RogueWave SOFTWARE

Accelerating Great Code

OVERCOMING RELATIONAL DATABASE LIMITATIONS WITH NOSQL

Relational databases are the workhorses of the modern database industry. They have limitations, however, when it comes to handling some types of data, in particular the large quantities of free-form data generated through mobile technology. NoSQL databases provide solutions for some of these problems but they introduce another problem in having no single query language that drives them. This paper examines the problem, surveys the solutions, and answers the question of how to implement the solutions through a consistent API.

LIMITATIONS OF RELATIONAL DATABASES

Traditionally, we have relied on relational database systems for storing data. Relational database systems provide data integrity and consistency by enforcing atomicity, consistency, isolation, and durability (ACID) properties. This is essential in many scenarios. For example, it avoids contention should an ATM withdrawal and a deposit transaction happen on the same account at the same time. The problem is that in many scenarios, such as caching shopping cart history, ACID properties are a significant performance overhead, which leads to problems with scalability.

Another aspect of many modern applications is that they work with unstructured data. Many applications use JSON to store their data. Relational database management systems (RDMS) don't provide an efficient way to provide create, read, update, and delete (CRUD) operations on this data.

NOSQL

NoSQL refers to a class of database management systems. NoSQL databases are not intended to replace relational databases but instead provide solutions where relational databases are not a good fit. They are often classified as:

1. **Key-value:** As the name suggests, these databases are intended to store key-value pairs. Key-value databases are designed to be fast, trading durability for raw speed[1]. While in traditional key-value data structures a string key is associated with a string value, in a key-value database, the value is not limited to a simple string but can also hold more complex data structures such as lists, sets, hashes, and bit arrays[2]. An example of a key-value database is Redis.
2. **Document based:** These databases are similar to key-value databases but store documents in the value part. Generally, the documents that are stored are in self-describing formats, such as XML, BSON, and JSON. An example of a key-document database is MongoDB.
3. **Column based:** These databases were created to store and process very large amounts of data distributed over many machines. There are still keys but they point to multiple columns. The columns are arranged by column family. Examples of column-based databases are Cassandra and HBase.
4. **Graph based:** In relational databases, references to other rows and tables are indicated by referring to their primary-key attributes via foreign-key columns. Relationships are first-class citizens of the graph data model and traversing the joins or relationships is very fast[3]. The relationship between nodes is not calculated at query time but is actually persisted as a relationship. An example of a graph-based database is Neo4j.

[1] "Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement", Eric Redmond and Jim R. Wilson, Lewisville, TX: The Pragmatic Programmers, 2012

[2] "An Introduction to Redis Data Types and Abstractions – Redis", accessed June 10, 2016.

[3] "Relational Databases Vs. Graph Databases: A Comparison," accessed June 10, 2016.

POLYGLOT PERSISTENCE

Polyglot persistence is where you can leverage the strengths of many kinds of databases in the same system. This has become necessary because different databases are designed to solve different problems. Using a single database engine for all of the requirements usually leads to non-performant solutions4. For example, an e-commerce application may use a key-value store for its shopping cart. Accessing a shopping cart doesn't require the overhead of transactions and ACID properties. The key aspect is to access the cart quickly. On the other hand, when the user checks out, the transactional data has to be secure and atomic. So a relational database is a better fit here. To store the transaction history, a document-based database may be a good choice. You can search it quickly and it scales well as the e-commerce application grows.

While the NoSQL approach seems a good solution to these issues, NoSQL databases don't have a common query language like relational databases. So interacting with different databases requires dealing with the complexity of working with different query languages and integrating them into an application.

THE SOURCEPRO DB SOLUTION

SourcePro DB has a proven track record for accessing relational databases using a high-level, database-independent C++ interface. Using the same interface, SourcePro DB can also interact with NoSQL databases, enabling polyglot persistence without the need to learn new query languages.

The following white papers demonstrate how to access MongoDB, Cassandra, and Redis using the SourcePro DB interface, including code samples.

- Using SourcePro with MongoDB
- Using SourcePro with Cassandra
- Using SourcePro with Redis

[4] "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence", Pramod J. Sadalage and Martin Fowler, United States: Addison-Wesley Educational Publishers, 2012.

Rogue Wave provides software development tools for mission-critical applications. Our trusted solutions address the growing complexity of building great software and accelerates the value gained from code across the enterprise. The Rogue Wave portfolio of complementary, cross-platform tools helps developers quickly build applications for strategic software initiatives. With Rogue Wave, customers improve software quality and ensure code integrity, while shortening development cycle times.