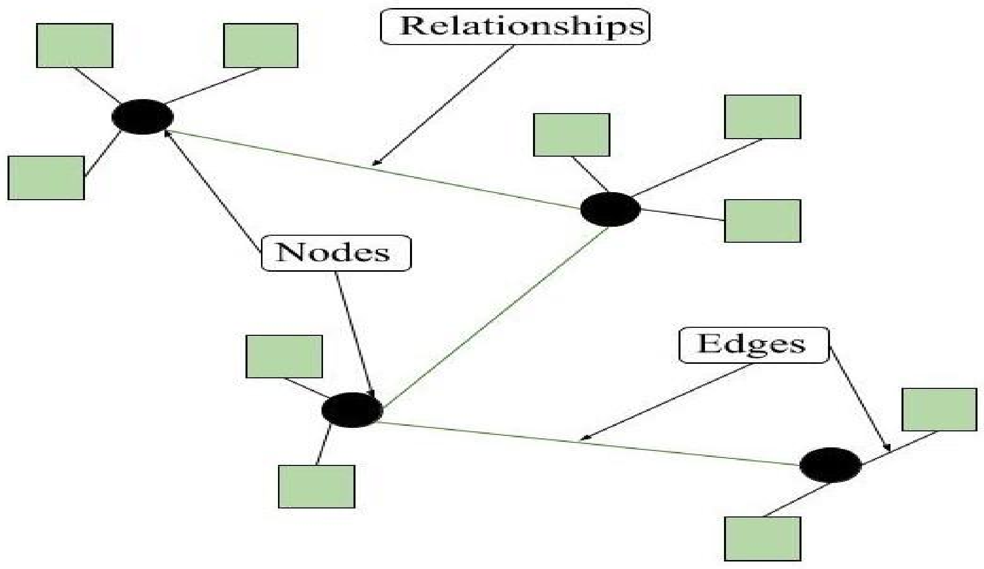
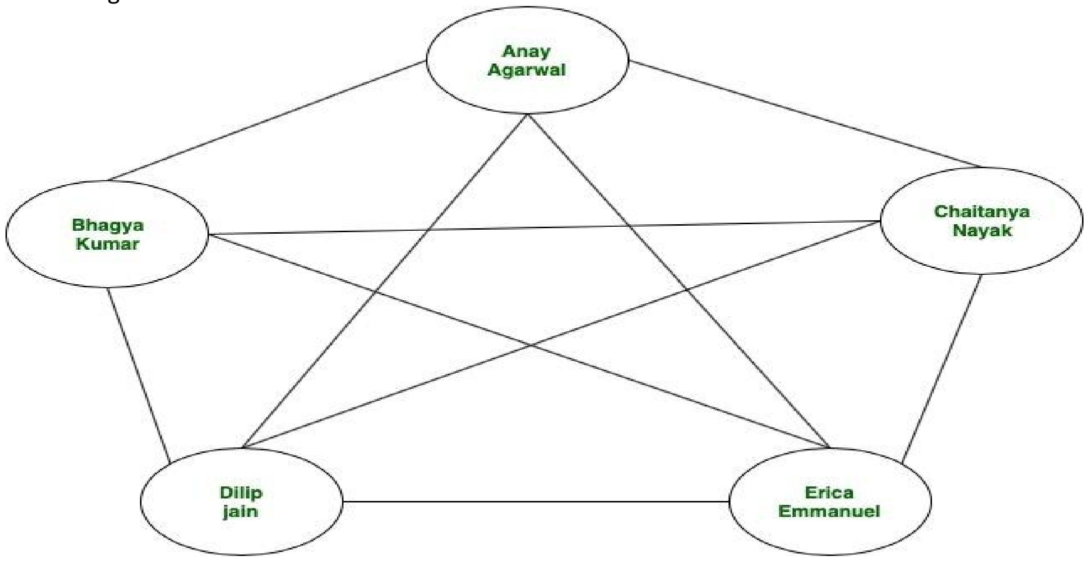
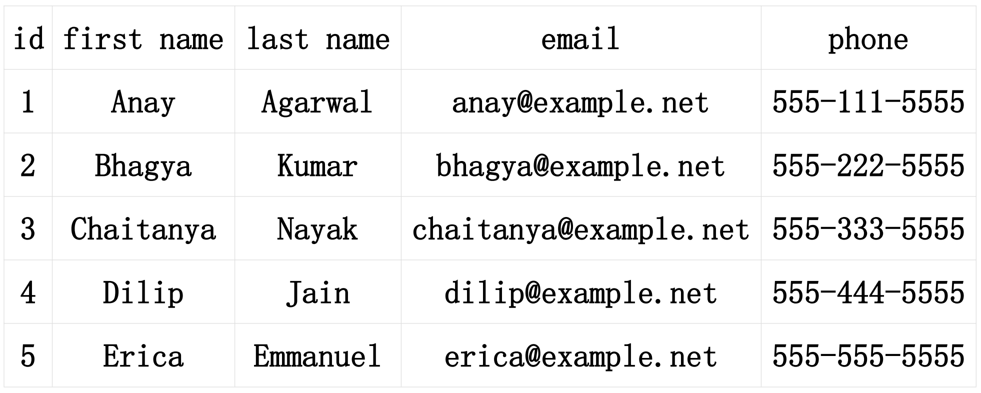


