Part VI: Choosing a Database for Your Application

15. Guidelines for Selecting a Database

> *"One's philosophy is not best expressed in words; it is expressed in the choices one makes."*
>
> —ELEANOR ROOSEVELT
> POLITICIAN, ACTIVIST, AND DIPLOMAT

Topics Covered In This Chapter

Developers have never had as many good database options as they have today. Relational databases have a long and proven track record of successful use in a wide range of applications. These databases have been so successful they virtually eliminated the widespread use of earlier database models, such as file-based, hierarchical, and network databases. It was not until the advent of commercial web systems, such as search engines, that relational databases strained to meet developers' demands.

The growing demands for web-scale data management systems drove a renaissance in nonrelational database design. Yahoo! developed Hadoop. Google created BigTable. Amazon designed and deployed DynamoDB. Instead of keeping their intellectual property locked up in a corporate vault somewhere, these companies published papers and, in some cases, released code for others to build on. This in turn allowed other developers to build on those designs and expand the ecosystem of NoSQL databases and supporting tools.

If you are a developer starting a data management project today, you will have to decide which type of database management system to use. Your major options are

- Relational databases, such as PostgreSQL, MySQL, and Microsoft SQL Server
- Key-value databases, such as Redis, Riak, and Oracle BerkeleyDB
- Document databases, such as MongoDB, CouchDB, and CouchBase
- Column family databases, such as Cassandra and HBase
- Graph databases, such as Neo4j and Titan

❖ **Note**

Discussing relational database design in any detail is outside the scope of this book, but if you think a relational database might be an option for you, see Michael Hernandez's, *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design* for guidance.

For those best served by a NoSQL option, this chapter includes some points to consider as you evaluate your options.

Choosing a NoSQL Database

In relational database design, the structure and relations of entities drives design—not so in NoSQL database design. Of course, you will model entities and relations, but performance is more important than preserving the relational model.

The relational model emerged for pragmatic reasons, that is, data anomalies and difficulty reusing existing databases for new applications. NoSQL databases also emerged for pragmatic reasons, specifically, the inability to scale to meet growing demands for high volumes of read and write operations.

In exchange for improved read and write performance, you may lose other features of relational databases, such as immediate consistency and ACID transactions (although this is not always the case).

Throughout this book, queries have driven the design of data models. This is the case because queries describe how data will be used. Queries are also a good starting point for understanding how well various NoSQL databases will meet your needs. You will also need to understand other factors, such as

- The volume of reads and writes
- Tolerance for inconsistent data in replicas
- The nature of relations between entities and how that affects query patterns
- Availability and disaster recovery requirements
- The need for flexibility in data models
- Latency requirements

The following sections provide some sample use cases and some criteria for matching different NoSQL database models to different requirements.

Criteria for Selecting Key-Value Databases

Key-value databases are well suited to applications that have frequent small reads and writes along with simple data models. The values stored in key-value databases may be simple scalar values, such as integers or Booleans, but they may be structured data types, such as lists and JSON structures.

Key-value databases generally have simple query facilities that allow you to look up a value by its key. Some key-value databases support search features that provide for somewhat more flexibility. Developers can use tricks, such as enumerated keys, to implement range queries, but these databases usually lack the query capabilities of document, column family, and graph databases.

Key-value databases are used in a wide range of applications, such as the following:

- Caching data from relational databases to improve performance
- Tracking transient attributes in a web application, such as a shopping cart
- Storing configuration and user data information for mobile applications

- Storing large objects, such as images and audio files

❖ **Note**

In addition to key-value databases you install and run on the premises, there are a number of cloud-based choices as well. Amazon Web Services offers SimpleDB and DynamoDB, whereas Microsoft Azure's Table service provides for key-value storage.

Use Cases and Criteria for Selecting Document Databases

Document databases are designed for flexibility. If an application requires the ability to store varying attributes along with large amounts of data, then document databases are a good option. For example, to represent products in a relational database, a modeler may use a table for common attributes and additional tables for each subtype of product to store attributes used only in the subtype of product. Document databases can handle this situation easily.

Document databases provide for embedded documents, which are useful for denormalizing. Instead of storing data in different tables, data that is frequently queried together is stored together in the same document.

Document databases improve on the query capabilities of key-value databases with indexing and the ability to filter documents based on attributes in the document.

Document databases are probably the most popular of the NoSQL databases because of their flexibility, performance, and ease of use.

