Data Modeling Relational Vs NoSQL

July 13, 2021

1 Data Modeling

Definition An abstraction that organizes elements of data and how they relate to each other. The process of creating data models for an information system. Data modeling is important because:

- Data Organization: The organization of the data for your applications is extremely important and makes everyone's life easier.
- Use cases: Having a well thought out and organized data model is critical to how that data can later be used. Queries that could have been straightforward and simple might become complicated queries if data modeling isn't well thought out.
- Starting early: Thinking and planning ahead will help you be successful. This is not something you want to leave until the last minute.
- Iterative Process: Data modeling is not a fixed process. It is iterative as new requirements and data are introduced. Having flexibility will help as new information becomes available.

1.1 Relational Databases

- This model organizes data into one or more tables or (relations) of columns and rows with a Unique key identifying each row.
- Generally, each table represents one entity type. Table is a collections of rows and columns
- A software system to maintain Relational Databases is an RDBMS.

Common RDBMS

• Oracle	Used in Banking systems
• Teradata	
• MySQL	
• PostgreSQL	
• SQLite	File Format used in development and simple tasks.

Basics of Relational Databases

• Database/Schema	Is a collection of tables.
• Tables/Relation	A group of rows sharing the same labeled elements.
• Column/Attribute	Labeled element E.g name, city
• Row/Tuple	A single item E.g Amanda, email, NYC

1.1.1 When to use Relational Database?

- **Flexibility for writing in SQL queries** With SQL being the most common database query language.
- **Modeling the data not modeling queries** Modeling is independent of queries.
- **Ability to do JOINS** Unique to relational databases.
- **Ability to do aggregations and analytics** GROUPBY ORDER BY, SUM, AVG
- **Secondary Indexes available**: You have the advantage of being able to add another index to help with quick searching.
- **Smaller data volumes**: If you have a smaller data volume (and not big data) you can use a relational database for its simplicity.
- **ACID Transactions**: Allows you to meet a set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, and thus maintain data integrity.
- **Easier to change to business requirements**

1.1.2 When NOT to use Relational Database?

- **Have large amounts of data**: Relational Databases are not distributed databases and because of this they can only scale vertically by adding more storage in the machine itself. You are limited by how much you can scale and how much data you can store on one machine. You cannot add more machines like you can in NoSQL databases.
- **Need to be able to store different data type formats**: Relational databases are not designed to handle unstructured data.
- **Need high throughput – fast reads**: While ACID transactions bring benefits, they also slow down the process of reading and writing data. If you need very fast reads and writes, using a relational database may not suit your needs.
- **Need a flexible schema**: Flexible schema can allow for columns to be added that do not have to be used by every row, saving disk space.
- **Need high availability**: The fact that relational databases are not distributed (and even when they are, they have a coordinator/worker architecture), they have a single point of failure. When that database goes down, a fail-over to a backup system occurs and takes time.
- **Need horizontal scalability**: Horizontal scalability is the ability to add more machines or nodes to a system to increase performance and space for data.

1.1.3 ACID Transactions

Properties of database transactions intended to **guarantee validity even in the event of errors or power failures.**

- **Atomicity** the **whole transaction is processed or nothing is processed**. A commonly cited example of an atomic transaction is money transactions between two bank accounts. The transaction of transferring money from one account to the other is made up of two operations. First, you have to withdraw money in one account, and second you have to save the withdrawn money to the second account. An atomic transaction, i.e., when either all operations occur or nothing occurs, keeps the database in a consistent state. This ensures that if either of those two operations (withdrawing money from the 1st account or saving the money to the 2nd account) fail, the money is neither lost nor created.
- **Consistency** Only transactions that **abide by constraints and rules are written into the database**, otherwise the database keeps the previous state. The data should be correct across all rows and tables.
- **Isolation Transactions are processed independently and securely**, order does not matter. A low level of isolation enables many users to access the data simultaneously, however this also increases the possibilities of concurrency effects (e.g., dirty reads or lost updates)
- **Durability** Completed transactions are saved to database even in cases of system failure. A commonly cited example includes tracking flight seat bookings. So once the flight booking records a confirmed seat booking, the seat remains booked even if a system failure occurs.

