The Multi-Model Database

Cloud Applications in a Complex World

Table of Contents

Introduction

Cloud applications have been a disruptive force in every industry and have changed the way most companies do business. Enterprises that have embraced the cloud application business model are easy to spot because they are outperforming their competition, while those that haven't are struggling and trying desperately to catch up.

A cloud application is one with many endpoints including browsers, mobile devices, and/or machines that are geographically distributed, intensely transactional, continuously available, as well as instantaneously and intelligently responsive no matter the number of users or machines using the application. Properly satisfying a cloud application's data management requirements is no easy task and necessitates a different database than those used for the past thirty-plus years.

Part of the "new normal" where data and cloud applications are concerned is the ability for the underlying data layer to smartly handle multiple types of data models that exist in the application and persist each in a single datastore. This *adaptive data management* capability is sometimes called a "multi-model" database.

This paper explores the multi-model concept, its rationale, and how DataStax Enterprise (DSE) realizes the vision of a multi-model database that supports today's cloud applications.

Multi-Model: An Evolutionary Tale

Understanding the why's and how's of a multi-model database requires a brief look back at the various evolutions of data management that have brought things to where they are today.

The first evolution is one that has seen the move from very centralized database deployments to ones that are semi-decentralized. Today, that evolution has continued to deployments that are radically distributed. The last evolution came about as a direct result of must-have requirements for cloud applications (e.g. write, read, distribute, stay-active everywhere).

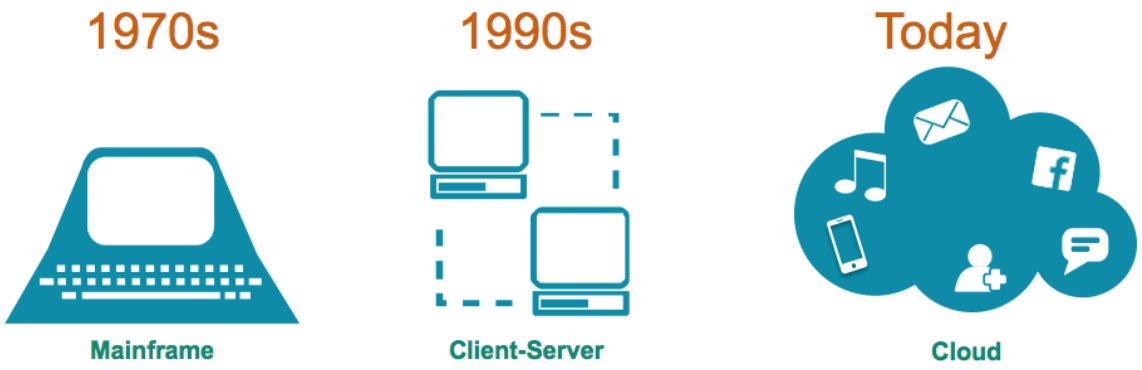

Figure 1 – Data evolution from heavily centralized to radically distributed data deployments.

The next two evolutions deal with the concept of polyglot persistence, which refers to the practice of using multiple data storage technologies to support either a single or multiple applications.

Initially, the idea of polyglot persistence came about via the need to separate different data management workloads so that there would be no competition for data or compute resources between systems devoted to transactional, analytics, search, and other workloads. It started with having distinct applications targeting separate datastores, which was fine for older applications. However, cloud applications demanded that the database evolve to handle the consolidation of workloads onto a single platform to support the ideas of hybrid transaction analytical processing (HTAP), also referred to by other names such as "translytics."

The third evolution resembles the second, but involves differing data models versus workloads. The progression began predominantly with the RDBMS (Relational DataBase Management System) data model moving to a mixture of different technologies and data models to, now, a unified platform capable of supporting multiple data models against a single integrated backend.

With the last two data evolutions, the goal for cloud applications has been to provide both mixed workload and multi-model capabilities in one platform.

