Chapter 2 Data Model and Query Languages

Introduction

- Data models deeply affects how we think about the problem.
- Data models are built by layering one on top of another. The key question is: “how is it represented in terms of the next-lower layer?” Each layer hides the complexity of the layers below by providing a clean model.
- Every data model embodies assumptions on how it is going to be used. The data model has a profound effect on what the software above can or cannot do.
- In this chapter, we will compare the relational model, the document model, and a few graph-based models. We will also look at and compare various query languages.

Relational Model Versus Document Model

- The goal of the relational model is to hide implementation details behind a clean interface.
- SQL was rooted in relational databases for business data processing in the 1960s and 1970s, and was used for transaction processing and batch processing.
- SQL was proposed in 1970 and is probably the best-known data model, where data is organized into **relations** (tables), which is an unordered collection of **tuples** (rows).
- There were also the network model and the hierarchical model in the 1970s and 1980s, but the relational model dominated them.
- Relational database management systems (RDBMSes) and SQL became the tool of choice around mid-1980s.
- Object databases came and went around the 1990s, and XML databases appeared in early 2000s. However, neither really overtook the relational model.
- The relational model turned out to generalize very well.

NoSQL

- NoSQL was just a catchy hashtag on Twitter for a meetup. NoSQL is the latest attempt to overthrow the relational model’s dominance.
- Driving forces for NoSQL
    - A need for better scalability (larger datasets, high write throughput)
    - Preference for free and open source software
    - Specialized query operations that are not supported by the relational model
    - The restrictiveness of the relational schemas.
- It’s likely that relational databases will continue to be used along with many nonrelational databases.

The Object-Relational Mismatch

- The OOP paradigm requires translation between the relational database model.
- Strategies to deal with the mismatch
    - Normalized databases with foreign keys.
    - Use a database that Supports for structured data (PostgreSQL)
    - Encode as JSON or XML and store as text in database. It can’t be queried this way.
    - Store as JSON in a document in document-oriented databases (MongoDB). This has a better locality than the normalized representation.

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

- For *enum*-type strings, we can store a separate normalized ID to string table, and use the ID in other parts of the database. This will enforce consistency (same spelling, better search), avoid ambiguity, be easier to update, support localization.
- Using ids reduces duplication. This is the key idea behind normalizing databases. Yet this requires a many-to-one relationship, which may not work well with document databases, whose support for joins are weak.

Are Document Databases Repeating History

- In 1970s, the IBM's Information Management System also used a hierarchical model, similar to the JSON model. Data was represetned as a tree of records, nested within records, and the *network model* and *relational model* was proposed to solve the limitations of the hierarchical model.

The network model

- Also known as the CODASYL (Conference on DAta Systems Languages) model, the network model allows multiple parents for each record. The lines between records were not foreign keys, but more like pointers (called access path), which a user can follow through.
- A query is done by moving a cursor through the database by iterating over lists of records and following access paths.
- Although it made the most efficient use of the hardware, the model made querying and updating the database complicated and inflexible.

The relational model

- The relational model is simply a relation (table) and a collection of tuples (rows).
- In a relational database, the *query optimizer* automatically decides how to efficiently execute the query, where the developer has to decide the access path in the network model.
- Query optimizers will only need to be run once, and all applications using the database can benefit from it.

Comparison to document databases

- For one-to-many relationships, document databases are similar to hierarchical model in that they store nested records within their parent record rather than separate records.
- For many-to-one and many-to-many relationships, document databases are similar to relational databases. The related item is referenced by a *document reference* (in document databases) and a *foreign key* (in a relational database).

Relational Versus Document Databases Today

- We then compare the data model of relational and document databases.
    - Document databases: better schema flexibility, better performance due to locality, and closer to data structures in applications.
    - Relational databases: better support for joins, and many-to-one and many-to-many relationships.

Which data model leads to simpler application code?

- Document model may be a good choice if the application has a document-like structure:tree of one-to-many relationships, while relational models may require splitting a document-like structure into multiple tables.
- Records in document models are more difficult to directly access when they are deeply nested.

- For applications with many-to-many relationships, document models are less appealing due to the poor support for joins.
- For highly interconnected data, document model is awkward, and the relational model is acceptable, and graph models are the most natural.

Schema flexibility in the document level

- XML support in relational databases usually comes with schema validation while JSON support does not.
- Document databases are sometimes called *schemaless*, yet there is an implicit schema. A more accurate term is schema-on-read, meaning schema is only interpreted when data is read, in contrast with schema-on-write for relational database, where validation occurs during write time.
- An analogy to type checking is dynamic type checking (runtime) and static typing checking (compile-time). In general there is no right or wrong answer.
- To change a schema in document databases, applications would start writing new documents with the new schema, while in relational databases, one would need a migration query (which is quick for most databases, except MySQL).
- Schema-on-read is advantageous if items in the collection don't all have the same structure.
- Schema-on-write is advantageous when all records are expected to have the same structure.

