Wide-Column Databases and Beyond

Chapter 16

Michael Tsiang

Stats 167: Introduction to Databases

UCLA

Do not post, share, or distribute anywhere or with anyone without explicit permission.

Wide-Column Databases

Scaling Up

Document databases are optimal for flexible storage of hierarchical or nested semi-structured data. However, because documents can be deeply nested and vary in structure, high-volume writes to the database may require more processing time than writes to flatter (unnested), more structured formats.

- But what if we want to store data that is not deeply nested?
- Is there a more efficient and scalable data model for extremely high-volume writes?

Wide-Column Databases

A **wide-column database** (also called a **column-family database**) stores data in rows, each of which can have a flexible set of columns.

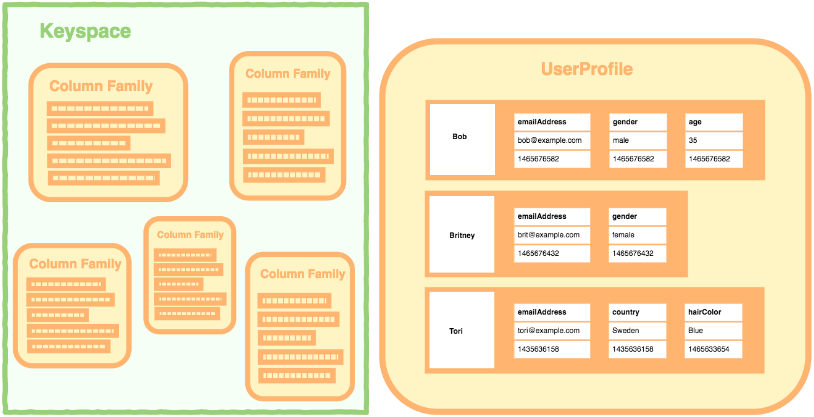
Related rows are grouped into **column families** (also referred to as **tables** in modern Cassandra terminology), and multiple column families are organized under a **keyspace**.

A column family in a wide-column database is analogous to a table in a relational database, and a keyspace is analogous to a schema.

However, unlike relational tables, each row in a column family can have a different set of columns.

Example of Wide-Column Database

An illustration of a keyspace and a column family:



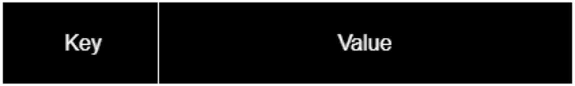
Source: Database.Guide article, “What is a Column Store Database?”

Two-Dimensional Key-Value Database

In addition to being viewed as a relational database model with more flexible columns, a wide-column database can also be viewed as a two-dimensional key-value database.

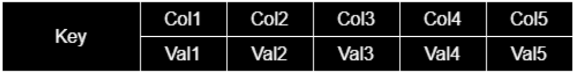
If we think of a key-value pair as a row in a table, the key is a row identifier and the data is the value in the row:

$$\text{key} = \text{rowID},\ \text{value} = \text{data}$$



In a wide-column database, the key has two components, where the first component is a row identifier and the second is a column name (i.e., the column key within the row):

$$\text{rowID} \rightarrow \{\text{columnID: value}\}$$



Source: Dan D. Kim article, “Wide-Column Databases”

Characteristics of Wide-Column Databases

Wide-column databases have a column-oriented design within row groups[1]: Data is organized by rows, but within each row, columns are stored and accessed separately. Due to this design, wide-column databases have several key characteristics:

- Flexible structure: Each row can have a different set of columns.
- Highly scalable: Data is partitioned by row, enabling efficient horizontal scaling (e.g., sharding) across distributed systems.
- High write throughput: Wide-column databases are optimized for OLTP workloads with high-volume writes.
- Flexible schema evolution: New columns can be introduced per row without modifying a predefined schema.
- Tunable consistency and partition tolerance: Many wide-column databases allow configurable trade-offs between consistency, availability, and partition tolerance in distributed systems.

[1] This is (confusingly) different from a columnar database.

Wide-Column Database Use Cases

Wide-column databases are useful in situations involving massive amounts of writes of semi-structured data that need to scale across many machines. Common use cases include:

- IoT (Internet of Things) sensor data
- Time series data (e.g., temperature monitoring or financial trading)
- Event logs
- GPS tracking
- Messaging systems
- Real-time engagement metrics
- Recommendation engines

When Not to Use

Wide-column databases are designed for high-volume writes and horizontal scaling across distributed systems. However, they are not suited to certain scenarios, such as:

- Applications with a **fixed schema** and structured (tabular) data
- Workloads requiring full **ACID compliance** (e.g., banking transactions)
- Use cases involving **complex joins or analytical queries**
- Scenarios with **frequent updates or deletes**
- **Read-heavy workloads**, where document or key-value databases may have better performance
- **Short-lived data**, where durability and distributed replication add unnecessary overhead

Apache Cassandra

Apache Cassandra and CQL

Apache Cassandra (https://cassandra.apache.org/) is a popular, free, and open-source, distributed, wide-column database management system.