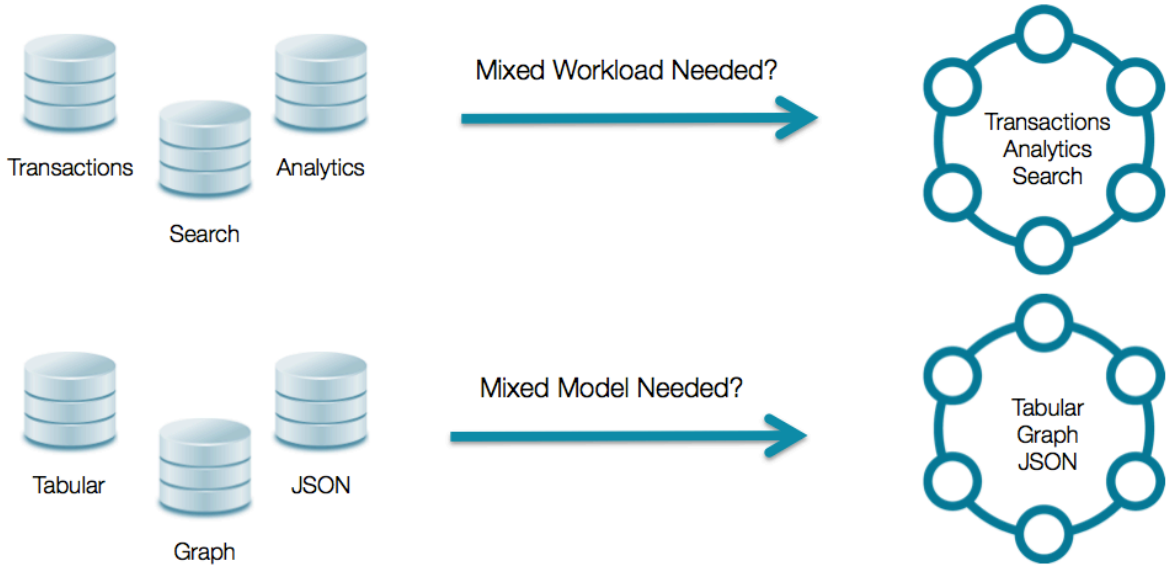

Figure 2 – An end goal for cloud applications: support differing workloads and data models in one platform.

From RDBMS to NoSQL to Multi-Model

The RDBMS ecosystem exhibits a few key cherished characteristics such as: a vendor agnostic standard language for the developers (SQL); well-defined separation between logical (developer) and physical (DBA) aspects of the DBMS; and a cohesive set of mechanisms in a variety of languages (drivers) by which applications can interact with the databases. This allows enterprises to adopt data infrastructure technologies without worrying about vendor lock-in.

However, while the above characteristics serve centralized applications well, there are impediments to its adoption in cloud applications:

- A master-slave architecture mandating concessions on uptime and resiliency
- Scale and write-and-distribute-anywhere constraints for cloud application workloads
- Strict adherence to logical data layer constructs such as third normal form (3NF), which value storage efficiency more than application agility
- A rigid data model that make the use of semi and unstructured data extremely cost prohibitive at scale
- The sharding architectural "Band-Aid" that increases operational expense exponentially

While certain NoSQL technologies like Apache Cassandra™ address the previously mentioned challenges of the RDBMS ecosystem, the NoSQL movement created a fragmented technology offering for enterprise customers due to the two issues addressed below.

First, polyglot persistence implied that customers would either use a limited set of one of the models (Key Value, Tabular, JSON, Graph) or perform extract-transform-load operations across data stores. Enterprise use cases such as master data management (MDM), customer-360-view and others, mandated the latter, which can increase the total cost of ownership (TCO) where NoSQL is concerned.

Second, each NoSQL vendor's mechanism for interacting with the data store was different, both with respect to its dialect and where they lay on the logical/physical divide. This forced application developers to write abstraction layers if they needed more than a single model in their application. Further, these abstraction layers had to work at very different level across the physical/logical spectrum to keep application development aligned.

