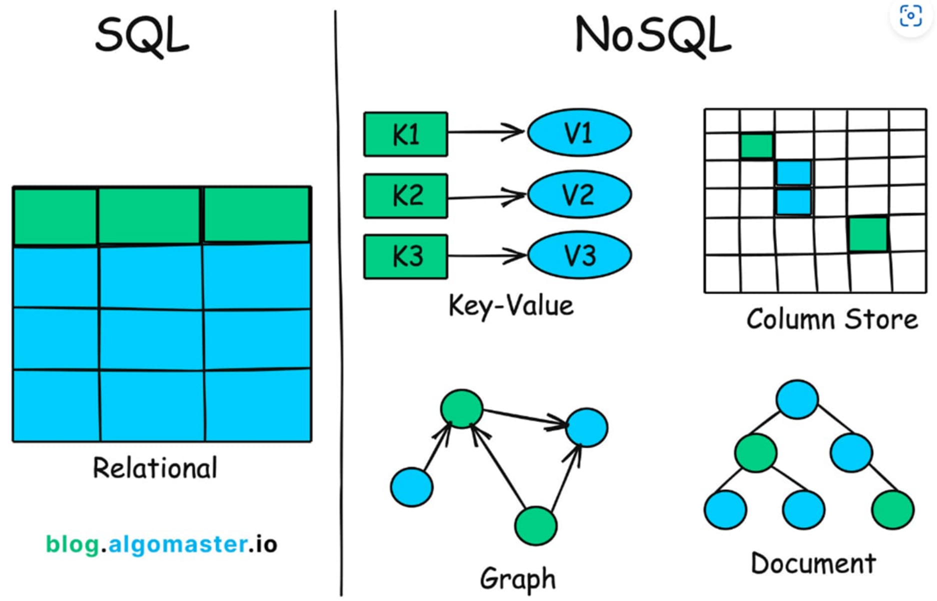
25th September

What is SQL vs What is NoSQL

- **SQL (Relational Databases):**
  SQL databases store data in structured tables (rows & columns). They enforce a fixed schema (you define tables, columns, types ahead of time). Relations (foreign keys), constraints, and ACID transactions (Atomicity, Consistency, Isolation, Durability) are core features. Common examples: MySQL, PostgreSQL, Oracle, SQL Server.
- **NoSQL (Non-Relational Databases):**
  NoSQL databases store data in formats other than the classic table-row model. They support flexible or dynamic schemas, such as document stores (JSON/BSON), key-value pairs, wide-column, or graph data. They often relax some constraints (such as full ACID or requiring rigid relations) to gain in flexibility, scalability, or speed. Examples: MongoDB, Cassandra, Redis, DynamoDB, Couchbase etc.



blog.algomaster.io

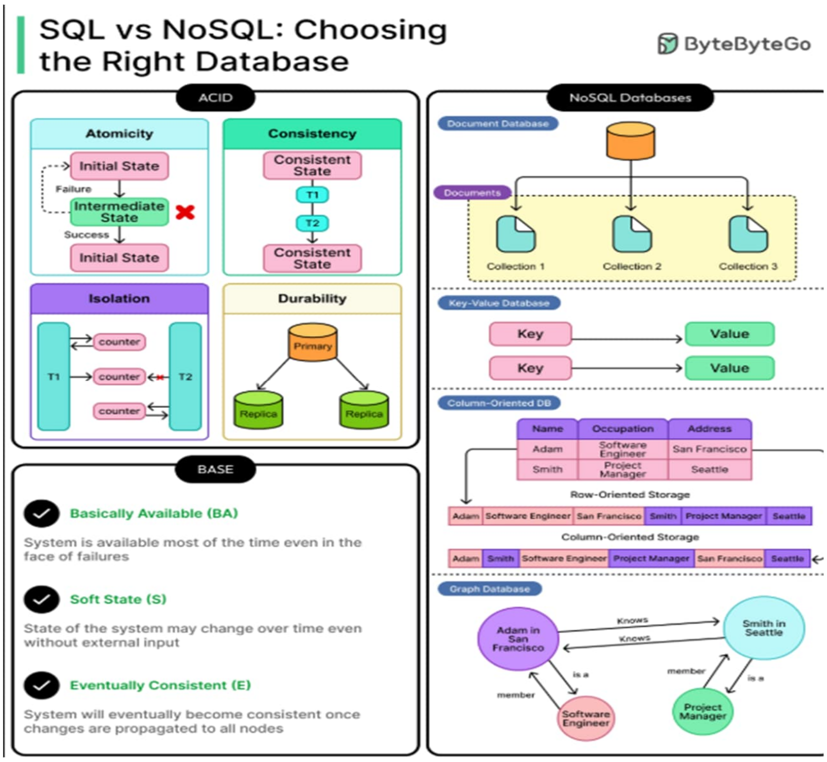
Scenarios / When to Choose One Over the Other

