Ch Mubashir_56892_Relational(SQL)_vs_No(SQL)_Assignment.

Topic :

“Relational vs. NoSQL Databases – Impact on Modern Applications”.

1. Introduction :

• The importance of databases in modern applications :

For any organization, a database serves as the repository that enables the storage of large data. Additionally, a database makes the organization's data accessible for managing, modifying, updating, organizing, and retrieving. Usually, a database is managed and controlled by a software called DBMS, short for database management system.

• Overview of Relational Databases (SQL) and NoSQL Databases :

Relational Databases (SQL) and NoSQL Databases are largely accepted types of databases used to store and manage data. Relational Databases such as MySQL and PostgreSQL store information in organized tables with rows and columns and also use SQL as the query language. They offer superior efficiency for high complexity queries and strong data integrity with ACID compliance such as for banking systems. On the other hand, NoSQL Databases such as MongoDB or Cassandra have greater flexibility because they store information in documents, key-value pairs or even graphs. These databases are built to scale efficiently and perform well with unstructured, big data. They are perfect for real time applications, big data, and systems needing horizontal scaling. The selection of either SQL or NoSQL depends on the structure of the data, scalability, and other application requirements.

• Different types of databases exist and their relevance to modern applications:

There are various types of databases since there is no one database that can serve all the complex requirements of contemporary applications.The diversity of databases is an attempt to capture the changing nature of data, the size of applications, and the unique demands of various usage scenarios.

2. Relational & NoSQL Databases :

1. Relational Databases (SQL) :

Relational databases are organized databases that hold data in tables (columns and rows) and employ Structured Query Language (SQL) for defining, querying, and modifying data. They are founded on the relational model, which stores data in tables with relations defined via primary keys and foreign keys.

Key Features:

Schema-Based: Data is stored in preorganized tables with fixed schemas (columns and data types).

Relationships: Implements one-to-one, one-to-many, and many-to-many relationships among tables.

Examples:
MySQL: An open-source relational database used by most web applications.
Oracle: A commercial database renowned for its performance, scalability, and enterprise capabilities.
SQL Server: A database developed by Microsoft for enterprise applications.

2. NoSQL Databases:

NoSQL (Not Only SQL) databases are non-relational databases that are used to store unstructured, semi-structured, or structured data. They are very flexible, scalable, and optimized for certain use cases such as big data, real-time applications, and distributed systems.

Key-Value:

Store data in the form of key-value pairs.Best suited for caching, session management, and basic queries.
Examples: Redis, DynamoDB.

Document Stores:

Store data in document formats such as JSON or XML.Flexible schema and best suited for

hierarchical data.
Examples: MongoDB, Couchbase.

Column-Family Stores:

Store data in columns instead of rows, for big data.Best suited for analytics and time-series data.
Examples: Cassandra, HBase.

Graph Databases:

Store data in nodes and edges to establish relationships.Ideal for applications such as social networks, recommendation engines, and fraud detection.
Examples: Neo4j, ArangoDB.

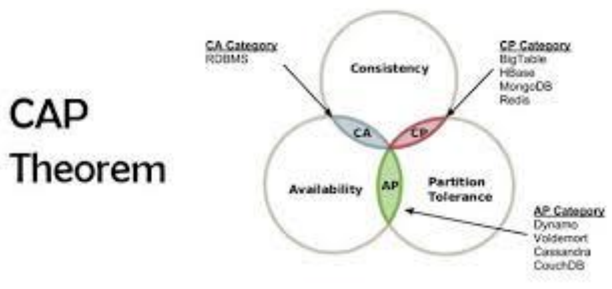
3. CAP Theorem:

The CAP theorem says that in a distributed system, it is not possible to provide all three of the following simultaneously:

Consistency (C): All reads get the latest write or an error.

Availability (A): All requests get a response, even if some of the nodes are down.

Partition Tolerance (P): The system keeps running even in the presence of network partitions (communication failures between nodes).



Application to NoSQL Databases:

NoSQL databases are architected to value two of the three CAP properties, based on their application scenario:

CP (Consistency + Partition Tolerance): Databases such as MongoDB and
HBase value consistency and partition tolerance. They maintain data as correct and consistent across nodes even in network failures but compromise on availability.