Multi-model databases represent the next phase of maturity for the NoSQL industry that aim to address the hurdles of adoption, especially as it relates to mainstream developers and administrators within the enterprise. This is accomplished by supporting multiple data models against a single, integrated backend. Such a multi-model database platform should exhibit:

- Support for more than one post relational data model (e.g. tabular, JSON, graph) at a logical layer for ease of development for application developers
- Ensure all models are exposed via cohesive mechanisms thereby avoiding cognitive context switching for developers
- A unified persistence layer that delivers geo redundant, continuously available characteristics and a common framework for operational aspects such as security, provisioning, disaster recovery, etc.
- Empower a variety of use cases across OLTP and OLAP workloads for lines of businesses within an enterprise to innovate with agility
- Deliver best in case TCO efficiency for the long haul to enable wider adoption within centralized IT teams of an organization

DataStax Enterprise and Multi-Model

DataStax delivers a comprehensive data management layer in DataStax Enterprise that sports a unique always-on architecture that accelerates the ability of enterprises, government agencies, and systems integrators to power the exploding number of cloud applications. DSE well serves cloud applications that require data distribution across datacenters and clouds, through the use of its secure, operationally simple platform that is built on Apache Cassandra.

Version 5.0 and above of DSE support adaptive data management (i.e. multi-model) with all data being stored in Cassandra. No matter the data model(s) used, each receives the benefits of Cassandra's always-on, distribute-anywhere, active-anywhere, linear-scale architecture.

Further, each data model inherits all of DSE's commercial extensions including advanced security, built-in analytics, enterprise search, and visual administration and monitoring.

Deciding Which Data Models to Use for a Cloud Application

A cloud application is one that is intensely multi-faceted. For example, a modern retail cloud application includes various modules such as product catalogs, user profile management, fraud detection, recommendation engine, shopping cart, clickstream/log analysis, and others.

Each component of a cloud application may have distinct data model support requirements, but deciding which model to use can be confusing. One approach is to step back and ask what level of data complexity and connectedness is involved. Requirements that have little to no complexity or connectedness (i.e. data relationships) can be served by simpler data models while others having greater complexity and value in relationships are best handled by a graph data model.

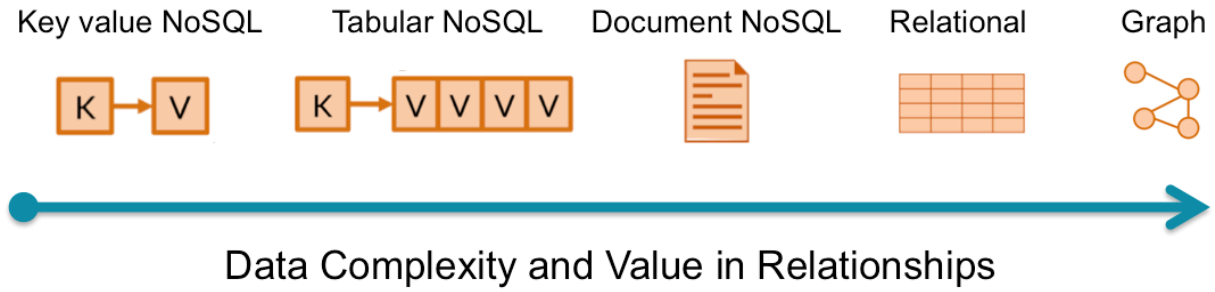

Figure 3 – The data model continuum represented by complexity and data connectedness.

DataStax Enterprise currently supports four different data models.

Key Value and Tabular

While Cassandra can still be used as a simple key-value datastore, it has come a long way from its original roots. Cassandra 3.0 and above support Materialized Views and other powerful functionality, which, along with the widespread adoption of the CQL dialect, presents Cassandra as more of a tabular store to the application developer.

The specific use cases handled by Cassandra's data model include almost all time-series data scenarios (e.g. IoT, user activity management, financial streaming) and other similar write-and-read heavy situations. As compared to a relational design, tables in Cassandra are typically modeled in a denormalized fashion around the queries needed to satisfy requests from the application.

JSON / Document

The JSON/Document model support in DSE has the flexibility to store data with complex nested schemas and is able to easily move data to and from application tiers – both of which are the primary use cases of popular Document-oriented databases. The major difference between DSE and some other databases that support JSON is the requirement that the JSON data adhere to a database schema.

In some schema-less document records, one record might vary significantly from the next, with no expectation that the database will enforce any record structure. DSE, however, retains the requirement that a CQL table holding JSON data be defined with a schema that implements columns and column types in a row. It should be noted that some document database vendors are now recognizing the need for schema enforcement for JSON data and have introduced mechanisms to validate data that is loaded into the database.