Here are situations where one tends to be better suited:

Situation / Requirement	**Prefer SQL when …**	**Prefer NoSQL when …**

Strong transactional integrity / financial use cases	You need strict transactions (e.g. banking, order payments), complex joins, relational consistency across multiple tables.	Less critical; eventual consistency okay; operations can tolerate some delay in consistency.
Schema stability vs schema evolution	Data structure is known in advance, doesn't change often; you want strong typing, constraints.	Data model evolves, new features or fields get added often; unstructured or semi-structured data.
Scale / volume / performance	Moderate scale, emphasis on read/write with joins, reporting, analytics with SQL queries.	Very large scale, high write or read throughput, distributed systems, big data / IoT / streaming / logs.
Type of queries needed	Complex queries with aggregations, joins, ad-hoc reporting.	Simple queries, key-value lookups, or queries tailored to document structure; sometimes map-reduce or aggregation capabilities but generally less relational.
Flexibility vs schema enforcement	Rigorous schema enforcement helps maintain data integrity, cleaner constraints, predictable structure.	Flexibility helpful for rapidly changing apps, schema-less storage, storing varied/unexpected attributes.
Geographic distribution / scaling	Vertical scaling (bigger server), sometimes sharding but more complex.	Horizontal scaling is easier; distributing data across many nodes or servers.

Advantages: SQL vs NoSQL



SQL vs NoSQL: Choosing the Right Database

Name	Occupation	Address
Adam	Software Engineer	San Francisco
Smith	Project Manager	Seattle

Row-Oriented Storage

Adam	Software Engineer	San Francisco	Smith	Project Manager	Seattle

Column-Oriented Storage

Adam	Smith	Software Engineer	Project Manager	San Francisco	Seattle

✓ Basically Available (BA)

System is available most of the time even in the face of failures

✓ Soft State (S)

State of the system may change over time even without external input

✓ Eventually Consistent (E)

System will eventually become consistent once changes are propagated to all nodes

Here are what each does well (and their trade-offs).

Feature	Advantages of SQL	Advantages of NoSQL
Data Integrity & Consistency	Strong ACID guarantees; great for use-cases where you can't afford data anomalies (e.g. financial, billing).	Many NoSQL systems are eventually consistent or provide relaxed consistency; can be tuned per application. Useful where availability & partition tolerance are priorities.
Schema / Data Structure	Rigid schema enforces uniform structure; good for ensuring data quality and predictable structure.	Flexible schema allows storing unstructured or semi-structured data easily; adding new fields or different kinds of objects is simpler.
Complex Queries & Analytics	Joins, aggregations, sub-queries, Views, Stored Procedures, etc. SQL excels here.	Some NoSQL systems support aggregation pipelines, but complex relational queries (many joins) are harder / less efficient. Might require denormalization or duplicating data.
Scalability	Vertical scaling works; some SQL systems support sharding or	Designed for horizontal scaling; distributed databases, replication,

	distributed SQL, but with more complexity.	partitioning etc. Good for large volumes, big data, high throughput.
Development Speed & Flexibility	Mature tooling, well-known patterns, established standards, many developers familiar with SQL.	Faster iteration when schema isn’t locked; ability to evolve features without heavy schema migrations; stored JSON etc.

Real-Life / Interesting Examples

- **E-commerce / Retail Platform:**
  Use SQL database (e.g. PostgreSQL) to manage orders, inventory, payments (so you have ACID guarantees, relational integrity). But use a NoSQL database like MongoDB to store user session data, product reviews, or personalization data (which may have varying fields).
- **Social Media / Messaging App:**
  NoSQL (e.g. Cassandra, DynamoDB) to store large volumes of posts, comments, likes, which have high write/read rates and flexible data (images, text, metadata). But SQL for user account info, billing, and configuration data where consistency and relations matter.
- **IoT / Sensor Data / Time Series Analytics:**
  NoSQL wide-column or time-series databases (e.g. Cassandra, InfluxDB) are good when you're ingesting a huge stream of data from sensors and need horizontal scaling.
- **Banking / Accounting Systems:**
  SQL is the go-to because transactions, consistency, audit trails are essential.