AP (Availability + Partition Tolerance): Cassandra and Couchbase give utmost importance
to availability and partition tolerance. They allow the system to stay up and
responsive even though some data will be temporarily inconsistent.

CA (Consistency + Availability): Mostly not used in distributed systems since partition tolerance is a must for scalability and fault tolerance.

3. Key Differences & Comparisons :

SQL vs. NoSQL: A Detailed Comparison

1. Data Structure

SQL (Relational Databases):

Structure: SQL databases are table-based with a pre-defined schema. Data is stored in rows and columns, and relationships between tables are defined using foreign keys.

Example: MySQL, PostgreSQL, Oracle.

NoSQL (Non-Relational Databases):

Structure: NoSQL databases may be document-based, key-value pairs, wide-column stores, or graph databases. They are schema-less, meaning more flexible data models are possible.

Example: MongoDB (document-based), Redis (key-value), Cassandra (wide-column), Neo4j (graph).

2. Scalability

SQL:

Scalability: SQL databases are most commonly scaled vertically, i.e., you just add power to the current hardware (e.g., CPU, RAM). You can scale horizontally (add additional servers) but it is harder and usually necessitates sharding.

Example: Scaling a MySQL database could entail upgrading the server or read replicas.

Scalability:

NoSQL databases are designed for horizontal scalability, meaning you can add more servers to handle increased load. This makes them well-suited for distributed systems and large-scale applications.

Example: Cassandra can easily scale out by adding more nodes to the cluster.

3. Flexibility

SQL:

Flexibility: SQL databases require a predefined schema, which can be rigid. Changes to the schema often require migrations, which can be time-consuming and risky.

Example: It takes modifying the table structure to add a new column to a table in a SQL database.

NoSQL:

Flexibility:

NoSQL databases are schema-less, and therefore they support dynamic and flexible data models. This facilitates easy adaptability to evolving requirements without considerable downtime.

Example: In MongoDB, you can add new fields to documents without impacting existing data.

4. Performance

SQL:

Performance: NoSQL databases excel in handling complex transactions and queries, particularly structured data. Write performance can be very high but read performance degrades with complex transactions.

Example: An optimally indexed NoSQL database returns data at speed with JOIN operations.

Performance:

NoSQL databases are optimized for specific use cases, such as high write throughput or low-latency reads. They can handle large volumes of unstructured data more efficiently than SQL databases.

Example: Redis can handle millions of operations per second with low latency.

. 5 Consistency

SQL

SQL databases normally adhere to the ACID (Atomicity, Consistency, Isolation, Durability) model with strong consistency. It implies that the data will be consistent across the database after a transaction has been committed.

Example: A bank transaction within a SQL database will make the debit and credit operations atomic and consistent.

NoSQL:

Consistency: NoSQL databases tend to use the BASE (Basically Available, Soft state, Eventual consistency) model, where availability and partition tolerance are given more importance than strong consistency. This implies that data can be eventually consistent, but not necessarily immediately.

Example: In a distributed NoSQL database such as Cassandra, data can take a while to propagate to all nodes, resulting in eventual consistency.

ACID vs. BASE Models

ACID (Atomicity, Consistency, Isolation, Durability):

Atomicity: It guarantees that all the operations within a transaction get executed successfully; otherwise, the transaction is rolled back.

Consistency: It guarantees that the database is always in a consistent state both before and after the transaction.

Isolation: It guarantees that one transaction does not interfere with other transactions that are running concurrently.

Durability: It guarantees that once a transaction is committed, it will be permanent even if there is a system failure.