Figure – Graph model format of NoSQL Databases

Example

We have a social network in which five friends are all connected. These friends are Anay, Bhagya, Chaitanya, Dilip, and Erica. A graph database that will store their personal information may look something like this:





id	first name	last name	email	phone
1	Anay	Agarwal	anay@example.net	555-111-5555
2	Bhagya	Kumar	bhagya@example.net	555-222-5555
3	Chaitanya	Nayak	chaitanya@example.net	555-333-5555
4	Dilip	Jain	dilip@example.net	555-444-5555
5	Erica	Emmanuel	erica@example.net	555-555-5555

Now, we will also a need another table to capture the friendship/relationship between users/friends. Our friendship table will look something like this:

user_id	friend_id
1	2
1	3
1	4
1	5
2	1
2	3
2	4
2	5
3	1
3	2
3	4
3	5
4	1
4	2
4	3
4	5
5	1
5	2
5	3
5	4

Assume that the friendship table uses id's of both the friends. Assume that our social network here has a feature that allows every user to see the personal information of his/her friends. So, If Chaitanya were requesting information then it would mean she needs information about Anay, Bhagya, Dilip and Erica. We will approach this problem the traditional way(Relational database). We must first identify Chaitanya's id in the User's table:

id	first name	last name	email	phone
3	Chaitanya	Nayak	chaitanya@example.net	555-333-5555

Now, we'd look for all tuples in friendship table where the user_id is 3. Resulting relation would be something like this:

user_id	friend_id

3	1
3	2
3	4
3	5

Is columnar database just another name for a NoSQL database?

Columnar databases are one of several types of NoSQL databases. NoSQL databases include several variations from traditional row-based relational databases, including Key-Value Stores, Document Databases, Graph Databases, and Columnar or Column-Oriented Databases.

What are the advantages of a Columnar Database?

The manner in how a columnar database stores data together is functionally similar to defining an index on every column in a row-based database. For example, calculating the average price or sum of values is fast, because there are no page scans, no traversing all rows on the disk to find the prices to average. Because the data in each column is contiguous, sorting can also be performed efficiently. Columnar databases are also highly scalable and compress well.

What are the drawbacks of a Columnar Database?

Because the column data is stored together in a columnar database, a row of data is split across multiple sections. When a row is written, each column is written separately, causing slower writes and lower consistency during the write, but the data will become consistent eventually as all the writes complete. Therefore, one trade-off with columnar databases is slower write times in exchange for faster aggregate function times. In short, row operations are slower, but column operations are faster.

What is record columnar format?

Within database management systems, the record columnar file or RCFile is a data placement structure that determines how to store relational tables on computer clusters. It is designed for systems using the MapReduce framework.

Assignment 1: implement a key-value store with transactions and rollbacks (https://leetcode.com/problems/time-based-key-value-store/description/)

Assignment 2: Implement one way and two way ANOVA.

Assignment 3 : Implement text mining for social website data analysis.

Assignment 4: