Designing Data-Intensive Applications

Chapter 2: Data Models and Query Languages

Introduction

- Most applications are built by layering one data model on top of another
    - App code -> database -> bytes -> electrical current
- They are central to how problems are solved and thought about

Relational Model Versus Document Model

- SQL - data organized into relations
    - Helped hide the underlying implementation details
- Network/Hierarchical Model

The Birth of NoSQL

- Aim to handle greater scalability and very high write throughput
- Prefer free open source software over commercial
- Special queries that allow additional flexibility

Object-Relational Mismatch

- Impedance mismatch - Sometimes the structure of SQL might not match reality of application code
- People might have multiple previous/current jobs, but only stored in a single column in a structured way or require extra work to organize
- Document oriented DB's might alleviate some of these problems

Many-to-One and Many-to-Many Relationships

- Sometimes normalization/collapsing duplicates can be useful at simplifying relationships
    - Might have many ways to represent the same area with text
    - Areas might change name and can be trivially updated across all records
    - Might add complexity of mapping many text inputs to IDs

Are Document Databases Repeating History?

- IMS from IBM preceded these systems and had similar difficulties
- Network model - CODASYL allows a record to have multiple parents
    - "Greater Seattle Area" could be linked to every user in that region
    - Had to access individual record with access paths. Kind of like traversing a linked list
- Relational model
    - Table is just a collection of tuples(rows)
    - Much simpler read operations on columns and rows
    - Query optimizer deals with finding optimal way to execute commands on underlying data

Relational Versus Document Databases Today

- Document model favors flexibility, better performance via locality, closer to application representation

- Relational model supports joins and many-to-one and many-to-many relationships more natively
- Which data model leads to simpler application code?
    - Depends on the underlying structure and usage of the data
- Document model allows data to go into DB without a schema
    - Schema is applied on read
    - Can be useful with heterogeneous data when new unseen things may be added and dealt with later
    - Can lead to additional cost if whole document needs to be retrieved but only a small sliver of it is actually of interest
- In contrast to relational model with schema on write
    - Schema changes can be difficult
    - Relational DBs have started supporting XML/JSON to capture some of the document benefits
- Similar to static vs dynamic typing in programming

Query Languages for Data

- Declarative
    - Specify the data pattern and then query optimizer tries to find optimal way to find data that matches that pattern
    - More easily parallelized under the hood
- Imperative
    - Much more verbose but flexible. Coding the specific operations
    - Parallel code harder to write for each query

Declarative Queries on the Web

- Example showing declarative vs imperative with CSS/XSL vs Javascript
    - CSS/XSL(Declarative) much more concise on this task than Javascript(imperative)

MapReduce Querying

- In between imperative and declarative
- Write two separate tasks, map and reduce
    - Must be pure functions, not requiring any external info or operations to a single row

Graph-Like Data Models

- Better at handling many-to-many relationships
- FB has one huge graph with vertices representing people, locations, events, etc and edges indicate connections between those
- Property graphs and triple store to represent these graphs
- Cypher, SPARQL, Datalog for querying these graphs

Property Graphs

- Vertex
    - Identifier
    - Out/in edges
    - Collection of properties
- Edge

        - Identifier
        - Starting point of edge
        - End point of edge
        - Label to describe relationship
        - Collection of properties
    - No real schema, everything can be interconnected
    - Can go in both directions, inbound and outbound
    - Good for nested things like street, city, state, country etc.

Cypher Query Language

- Query formatted like find all people who have both “born in”=US and “lives in”=EU
- Can be requested in multiple different ways

Graph Queries In SQL

- Much more verbose and requires recursive calls

Triple-Stores and SPARQL

- Subject predicate object - Jim likes bananas
- RDF(resource description framework) created on top of this concept to make the semantic web so many sites could interact and share information, but has not gained great traction
- Similar query format to cypher

The Foundation: Datalog

- Similar to triple-store but older