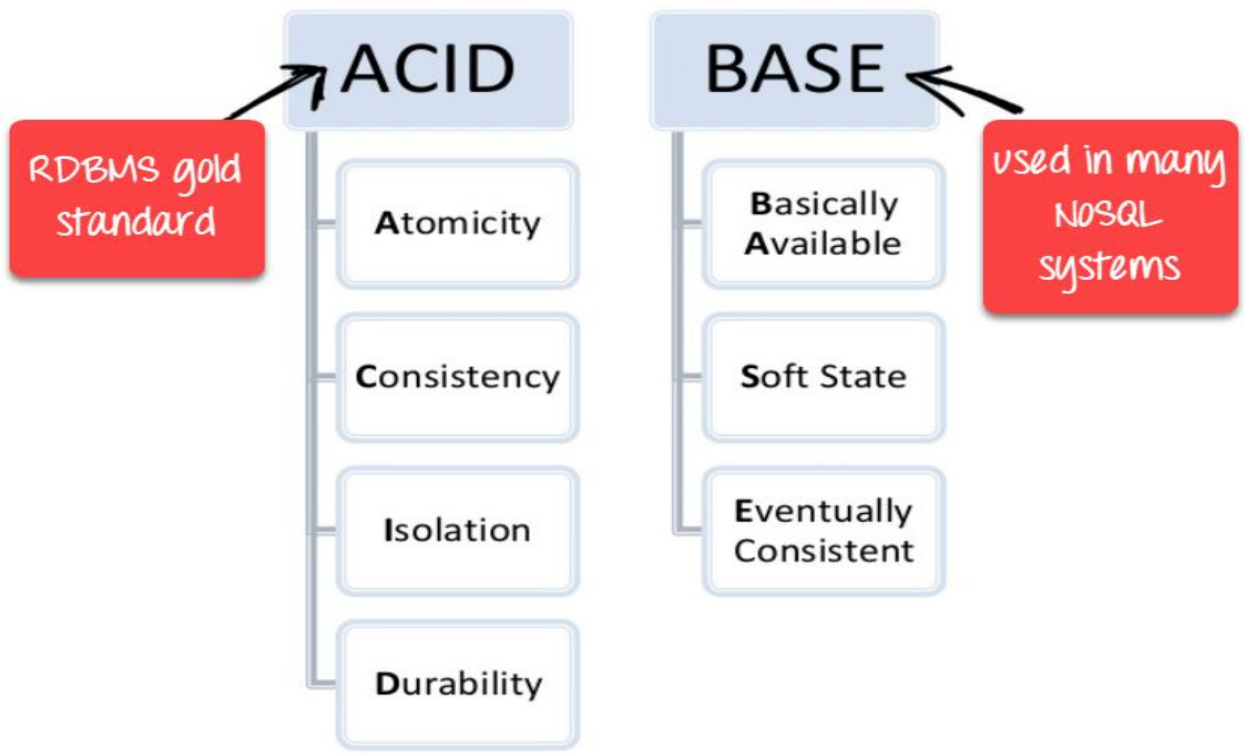
Impact on Performance: ACID compliance may result in increased latency and decreased throughput because of the cost of maintaining strong consistency and isolation.

BASE (Basically Available, Soft state, Eventual consistency):

Basically Available: The system is always available, even in failure scenarios.

Soft state: The system state will evolve with time, even without user input, due to eventual consistency.

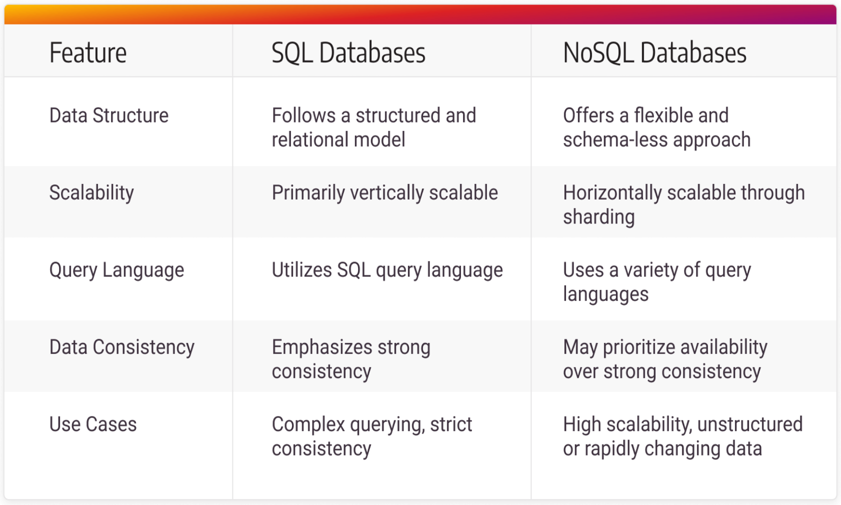
Eventual consistency: The system will converge to consistency eventually, provided no new updates are added.



Impact on Performance:

BASE enables greater availability and scalability, which typically translates to improved performance for distributed systems, but at the expense of strong consistency.