1.1.4 PostgreSQL

Open source Object-relational database systems. Uses SQL but PostgreSQL syntax is different from other SQL syntax.

1.2 NoSQL or Non-Relational Databases

NoSQLs have:

- A simpler Design.
- simpler horizontal scaling
- finer control of availability.
- Data Structures used are different than those in Relational Database to make some operations faster.
- Created to deal with issues faced with SQL databases. As described in 1.1.2

Common Types of NoSQL DB

Apache Cassandra	Partion Row Store, Data is distributed by partitions.
MongoDB	Key+search in documents stored based on content.
DynamoDB	Key-Value Store. Collection of KV pairs
Apache HBase	Tables with rows and columns with Flexible schema.
Neo4J	Graph database for relationships (Nodes and Edges)

1.2.1 Apache Cassandra

Provides scalability and high availability without compromising performance.

- Linear Scalability
- proven fault tolerance on commodity hardware or cloud infrastructure. Make Apache Cassandra perfect for mission-critical data. **Apache Cassandra uses its own query language CQL**

What type of companies use Apache Cassandra?

- Uber uses Apache Cassandra for their entire backend
- Netflix uses Apache Cassandra to serve all their videos to customers

Good use cases for NoSQL (and more specifically Apache Cassandra) are :

- Transaction logging (retail, health care)
- Internet of Things (IoT)
- Time series data item Any workload that is heavy on writes to the database (since Apache Cassandra is optimized for writes).

Would Apache Cassandra be a hindrance for my analytics work? If yes, why?

Yes, if you are trying to do analysis, such as using GROUP BY statements. Since Apache Cassandra requires data modeling based on the query you want, you can't do ad-hoc queries. However you can add clustering columns into your data model and create new tables.

When to use a NoSQL Database?

- **Large amounts of data** The more servers/systems you add to the database the more data that can be hosted with high availability and low latency (fast reads and writes).
- **Need horizontal scalability** - Horizontal scalability is the ability to add more machines or nodes to a system to increase performance and space for data
- **Need high throughput –fast reads** While ACID transactions bring benefits they also slow down the process of reading and writing data. If you need very fast reads and writes using a relational database may not suit your needs.
- **Need flexible schema** Flexible schema can allow for columns to be added that do not have to be used by every row, saving disk space.
- **Need high availability** Relational databases have a single point of failure. When that database goes down, a failover to a backup system must happen and takes time.
- **Need different data type formats** NoSQL was also created to handle different data configurations: structured, semi-structured, and unstructured data. JSON, XML documents can all be handled easily with NoSQL.
- **Users are distributed** –low latency for geographically distributed users. **NoSQL was build to overcome consequences of Relational DBs and for Big Data.**

When NOT to use a NoSQL Database?

- **Need ACID Transaction** If you need a consistent database with ACID transactions, then most NoSQL databases will not be able to serve this need. NoSQL database are eventually consistent and do not provide ACID transactions. However, there are exceptions to it. Some non-relational databases like MongoDB can support ACID transactions.
- **Need JOINS** NoSQL does not allow the ability to do JOINS. This is not allowed as this will result in full table scans.
- **Need Aggregations or Analytics**
- **Changing Business requirements** Ad-hoc queries are possible but difficult as the data model was done to fix particular queries
- **Queries are not available** You need your queries in advance. If those are not available or you will need to be able to have flexibility on how you query your data you might need to stick with a relational database.
- **Small Dataset** NoSQL databases were made for big datasets not small datasets and while it works it wasnt created for that.

1.2.2 NoSQL and ACID

There are some NoSQL databases that offer some form of ACID transaction. As of v4.0, MongoDB added multi-document ACID transactions within a single replica set. With their later version, v4.2, they have added multi-document ACID transactions in a sharded/partitioned deployment.

Another example of a NoSQL database supporting ACID transactions is MarkLogic.