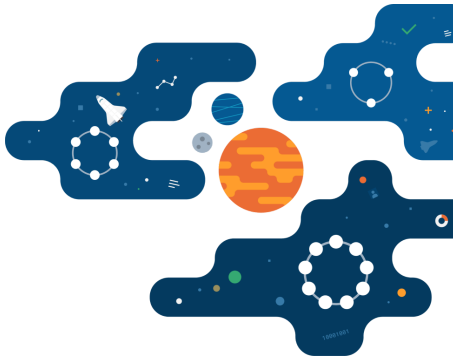
WHITEPAPER



Data Modeling in Apache Cassandra™

Five Steps to an Awesome Data Model

For global-scale applications, Apache Cassandra is the favorite choice among architects and developers.

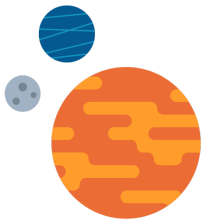
It offers many advantages including zero-downtime high availability, zero lock-in with easy data portability, and global scale with extreme performance. Today, Cassandra is among the most successful NoSQL databases. It is used in countless applications from online retail to internet portals to time-series databases to mobile application backends.

Having a well-designed data model is essential to meeting application performance and scalability goals. In this paper, aimed at technical people experienced with relational databases, we discuss five useful steps to realizing a high-quality data model for your Cassandra application.

DataStax

01

Data Modeling in Cassandra



Why data modeling is critical

Having the correct data model is important for any application, but it becomes especially critical for applications that need zero downtime, zero lock-in, and global scale. An application may work with thousands of records and a few hundred concurrent users, but what happens when record and user counts are in millions or billions?

Regardless of the database, if the data model isn't right, or doesn't suit the underlying database architecture, users can experience poor performance, downtime, and even data loss or data corruption. Fixing a poorly designed data model after an application is in production is an experience that nobody wants to go through. It's better to take some time upfront and use a proven methodology to design a data model that will be scalable, extensible, and maintainable over the application lifecycle.

Differences between Cassandra and relational databases

Most readers will be familiar with relational databases such as Oracle®, MySQL®, and PostgreSQL®. Before jumping into data modeling with Cassandra, it's helpful to explain a few differences between Cassandra and relational databases. High-level relational databases prioritize space over response time—one copy of data stored, but many joins that slow performance in order to pull a result set out of it. Cassandra prioritizes response time over space—a few copies of data and a real-time return of a result set.

- **Zero-downtime, high availability** – In a relational database, high availability has to be carefully configured and tested for each system, which delays the time it takes your application to get to production. The cloud-native architecture of Cassandra allows for your data to sit on premises, in any cloud, or multiple clouds with built-in data availability and out-of-the-box direction of traffic to available nodes.
- **Zero lock-in, highly portable data** – In a relational database, developers and DBAs rely on ETL to move data around which takes a significant amount of time when you have massive amounts of data. The time it takes to move data means it can be unavailable to your system, and often applications running on relational databases have scheduled maintenance windows to handle porting data. The cloud-native architecture of Cassandra powers highly portable data out of the box, allowing developers to move faster when building applications. With Cassandra, there's no need for scheduled maintenance windows, even during database upgrades.

- **Global scale, extreme performance worldwide** – In a relational database, to achieve distributed data, geo-local low latencies across multiple geographies is a challenge and when you layer on top of that the need to serve out extremely high throughput the relational databases fall over. Relational databases at their core were not designed for this level of customer interaction at a true global scale. In Cassandra, the performance of high writes at global scale is a major reason enterprises select Cassandra for delivering applications with lightning fast customer experiences globally.

- **In Cassandra, denormalization is expected** – With relational databases, designers are usually encouraged to store data in a normalized[1] form to save disk space and ensure data integrity. In Cassandra, storing the same data redundantly in multiple tables is not only expected, but it also tends to be a feature of a good data model.

- **In Cassandra, writes are (almost) free** – Owing to Cassandra's architecture, writes are shockingly fast compared to relational databases. Write latency may be in the hundreds of microseconds and large production clusters can support millions of writes-per-second.[2]

- **No joins** – Relational database users assume that they can reference fields from multiple tables in a single query, joining tables on the fly which can slow performance of your application. With Cassandra, joins don't exist, so developers structure their data model to provide equivalent functionality.