A comparison table summarizing SQL vs. NoSQL :



Feature	SQL Databases	NoSQL Databases
Data Structure	Follows a structured and relational model	Offers a flexible and schema-less approach
Scalability	Primarily vertically scalable	Horizontally scalable through sharding
Query Language	Utilizes SQL query language	Uses a variety of query languages
Data Consistency	Emphasizes strong consistency	May prioritize availability over strong consistency
Use Cases	Complex querying, strict consistency	High scalability, unstructured or rapidly changing data

4. Impact on Modern Applications :

Relational Databases (SQL)

Relational databases are structured and built as tables with rows and columns to store data. They work on the foundation of the ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure data integrity and reliability. SQL (Structured Query Language) is the communication medium with such databases.

How they power modern applications:

In cases where data integrity, complex querying, and structured data is required, relational databases outperform the rest. They are there where transactions are needed, like in financial systems with utmost consistency and accuracy.

Real-world Examples:

Banking: Transactions, customer accounts, and financial records are being managed in the relational database in such a way so as to maintain consistency with security.

E-commerce: This deals with product catalogs, customer orders, inventory management-supporting complex queries and transactions.

ERP Systems: Enterprise Resource Planning systems use a relational database to bring together and manage business processes like supply chain, HR, and finance.

NoSQL Databases

NoSQL databases are non-relational databases created to handle unstructured and semi-structured data. They offer extreme scalability and flexibility, allowing them to fulfil configured requirements in modern applications with regards to massive volumes of data or fast changes in the data structure.

How they power modern applications:

This database is tuned for speed, scalability, and flexibility, suitable for applications which require real-time data processing, high-availability, and horizontal scaling.

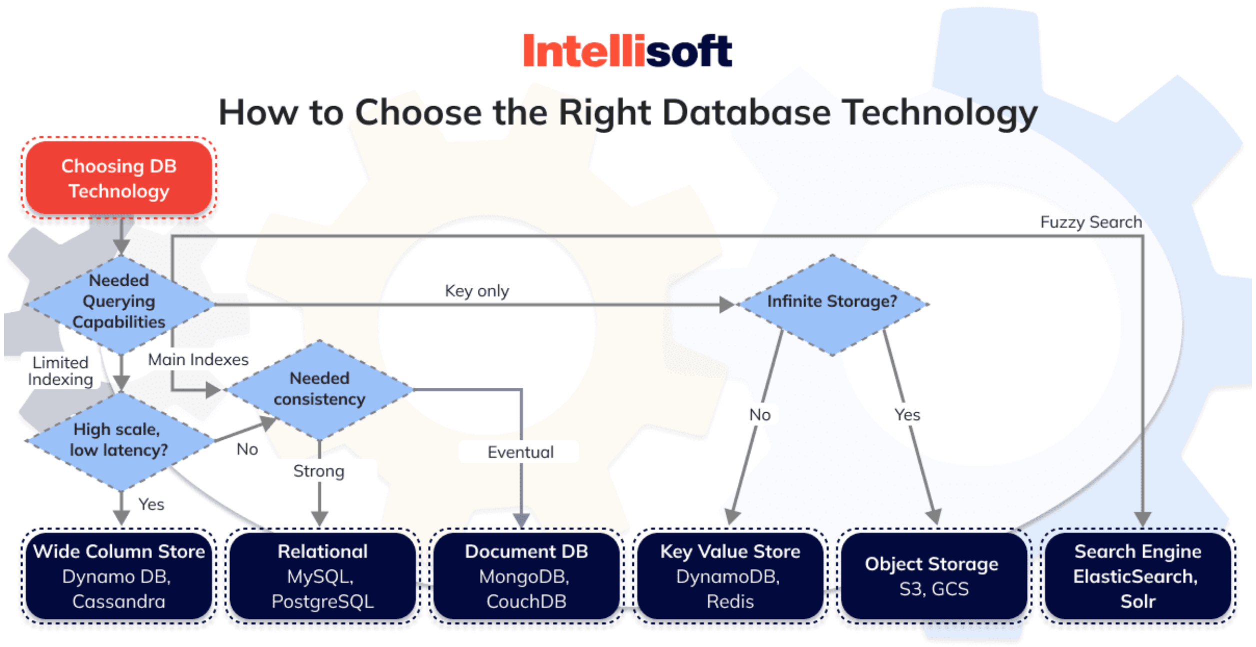
Real-world examples:

Social Media: Such big players like Facebook or Twitter use NoSQL databases to hold and derive huge quantities of user-generated content, real-time updates, and interactions.