These databases are well suited to a number of use cases, including

- Back-end support for websites with high volumes of reads and writes
- Managing data types with variable attributes, such as products
- Tracking variable types of metadata
- Applications that use JSON data structures
- Applications benefiting from denormalization by embedding structures within structures

Document databases are also available from cloud services such as Microsoft Azure Document and Cloudant's database.

Use Cases and Criteria for Selecting Column Family Databases

Column family databases are designed for large volumes of data, read and write performance, and high availability. Google introduced BigTable to address the needs of its services. Facebook developed Cassandra to back its Inbox Search service.

These database management systems run on clusters of multiple servers. If your data is small enough to run with a single server, then a column family database is probably more than you need—consider a document or key-value database instead.

Column family databases are well suited for use with

- Applications that require the ability to always write to the database
- Applications that are geographically distributed over multiple data centers
- Applications that can tolerate some short-term inconsistency in replicas
- Applications with dynamic fields
- Applications with the potential for truly large volumes of data, such as hundreds of terabytes

Google demonstrated the capabilities of Cassandra running the Google Compute Engine.[1] Google engineers deployed

- 330 Google Compute Engine virtual machines
- 300 1TB Persistent Disk volumes
- Debian Linux
- Datastax Cassandra 2.2
- Data was written to two nodes (Quorum commit of 2)
- 30 virtual machines to generate 3 billion records of 170 bytes each

1. Google. 2014, March 20. “Cassandra Hits One Million Writes Per Second on Google Compute Engine.” http://googlecloudplatform.blogspot.com/2014/03/cassandra-hits-one-million-writes-per-second-on-google-compute-engine.html

With this configuration, the Cassandra cluster reached one million writes per second with 95% completing in under 23 milliseconds. When one-third of the nodes were lost, the one million writes were sustained but with higher latency.

Several areas can use this kind of Big Data processing capability, such as

- Security analytics using network traffic and log data mode
- Big Science, such as bioinformatics using genetic and proteomic data
- Stock market analysis using trade data
- Web-scale applications such as search
- Social network services

Key-value, document, and column family databases are well suited to a wide range of applications. Graph databases, however, are best suited to a particular type of problem.

Use Cases and Criteria for Selecting Graph Databases

Problem domains that lend themselves to representations as networks of connected entities are well suited for graph databases. One way to assess the usefulness of a graph database is to determine if instances of entities have relations to other instances of entities.

For example, two orders in an e-commerce application probably have no connection to each other. They might be ordered by the same customer, but that is a shared attribute, not a connection.

Similarly, a game player's configuration and game state have little to do with other game players' configurations. Entities like these are readily modeled with key-value, document, or relational databases.

Now consider examples mentioned in the discussion of graph databases, such as highways connecting cities, proteins interacting with other proteins, and employees working with other employees. In all of these cases, there is some type of connection, link, or direct relationship between two instances of entities.

These are the types of problem domains that are well suited to graph databases. Other examples of these types of problem domains include

- Network and IT infrastructure management
- Identity and access management
- Business process management
- Recommending products and services
- Social networking

From these examples, it is clear that when there is a need to model explicit relations between entities and rapidly traverse paths between entities, then graph databases are a good database option.

Large-scale graph processing, such as with large social networks, may actually use column family databases for storage and retrieval. Graph operations are built on top of the database management system. The Titan graph database and analysis platform takes this approach.

Key-value, document, column family, and graph databases meet different types of needs. Unlike relational databases that essentially displaced their predecessors, these NoSQL databases will continue to coexist with each other and relational databases because there is a growing need for different types of applications with varying requirements and competing demands.

Using NoSQL and Relational Databases Together

NoSQL and relational databases are complementary. Relational databases offer many features that protect the integrity of data and reduce the risk of data anomalies. Relational databases incur operational overhead providing these features.

In some use cases, performance is more important than ensuring immediate consistency or supporting ACID transactions. In these cases, NoSQL databases may be the better solution. Choosing a database is a process of choosing the right tool for the job. The more varied your set of jobs, the more varied your toolkit.

Modern data management infrastructure is responsible for a wider range of applications and data types than ever before. When E. F. Codd developed the relational model in the 1970s, businesses and governments were the primary users of databases.

The personal computer, smartphones, and tablets did not exist. The Internet was used by government and academic researchers; the World Wide Web was almost 20 years into the

future. The Global Positioning System (GPS) was not fully operational until 1995.

