02 SQL vs NoSQL

HW

- CAP - Theorem
- Database sharding

SQL vs NoSQL

1. Introduction

When designing databases for applications, one of the first architectural decisions is whether to use a **relational database (SQL)** or a **non-relational database (NoSQL)**. This choice affects how data is stored, queried, and scaled.

2. SQL (Relational Databases)

2.1 What is SQL?

- **SQL** stands for **Structured Query Language**.
- It is used to manage **relational databases**.
- Data is stored in **tables** (rows and columns) with **fixed schema**.

2.2 Examples of SQL Databases

- MySQL
- PostgreSQL
- Microsoft SQL Server
- Oracle Database
- SQLite

2.3 Key Characteristics

Feature	Description
Structured Schema	Tables have predefined columns and data types.
ACID Compliance	Ensures data consistency, reliability (Atomicity, Consistency, Isolation, Durability).
Joins Support	SQL is powerful in joining multiple tables through foreign keys.
Schema-First Design	Schema must be defined before inserting data.

2.4 Use Cases

- Financial transactions
- ERP systems
- Inventory management
- Applications needing **complex queries** or **data integrity**

3. NoSQL (Non-Relational Databases)

3.1 What is NoSQL?

- **NoSQL** stands for "**Not Only SQL**".

- It refers to a broad category of **non-relational databases**.
- Designed for **flexibility**, **horizontal scalability**, and **high performance**.

3.2 Types of NoSQL Databases

Type	Description	Examples
Document	Stores data as JSON-like documents	MongoDB, CouchDB
Key-Value	Data is stored as key-value pairs	Redis, DynamoDB
Column-Family	Data stored in columns rather than rows	Apache Cassandra, HBase
Graph	Nodes and edges to represent data relationships	Neo4j, ArangoDB

3.3 Key Characteristics

Feature	Description
Schema-less	Data can have different fields and structures.
Eventual Consistency	Often favors performance over strict consistency.
Horizontal Scaling	Scales out easily by adding more servers.
Optimized for Big Data	Designed for high throughput and low latency.

3.4 Use Cases

- Real-time analytics
- IoT and sensor data
- Content Management Systems (CMS)
- Recommendation engines
- Social networks
- Mobile applications

4. Detailed Comparison: SQL vs NoSQL

Feature	SQL (Relational)	NoSQL (Non-Relational)
Data Model	Relational (tables)	Document, Key-Value, Graph, Column-Family
Schema	Fixed, predefined	Dynamic, flexible
Scalability	Vertical (scale-up)	Horizontal (scale-out)
Query Language	SQL	Varies (MongoDB Query Language, CQL, etc.)
ACID Compliance	Strongly enforced	Often relaxed for performance (CAP tradeoffs)
Joins Support	Native and powerful	Often not supported or expensive
Transaction Support	Strong, multi-row transactions	Varies; some have limited or eventual consistency
Best For	Structured data, complex queries, data integrity	Unstructured/semi-structured data, fast and massive-scale workloads
Examples	MySQL, PostgreSQL, Oracle	MongoDB, Redis, Cassandra, Neo4j

5. When to Choose What?

Choose SQL When:

- Data structure is **well-defined and consistent**.
- **Complex queries** or **joins** are required.
- You need **strong consistency** (ACID).
- Application is **transaction-heavy**.

Choose NoSQL When:

- You need to handle **huge volumes of unstructured or semi-structured data**.
- The schema may evolve frequently.
- You need **fast reads/writes** at **massive scale**.
- Application needs **high availability** and **distributed systems**.

6. Trade-offs and CAP Theorem

CAP Theorem (Consistency, Availability, Partition Tolerance) is especially relevant for NoSQL:

Tradeoff	SQL	NoSQL
C + A	Yes, if no partition	Not possible if partitioned
C + P	Possible (but low availability)	Some databases prefer this (e.g., MongoDB)
A + P	Usually not prioritized	Many NoSQL DBs choose this (e.g., Cassandra, DynamoDB)

7. Summary Table: SQL vs NoSQL

Feature	SQL	NoSQL
Model	Relational (Table-based)	Non-Relational (Document, Key-Value, Graph, Column)
Schema	Fixed, predefined	Flexible, schema-less
Scalability	Vertical (scale-up)	Horizontal (scale-out)
Transactions	Full ACID	Eventual consistency, BASE model
Performance	Optimized for complex joins & integrity	Optimized for speed and volume
Best Suited For	Structured data, enterprise applications	Big Data, real-time apps, changing schemas
Examples	MySQL, PostgreSQL, Oracle	MongoDB, Cassandra, Redis, CouchDB
Consistency	Strong consistency (ACID)	Tunable consistency (CAP theorem)
Joins	Supports complex joins	Limited or none
Query Language	Standard SQL	Varies by database
Storage Format	Row-based	JSON/BSON, key-value, wide-column, graph edges
Tooling and Maturity	Mature, strong tooling and support	Growing ecosystem, newer technologies

Final Takeaways

- **SQL** = structured, reliable, strong consistency, enterprise-grade.
- **NoSQL** = scalable, flexible, faster for massive or varied datasets.
- Choose based on your **use case**, **scaling requirements**, and **data structure**.

- In modern architectures, **polyglot persistence** (using both SQL and NoSQL) is common for optimizing across components.