Data locality for queries

- If the whole document (string of JSON, XML) is required by application often, there is a performance advantage to this storage locality (single lookup, no joins required). If only a small portion of the data is required, this can be wasteful.
- It's generally recommended to keep documents small and avoid writes that increases the size of documents (same size updates can be done in-place).
- Google's Spanner database allows table rows to be nested in a parent table. Oracle's database allows multi-table index cluster tables. The column-family concept (used in Cassandra and HBase) has a similar purpose for managing locality.

Convergence of document and relational databases

- Most relational databases (except MySQL) supports XML. PostgreSQL, MySQL, DB2 supports JSON.
- RethinkDQ supports joins, and some MongoDB drivers resolves database references.
- A hybrid of relational and document models is probably the future.

Query Language for Data

- SQL is a declarative query language, while IMS (IBM's Information Management System) and CODASYL are queried by imperative code.
- In a declarative language, only the goal is specified, not how the goal is achieved. It hides implementation details and leaves room for performance optimization.
- The fact that SQL is more limited in functionality gives databases more room for automatic optimizations.
- Declarative code is easier to parallelize as execution order is not specified.

Declarative Queries on the Web

- CSS and XSL are declarative languages to specify styling in HTML, while changing styles directly through Javascript is imperative.
- Specifying styles using CSS is much better than changing styles directly using Javascript.

MapReduce Querying

- MapReduce is a programming model for processing large amounts of data in bulk across many machines, and a limited form of MapReduce is supported by MongoDB and CouchDB.
- MapReduce is somewhere between declarative and imperative.
- SQL can, but not necessarily have to, be implemented by MapReduce operations.
- MongoDB supports a declarative query language called *aggregation pipeline*, where users don't need to coordinate a map and reduce function themselves.

Graph-Like Data Models

- An application with mostly one-to-many relationships or no relationships between records, the document model is appropriate, but for many-to-many relationships, graph models are more appropriate.
- Graphs can be homogeneous, where all vertices represent the same type of object and heterogeneous, where vertices may represent completely different objects.

Property Graphs

- In the property graph model, each vertex consists of
  - A unique identifier
  - A set of outgoing edges
  - A set of incoming edges
  - A collection of properties (key-value pairs)
- Each edge consists of
  - A unique identifier
  - The vertex at which the edge starts (the tail vertex)
  - The vertex at which the edge ends (the head vertex)
  - A collection of properties (key-value pairs)
- The property graph model is analogous to storing two relational databases, one for vertices, one for edges.
- Multiple relationships can be stored within the same graph by proper labeling the edges.
- Graphs can be easily extended to accommodate new data types.

The Cypher Query Language

- Cypher is a declarative query language for property graphs, created for the Neo4j graph database.

Graph Queries in SQL

- Usually, we know which joins to run in relational database queries, yet in a graph query, the number of joins is not fixed in advance, as we may need to follow an edge multiple times.

Triple-Stores and SPARQL

- In a triple-store, all information is stored in a three-part statement: *subject*, *predicate*, *object*.
- The subject is equivalent to a vertex in a graph.
- The object can be:
  - A value in a primitive datatype. In this case, the predicate and object forms the key and value of a property of the subject vertex.
  - Another vertex in the graph. In this case, the predicate is the edge between the subject vertex and the object vertex.

The semantic web

- The triple-store data model is completely independent of the semantic web.

- The main idea of semantic web is to have webpages publish data in a consistent format: Resource Description Framework (RDF). Unfortunately, the semantic web hasn't been realized in practice much.

The RDF data model

- RDF can be written in Turtle/N3 or XML.
- Namespaces are important in RDFs to distinguish the same word across different domains.

The SPARQL query language

- SPARQL is a query language for triple-stores using the RDF data model.

Comparing Graph Databases and the Network Model

CODASYL	Graph Databases	
Nested records	Specifies which record type can be nested within which other record type	No limitations
Access	Requires traversals to reach a particular record	Can use unique ID or index vertices in advance
Children of record	Has to be ordered	No limitations
Query language	Imperative code	Declarative languages

The Foundation: Datalog

- Datalog is a much older language than SPARQL or Cypher.
- It writes relationship (similar to the triple-store model) as *predicate(subject, object)*.
- One can compose new rules from simple rules (it can be recursive!).
- Words start with upper and lower cases are variables and predicates, respectively.
- New rules that has all simple rules matched are also added to the database.

Summary

Pros	Cons	
Hierarchical	Straightforward for one-to-many relationships	Not good for many-to-many or many-to-one relationships, imperative code for querying.
Network	Straightforward for many-to-many relationships	Imperative code for querying
Relational (SQL)	Generalize well	Throughput scalability, restrictiveness of relational schema (harder to extend), object-relational mismatch.
Document (NoSQL)	Throughput scalability, good for self contained documents, easier to extend.	Not good for data with many cross-document relationships.
Graph (NoSQL)	Good for many-to-many relationships, easier to extend..	Harder to do sum and max queries. Link

- Document, relational, and graph are all widely used. Although we can emulate one in another, it could be awkward.
- Next chapter will discuss tradeoff during implementing those data models.