Today, IT professionals are working with more of the same types of business data that existed in the 1970s as well as new types, such as social media and detailed customer demographics and preference data.

Mobile devices generate large volumes of data about users' behaviors and location. The instrumentation of cars, appliances, and other devices, referred to as the Internet of Things (IoT), is another potential data source. With so many changes in the scope and size of data and applications, it is no surprise that additional database management techniques are needed.

Relational databases will continue to support transaction processing systems and business intelligence applications. Decades of work with transaction processing systems and data warehouses has led to best practices and design principles that continue to meet the needs of businesses, governments, and other organizations.

At the same time, these organizations are adapting to technologies that did not exist when the relational model was first formulated. Customer-facing web applications, mobile services, and Big Data analytics might work well with relational databases, but in some cases they do not.

The current technology landscape requires a variety of database technologies. Just as there is no best programming language, there is no best database management system. There are database systems better suited to some problems than others, and the job of developers and designers is to find the best database for the requirements at hand.

Summary

Application developers have choices about which programming language they use, which development environments they work in, and which web frameworks they deploy. They also have choices when it comes to database management systems. The different types of database management systems were all developed to solve real-world problems that could not be solved as well with other types of databases.

One of the jobs of developers and designers is to choose an appropriate database system for their applications. You do this by understanding your problem domain and your user requirements. Often you will have options. You could use a key-value store or a document database in some cases. Other times, a graph database might be the best fit. Do not be surprised if you find yourself working with key-value databases one day and graph databases the next. The choice of database should be driven by your needs.

Review Questions

1. Name two use cases for key-value databases.
2. Describe two reasons for choosing a key-value database for your application.
3. Name two use cases for document databases.
4. Describe two reasons for choosing a document database for your application.
5. Name two use cases for column family databases.

6. Describe two reasons for choosing a column family database for your application.
7. Name two use cases for graph databases.
8. Describe two reasons for choosing a graph database for your application.
9. Name two types of applications well suited for relational databases.
10. Discuss the need for both NoSQL and relational databases in enterprise data management.

References

Apache Foundation. Apache Cassandra Glossary: http://io.typepad.com/glossary.html

Apache Foundation. Apache CouchDB 1.6 Documentation: http://docs.couchdb.org/en/1.6.1/

Apache Foundation. Apache HBase Reference Guide: http://hbase.apache.org/book/regions.arch.html

Basho Technologies, Inc. Riak Documentation: http://docs.basho.com/riak/latest/

Bishop, Christopher M. *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

Chang, Fay, et al. "BigTable: A Distributed Storage System for Structured Data." OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006. http://research.google.com/archive/bigtable.html

Chodorow, Kristina. *50 Tips and Tricks for MongoDB Developers*. Sebastopol, CA: O'Reilly Media, Inc., 2011.

Chodorow, Kristina. *MongoDB: The Definitive Guide*. Sebastopol, CA: O'Reilly Media, Inc., 2013.

Copeland, Rick. *MongoDB Applied Design Patterns*. Sebastopol, CA: O'Reilly Media, Inc., 2013.

Couchbase. Couchbase Documentation: http://docs.couchbase.com/

FoundationDB. "ACID Claims": https://foundationdb.com/acid-claims

FoundationDB. Key-Value Store 2.0 Documentation: https://foundationdb.com/key-value-store/documentation/index.html

Gremlin: https://github.com/tinkerpop/gremlin/wiki

Han, Jing, et al. "Survey on NoSQL Database." Pervasive computing and applications (ICPCA), 2011 6th International Conference on IEEE, 2011.

Hewitt, Eben. *Cassandra: The Definitive Guide*. Sebastopol, CA: O'Reilly Media, Inc., 2010.

Katsov, Ilya. "NoSQL Data Modeling Techniques." *Highly Scalable Blog*. March 1, 2012. http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/

MongoDB. MongoDB 2.6 Manual: http://docs.mongodb.org/manual/

Neo Technologies, Inc. Neo4j Manual: http://neo4j.com/docs/

Oracle Corporation. Oracle NoSQL Database, 12c Release 1: http://docs.oracle.com/cd/NOSQL/html/index.html

OrientDB. OrientDB Manual, version 2.0: http://www.orientechnologies.com/docs/last/

Redis. Redis Documentation: http://redis.io/documentation

Titan Distributed Graph Database: http://thinkaurelius.github.io/titan/