Date

No SQL

Benefits of Relational Databases

→ Following are the main benefits of Relational Databases.

- Designed for all purposes
- ACID
- Strong consistency, concurrency recovery
- Mathematical background
- Standard Query language (SQL)
- Lots of tools to use with i.e. Reporting services, entity frameworks, ...

ACID Properties

→ To ensure integrity of data, the database system must maintain:

- Atomicity
  - * Ensures that each transaction is treated as a single unit that either completes entirely or does not happen at all.
  - * If one part of the transaction fails, the entire transaction fails and the database remains unchanged.
- Consistency
  - * Ensures that a transaction takes the database from one valid state to another valid state, maintaining all defined rules, constraints, and data integrity.
- Isolation
  - * Ensures that concurrent transactions do not interfere with each other.
  - * Each transaction should act as if it is the only one being executed, even if others are happening at the same time.
- Durability
  - * Once a transaction is committed, the changes are permanent, even in the case of a system crash or power failure.

Horizontal Scaling (Scaling Out)

→ Horizontal scaling means that you scale by adding more machines into your pool of resources. Think of it like: more workers to get more work done.

Vertical Scaling (Scaling Up)

→ Vertical scaling means that you scale by adding more power (CPU, RAM) to an existing machine.

Impedance mismatch

→ An impedance mismatch can occur when accessing a relational database in an OOP language. Problems can arise because OOP languages like C++ or Python have very different approaches to accessing data.

→ Relational databases were not built for distributed applications because:

- Joins are expensive
- Hard to scale horizontally
- Impedance mismatch occurs
- Expensive (product cost, hardware, Maintenance).

Also its weak in

- Speed (Performance)
- High availability
- Partition tolerance.

NoSQL

→ A NoSQL database provides a mechanism for storage and retrieval of data that employs less constrained consistency models than traditional relational database.

→ NoSQL systems are also referred to as "Notonly SQL" to emphasize that they do in fact allow SQL-like query languages to be used.

→ NoSQL avoids:

- Overhead of ACID transactions
- Complexity of SQL query
- Burden of up-front schema design
- DBA presence
- Transactions (it should be handled at application layer)

→ NoSQL provides:

- Easy and frequency changes to DB
- Fast development
- Large data volumes (Google)
- Schema Less

→ When/ Why to use NoSQL

- Traditional RDBMS is too restrictive.
- ACID support is "not" "really" needed.
- Object-to-Relational (O/R) impedance
- RDBMS is not distributed or scalable by nature
- Logging data from distributed sources
- Storing events / temporal data
- Temporary Data
- Data requiring flexible schema
- Polyglot Persistence
  - ★ Use different types of databases for different data needs.
  - ★ NoSQL fits in when relational models don't suit a specific data type.

→ When NOT to use NoSQL

- Financial Data
- Strict ACID compliance
- Business - Critical Data

Schema-less Datamodel

→ In Relational Databases

- You can't add a record which does not fit the schema.
- You need to add NULLs to unused items in a row.
- We should conside the datatypes i.e. you can't add a string to an integer field.
- You can't add multiple items in a field (you should create another table: primary-key, foreign key, joins, normalization, ...).

→ In NoSQL Databases:

- There is no schema to consider.
- There is no unused cell.
- There is no datatype (implicit)
- Most of considerations are done in application layer.
- We gather all items in an aggregate (document).

NoSQL Datamodels

→ NoSQL databases are classified in four major datamodels:

- Key - value
- Document
- Column family
- Graph

→ Each database has its own query language.

i) Key - value Data

- Key - value databases are most simplest NoSQL databases.
- Internally, these databases use a hash table structure where each unique key maps to a specific value.
- You use a key (string) to retrieve associated value.
- There is no schema or enforced structure for the stored values.
- All data are stored as pairs: one key, one value.
- Basic operations:

  Insert (key, value)
  Fetch (key)
  Update (key)
  Delete (key)

ii) Column Family

→ A column family is similar to a table in a relational database, but much more flexible:

- Data is stored in rows, and each row has a unique row key.
- Each row is made up of columns, but not all rows need to have the same columns.
- Columns are grouped into column families.
- Each column in a row is stored as a tuple:

(column name, value, timestamp)

Example:

→ Facebook search uses Casandra which uses a column family data model.

Here is a comparison b/w MySQL and Casandra

- MySQL > 50 GB Data
  Writes Avg: ~ 300 ms
  Reads Avg: ~ 350 ms
- Casandara > 50 GB Data
  Writes Avg: 0.12 ms
  Reads Avg: 15 ms

iii) Graph Data Model

→ The Graph Data Model is one of the core data models used in NoSQL databases, It is designed to represent and query highly

Date

connected data efficiently.

→ Following are main properties of Graph data model:

- Based on Graph Theory
- Scale vertically, no clustering.
- You can use graph algorithms easily.
- Transactions
- ACID

iv) Document-Based data model

→ The Document-based data model is a popular type of NoSQL data model used by database like MongoDB.

→ It is designed to store, retrieve, and manage semi-structured data in the form of documents, usually in JSON, BSON or XML formats.

→ A document in this context is a self-contained unit of data similar to a record or row, but much more flexible and hierarchical.

