Data Modeling Overview

Saturday, April 6, 2019 3:00 AM

Data Modeling with SQL

What is data modeling?

At a high level, an abstractions of how data is organized, and how data elements relate to each other. This then translates into modeling for a parictuar system.

Examples:

- Excel Columns
- RDBMS

Balancing the needs of the business and the needs of the application systems (eg how much was sold or information about a customer logging into a website respectively)

Process:

- Gather Requirements
- Conceptual Data Modeling (aka ERD, white boarding etc)
    - What are the nouns and verbs
- Logical Data Modeling (Tables, Schemas, Columns)
- Physical Data Modeling (DDL, telling the database how to build the logical elements)
    - Most systems do this about 95% the same

Couse is mostly focused on the logical and physical portions

Key points about Data Modeling

- **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.

Everyone who works with data needs to understand how to model it (developers, data engineers, BI etc)

Relational Data Modeling

- Tables (Columns and Rows) (a relation)
    - Typically a entity maps to a table (in the logical model)
- Each row has an identifying key
- Tables relate to each other via keys
- RDBMS implements this, developed by Codd @ IBM
- SQL is the language used to interact

Advantages of Using a Relational Database

- **Flexibility for writing in SQL queries:** With SQL being the most common database query language.
- **Modeling the data not modeling queries**
- **Ability to do JOINS (via relationships)**
- **Ability to do aggregations and analytics**
- **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**

ACID Transactions

Properties of database transactions intended to guarantee validity even in the event of errors, 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 1st account and saving the money to the 2nd account) fail, the money is neither lost nor created. Source Wikipedia

- **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. Check out additional information about consistency on Wikipedia.

- **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). On the other hand, a high level of isolation reduces these chances of concurrency effects, but also uses more system resources and transactions blocking each other. Source: Wikipedia

- **Durability:** Completed transactions are saved to database even of 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. Source: Wikipedia.

When Not to Use a 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.

RDBMS are not distributed, so there is overhead

PostgreSQL

- RDBMS
- Object Oriented (aka complex types)

NoSQL DBs

- Simpler Design
- Horizontal Scaling
- Finer control of availability
- Designed to serve specific queries

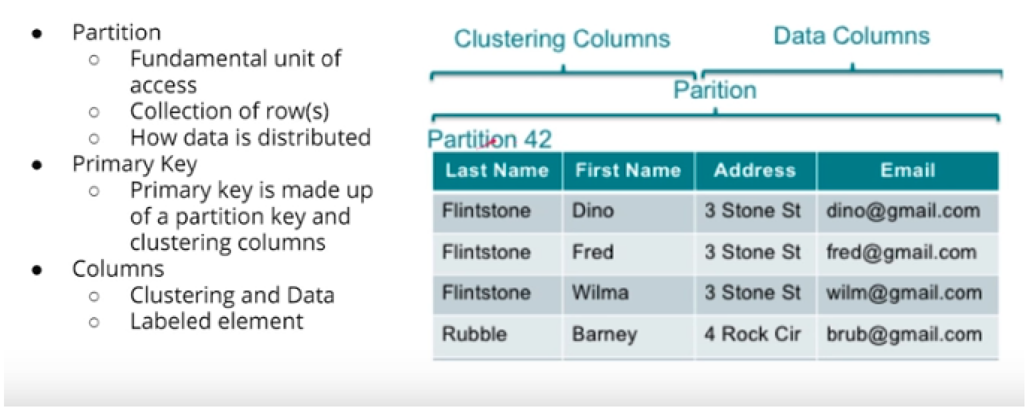
Exited since the 70's, but became common in the 2000's when data volumes grew

Casandra – Partitioned Row Store
MongoDB – Document Store
DynamoDB – Key Value Store
HBase – Wide Column Store
Neo4j – Graph Db

Apache Cassandra

Terms:

- Key space – collection of tables
- Table – a group of partitions
- Row – a single item
- Partition – Fundamental until of access, a group of rows, and how data is distributed
- Primary Key – Partition key + Clustering Columns
- Columns – Clustering or Data


Clustering Columns
Data Columns
Parition
Partition 42

- Partition
  - Fundamental unit of access
  - Collection of row(s)
  - How data is distributed
- Primary Key
  - Primary key is made up of a partition key and clustering columns
- Columns
  - Clustering and Data
  - Labeled element

Last Name	First Name	Address	Email
Flintstone	Dino	3 Stone St	dino@gmail.com
Flintstone	Fred	3 Stone St	fred@gmail.com
Flintstone	Wilma	3 Stone St	wilm@gmail.com
Rubble	Barney	4 Rock Cir	brub@gmail.com

Benefits:

- Horizontally Scalable

- Fault Tolerant
- High Availability

Uses its own query language, CQL

What type of companies use Apache Cassandra?

1. Netflix (video serving)
2. Transaction logging (retail, health care)
3. IOT
4. Time series data
5. Any workload that is heavy on writes to the database (since Apache Cassandra is optimized for writes).

When to use a NoSQL Database

- **Need to be able to store 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.
- **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. NoSQL databases were created to be able to be horizontally scalable. 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**: 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**: Relational databases have a single point of failure. When that database goes down, a failover to a backup system must happen and takes time.