In DSE, CQL supports collection types (maps, sets and lists) and user-defined types that all map into JSON list and map types. This allows DSE to store records that have similar structural complexity to document-style records without being completely free form. Sparse storage allocation in the underlying storage engine also means that column values can be left null with no penalty, further adding to the flexible structure of the record.

It is possible to define columns that might only be used with a handful of rows in the table, which matches the ability for a JSON document to leave values out of a record. It is also possible to add columns to a table at any time without a performance penalty.

If the requirement is to deal with an incoming data feed that might change as time progresses, this can be handled with a no-cost change to the table schema. In this respect, DSE has many of the Document storage capabilities of pure Document databases except for the ability to handle truly schema-less data without prior knowledge of the structure of incoming records.

There are a variety of use cases that lend themselves to the use of JSON inside DSE. For example, DSE is often used as the “state store” for cloud applications written with JavaScript UI frameworks that are designed to exchange JSON between the browser and the server. With DSE’s support for JSON, the server-side code required to do that is dramatically simplified. Prior to JSON support, the application developer would have to convert CQL results to JSON before sending query results back to the client. Conversely, incoming data would have to be transformed from JSON into CQL statements. With JSON/Document data model support now in DSE, none of that code is required.

Lastly, JSON support is also present in DataStax DevCenter, which is a visual development tool for writing CQL and JSON queries against DSE. Smart JSON editors in DevCenter include syntax highlighting, code completion, code correction, and much more, which make developing with JSON against DSE simple and straightforward.

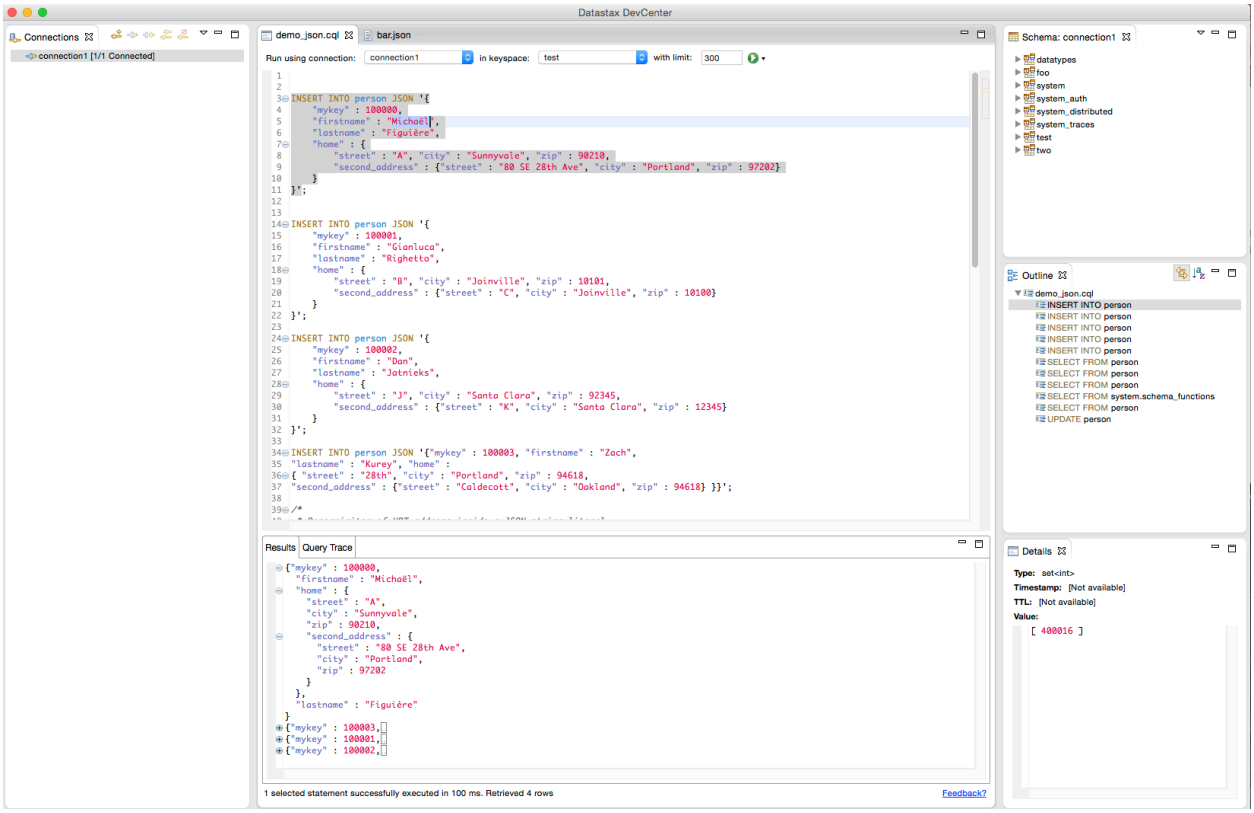

Figure 4 – Developing with JSON in DataStax DevCenter.