Real-time Analytics: NoSQL databases like Cassandra or MongoDB conduct the function of real-time processing and analytics of substantial amounts of data, including recommendation engines or fraud detection systems.

IoT (Internet of Things): NoSQL databases process and store data from millions of connected devices and handle high read and write throughput.

How businesses choose databases based on application needs?


Intellisoft
How to Choose the Right Database Technology
Choosing DB Technology
Needed Querying Capabilities
Key only
Fuzzy Search
Infinite Storage?
Limited Indexing
Main Indexes
Needed consistency
High scale, low latency?
No
Strong
Eventual
No
Yes
Yes
Wide Column Store Dynamo DB, Cassandra
Relational MySQL, PostgreSQL
Document DB MongoDB, CouchDB
Key Value Store DynamoDB, Redis
Object Storage S3, GCS
Search Engine ElasticSearch, Solr

5. Future Trends & Challenges

Introducing NewSQL :

NewSQL databases aim to combine the best of both worlds – the scalability of noSQL and the reliability of traditional SQL databases. They provide a middle ground solution that addresses the limitations of both SQL and noSQL databases. NewSQL databases offer the ACID properties while also providing horizontal scalability and high performance databases. This makes them suitable for modern applications that require both consistency and scalability.

How NewSQL Works ?

NewSQL databases achieve their goals by implementing various techniques. They leverage distributed architectures, sharding, and replication to ensure data is distributed across multiple nodes while maintaining consistency. NewSQL databases also use innovative indexing techniques and query optimization strategies to provide high-performance queries. By combining these techniques, NewSQL databases offer a powerful solution for modern data-intensive applications and address database scalability concerns.

Challenges in NewSQL Databases

1. **Scalability**:

    - While NewSQL databases scale horizontally, true linear scaling across distributed systems is hard. Balancing data distribution, replication and query performance is tricky.
    - As you add more nodes, managing network latency and communication between nodes gets harder.
2. **Security**:
    - Distributed systems are more vulnerable to security threats like data breaches and unauthorized access. Robust encryption, authentication and authorization across multiple nodes is critical.
    - Compliance with data protection regulations (e.g. GDPR, CCPA) adds another layer of complexity.
3. **Data Consistency**:
    - Strong consistency in a distributed environment can impact performance. NewSQL databases use techniques like **distributed consensus protocols** (e.g. Raft, Paxos) to ensure consistency but that introduces latency.
    - Trade-offs between consistency, availability and partition tolerance (CAP theorem) need to be managed.
4. **Complexity of Deployment and Management**:
    - NewSQL systems are hard to deploy, configure and maintain, you need specialized knowledge and tools.
    - Monitoring and troubleshooting distributed systems is harder than traditional monolithic databases.
5. **Ecosystem Maturity**:
    - Compared to established SQL and NoSQL databases, NewSQL ecosystem is still immature. Tools, libraries and community support is not as extensive.

Future of Database Technologies

1. **SQL and NoSQL convergence**:
    - The line between SQL and NoSQL will continue to blur, more databases will offer hybrid capabilities. For example, PostgreSQL now supports JSON and other NoSQL-like features, while NoSQL databases like MongoDB are adding SQL-like query capabilities.
2. **Serverless Databases**:
    - Serverless database architectures where the infrastructure is fully managed by the provider will gain traction. These systems scale automatically based on demand, reduce operational overhead for developers.
3. **AI and Machine Learning**:
    - Databases will integrate AI and machine learning capabilities for tasks like query optimization, anomaly detection and predictive analytics. For example, automated indexing and tuning based on workload patterns.

Books Reference :

"Database System Concepts" by Abraham Silberschatz, Henry F. Korth, and S. Sudarshan

"SQL and Relational Theory: How to Write Accurate SQL Code" by C.J. Date

Academic Papers :

"The CAP Theorem" by Eric Brewer

"Cassandra - A Decentralized Structured Storage System" by Avinash Lakshman and Prashant Malik

Links :

https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases

https://atlan.com/relational-database-vs-nosql/