→ Following are main properties of this data model:

- Each document is stored with a unique key. The document can hold complex, structured data.
- Efficient data retrieval is achieved through B-Tree indexes. Indexes can be created on any fields within the documents.
- Documents can hold:
  - → Key-value pairs
  - → Key-array pairs
  - → Nested documents

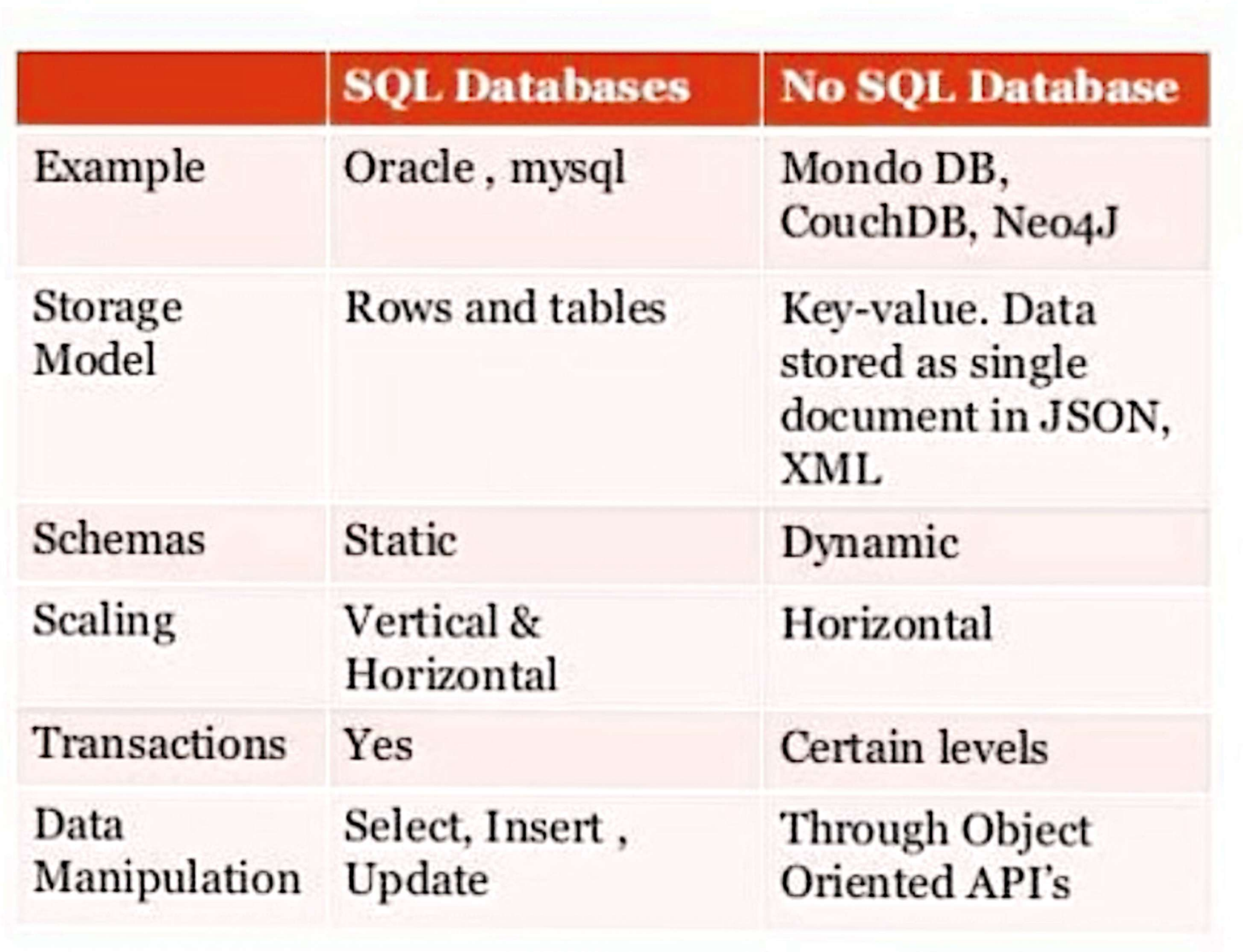
CAP Theorem

→ The CAP theorem states that it is impossible for any shared data-system to guarantee simultaneously all of the following three properties:

- Consistency — once data is written, all future read requests will contain that data.
- Availability — the database is always available & responsive.
- Partition Tolerance — if part of the database is unavailable, other parts are uneffected.

→ We can't achieve all the three items in distributed database systems.

Differences



	SQL Databases	No SQL Database
Example	Oracle , mysql	Mondo DB, CouchDB, Neo4J
Storage Model	Rows and tables	Key-value. Data stored as single document in JSON, XML
Schemas	Static	Dynamic
Scaling	Vertical & Horizontal	Horizontal
Transactions	Yes	Certain levels
Data Manipulation	Select, Insert , Update	Through Object Oriented API's

In Conclusion!

- RDBMS is a great tool for solving ACID problems
  - When data validity is super important
  - When you need to support dynamic queries
- NoSQL is a great tool for solving data availability problems
  - When it’s more important to have fast data than right data
  - When you need to scale based on changing requirements
- Pick the right tool for the job