Graph

Graph data model support is realized in DSE through DSE Graph, which is a graph database built for cloud applications that need to manage highly complex and connected data. DSE Graph delivers continuous uptime along with predictable performance and scale for modern systems dealing with complex and constantly changing data, while remaining operationally simple to manage. Support for graph is present in the DSE Server, the DataStax OpsCenter management tool, DataStax Studio – which is a visual developer tool for graph – and lastly, DataStax drivers.

A graph model should be considered for any use case involving complex data scenarios that consist of intense and numerous relationships among the data elements. Since an RDBMS and a graph database are similar in that they involve data that contains connections or relationships between data elements, why not just use an RDBMS versus a graph data model like the one found in DSE?

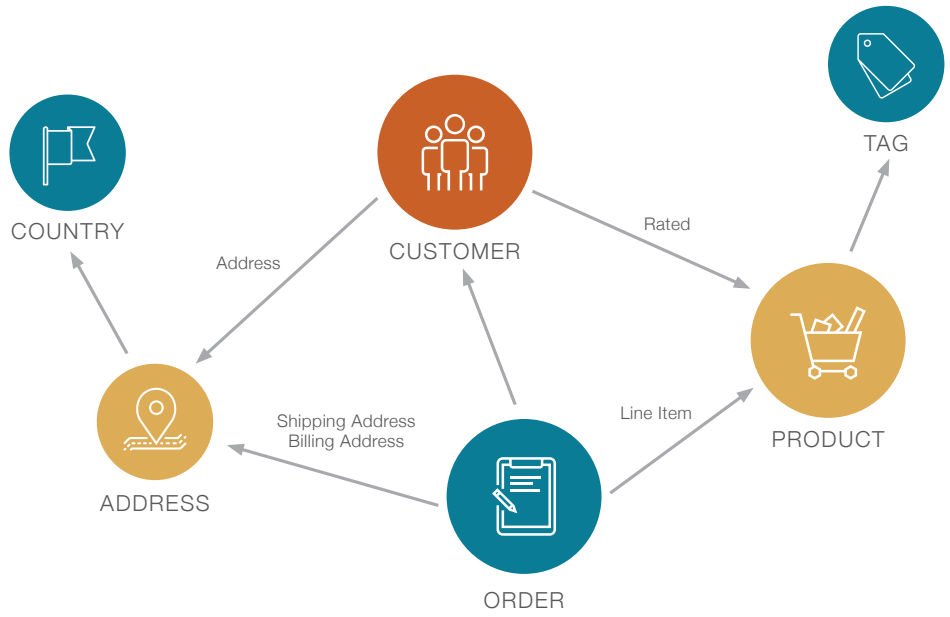

Figure 5 – A simple graph data model.

Foundationally an RDBMS and graph database differ in the underlying engine each uses to store and access data. There are also some data modeling features that RDBMS's and graphs don't share such as graph databases allowing characteristics (i.e. properties) to be assigned to edges, whereas RDBMS relationships have no such ability.

Another key difference between a graph database and an RDBMS is how relationships between entities/vertexes are prioritized and managed. While an RDBMS uses mechanisms like foreign keys to connect entities in a secondary fashion, edges (the relationships) in a graph database are of first order importance.

In other words, relationships are explicitly embedded in a graph data model. Essentially, a graph-shaped business problem is one in which the concern is with the relationships (edges) among entities (vertexes) than with the entities in isolation.