- **Developers need to think about consistency** – Relational databases are ACID compliant (Atomicity, Consistency, Isolation, Durability), characteristics that help guarantee data validity with multiple reads and writes. Cassandra supports the notion of tunable consistency, balancing between Consistency, Availability, and Partition Tolerance (CAP)[3]. Cassandra gives developers the flexibility to manage trade-offs between data consistency, availability, and application performance when formulating queries.

- **Indexing** – In a relational database, queries can be optimized by simply creating an index on a field. While secondary indexes exist in Cassandra, they are not a "silver bullet" as they are in a relational database management system (RDBMS). In Cassandra, tables are usually designed to support specific queries, and secondary indexes are useful only in specific circumstances.

---

[1] Database normalization is the process of structuring a relational database to reduce data redundancy and improve data integrity – details are available at https://en.wikipedia.org/wiki/Database_normalization

[2] Cassandra delivers one million writes per second at Netflix – https://medium.com/netflix-techblog/benchmarking-cassandra-scalability-on-aws-over-a-million-writes-per-second-39f45f066c9e

[3] How Apache Cassandra™ Balances Consistency, Availability, and Performance – https://www.datastax.com/2019/05/how-apache-cassandra-balances-consistency-availability-and-performance

How Cassandra stores data

Understanding how Cassandra stores data is essential to developing a good data model. Readers wishing to get a better understanding of Cassandra's internal architecture can read the DataStax Apache Cassandra™ Architecture whitepaper available at ***https://www.datastax.com/resources/whitepapers/apache-cassandra-architecture.***

Cassandra clusters have multiple nodes running in local data centers or public clouds. Data is typically stored redundantly across nodes according to a configurable replication factor so that the database continues to operate even when nodes are down or unreachable.

Tables in Cassandra are much like RDBMS tables. Physical records in the table are spread across the cluster at a location determined by a partition key. The partition key is hashed to a 64-bit token that identifies the Cassandra node where data and replicas are stored.[4] The Cassandra cluster is conceptually represented as a ring, as shown in Figure 1, where each cluster node is responsible for storing tokens in a range.

Queries that look up records based on the partition key are extremely fast because Cassandra can immediately determine the host holding required data using the partitioning function. Since clusters can potentially have hundreds or even thousands of nodes, Cassandra can handle many simultaneous queries because queries and data are distributed across cluster nodes.

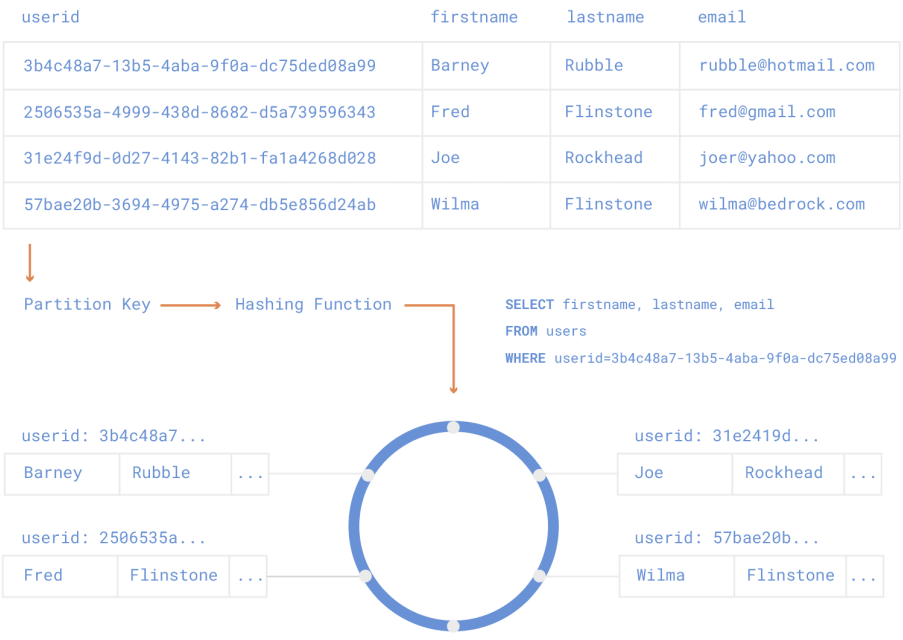


FIGURE 1 How Cassandra stores data

[4] This is a bit of an over-simplification. Recent versions of Cassandra support the notion of vnodes where each cluster node is responsible for multiple token ranges to better distribute data. https://docs.datastax.com/en/dse/5.1/dse-arch/datastax_enterprise/dbArch/archDataDistributeVnodesUsing.html. The ring is still a useful way to conceptualize how Cassandra stores data however