The following comparisons can be used to help in the decision making process of whether to use an RDBMS or a graph database like DSE Graph for a particular use case:

RDBMS	DSE Graph
Simple to moderate data complexity	Heavy data complexity
Hundreds of potential relationships	Hundreds of thousands to millions or billions of potential relationships
Moderate JOIN operations with good performance	Heavy to extreme JOIN operations required

Infrequent to no data model changes	Constantly changing and evolving data model
Static to semi-static data changes	Dynamic and constantly changing data
Primarily structured data	Structured and unstructured data
Nested or complex transactions	Simple transactions
Always strongly consistent	Tunable consistency (eventual to strong)
Moderate incoming data velocity	High incoming data velocity (e.g. sensors)
High availability (handled with failover)	Continuous availability (no downtime)
Centralized application that is location dependent (e.g. single location), especially for write operations and not just read	Distributed application that is location independent (multiple locations involving multiple data centers and/or clouds) for write and read operations
Scale up for increased performance	Scale out for increased performance

A graph database like DSE Graph will most times be better than an RDBMS when it comes to identifying commonalities and anomalies in large, complex, and highly connected datasets. While DSE Graph can be used for a variety of application use cases, the following are some of the most common that lend themselves to being managed by a graph database:

- Master Data Management: A graph is the best model for critical business data and their relationships that are consolidated across business units, which is then queried and maintained by various transactional and business intelligence (BI) business applications. Other examples include product catalogs, which often have complex hierarchical structures and are overlaid by taxonomies to capture composition or other relationships. In those cases, the data complexity is high enough to warrant a graph database. It is also typical for search functionality to be necessary, which can be handled with DSE Search.
- Recommendation/Personalization: Oftentimes, relevant recommendations can be best identified in a large graph of users and entity interactions. A graph is well suited to help recommend products, next actions, or advertising based on a user's information, past behavior, and interactions.
- Security Management and Fraud Detection: In a complex and highly interrelated network of users, entities, transactions, events, and interactions, a graph database can help determine which entity, transaction or interaction is fraudulent, poses a security risk, or is a compliance concern. In short, a graph database assists in finding the bad needle in a haystack of relationships and events.
- IoT, Network Asset Management and Monitoring: A graph is a good model for managing network assets (with their properties or configurations) and how they relate to each other over time. A graph can be used to manage and monitor the network, optimize resource allocation, detect and fix problems, etc. This can also include IoT use case where assets are devices or machines that generate time-series data (e.g. status records, event data). One way to approach the data management with DSE is to have the network asset information stored in DSE Graph and the IoT time-series data in Cassandra.

Conclusions

Because cloud applications involve numerous components that usually differ in their data model support requirements, a database that provides adaptive data management (or multi-model) capabilities will deliver a simpler and more agile solution for quickly bringing cloud applications to market. DataStax Enterprise has built-in multi-model capabilities and provides support for key-value, tabular, JSON / document, and graph data models.

Because data from all models are stored in a single backend, each data model inherits the following benefits from Cassandra:

- Continuous availability
- Easy geographical data distribution
- Operational low latency
- Linear scalability
- Operational maturity
- Simplified development (e.g. smart drivers that support all data models in each connector)

In addition, each data model benefits from the enterprise capabilities found only in DSE:

- Enterprise-grade security that protects sensitive data
- Functional cohesiveness that includes built-in analytics for analyzing operational data, integrated enterprise search that satisfies modern application search requirements, and an in-memory option for fast read operations
- Simplified visual management via OpsCenter and command line tools
- Expert 24x7x365 support
- Certified software updates, hot fixes, and formal end-of-life policies

For more information about DataStax Enterprise and downloads of DataStax software, visit http://www.datastax.com.

About DataStax

DataStax, the leading provider of database software for cloud applications, accelerates the ability of enterprises, government agencies, and systems integrators to power the exploding number of cloud applications that require data distribution across datacenters and clouds, by using our secure, operationally simple platform built on Apache Cassandra™. With more than 500 customers in over 50 countries, DataStax is the database technology of choice for the world's most innovative companies, such as Netflix, Safeway, ING, Adobe, Intuit, Target and eBay. Based in Santa Clara, Calif., DataStax is backed by industry-leading investors including Comcast Ventures, Crosslink Capital, Lightspeed Venture Partners, Kleiner Perkins Caufield & Byers, Meritech Capital, Premji Invest and Scale Venture Partners. For more information, visit DataStax.com or follow us on @DataStax. 05.27.16