Cassandra Query Language (CQL) is the primary language used to interact with Apache Cassandra.

CQL is a declarative language that uses SQL-like query syntax but is designed for Cassandra's distributed, wide-column architecture.

Cassandra Keys

To understand how to query with CQL, we need to understand Cassandra's distributed design and the role of keys to access data.

The **primary key** in Cassandra is composed of the partition key and zero or more clustering keys. Every table in Cassandra must have a primary key.

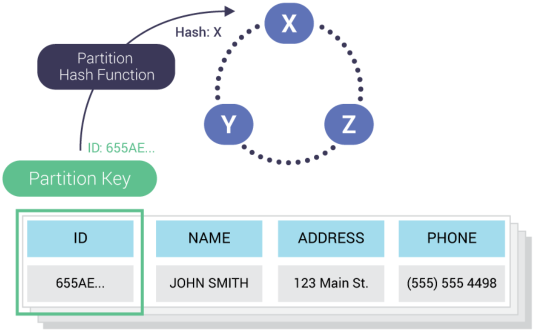
From the ScyllaDB article "Cassandra Clustering Key":

- The **partition key** is part of the primary key and defines which node will store the data based on its hash value and distributes data across the nodes in the Cassandra cluster. All rows with the same partition key will be stored together on the same node.
- The **clustering key(s)** (or **clustering columns**) follow the partition key in the definition of the primary key in Cassandra. The clustering key sorts the data within a partition, allowing rows with the same partition key to be ordered based on their different clustering key values.

Cassandra filtering is done only through the partition key and further through the clustering keys.

Example of a Partition Key and Hash Function

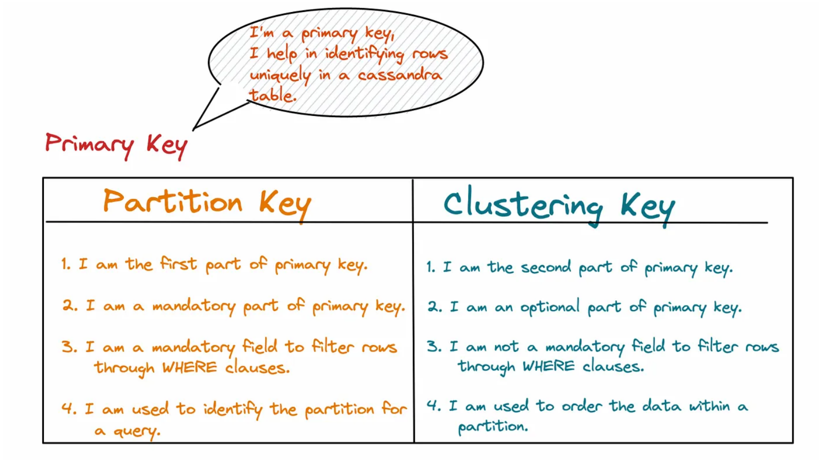
An illustration of the Cassandra ring architecture and the partition key and partition hash function:



Source: ScyllaDB Documentation, “ScylladDB Ring Architecture - Overview”

Differences Between Keys in Cassandra

A table illustration that explains the differences between primary keys, partition keys, and clustering keys:


I'm a primary key, I help in identifying rows uniquely in a cassandra table.
Primary Key

Partition Key	Clustering Key
1. I am the first part of primary key.	1. I am the second part of primary key.
2. I am a mandatory part of primary key.	2. I am an optional part of primary key.
3. I am a mandatory field to filter rows through WHERE clauses.	3. I am not a mandatory field to filter rows through WHERE clauses.
4. I am used to identify the partition for a query.	4. I am used to order the data within a partition.

Source: Medium article by Madhura Mehendalek, "Partition Key in Cassandra", 2023

Data Definition Language in CQL

The basic Data Definition Language (DDL) commands in CQL:

- `CREATE KEYSPACE`
- `ALTER KEYSPACE`
- `DROP KEYSPACE`
- `CREATE TABLE`
- `ALTER TABLE`
- `DROP TABLE`
- `CREATE INDEX`
- `DROP INDEX`

Examples and syntax are available in the Cassandra CQL documentation on DDL.

Data Manipulation Language in CQL

The basic Data Manipulation Language (DML) commands in CQL:

- `INSERT`
- `UPDATE`
- `DELETE`
- `SELECT`

Examples are shown in the Cassandra CQL documentation on DML.

Limitations of CQL Compared to SQL

Most of the standard clauses of a `SELECT` statement in SQL still exist in CQL, e.g., `FROM`, `WHERE`, `GROUP BY`, `ORDER BY`, `LIMIT`.

However, since Cassandra is a wide-column, non-relational database that prioritizes scalability and partitioned data access, `SELECT` queries in CQL are much more constrained than in SQL.

- CQL does not support joins or subqueries.
- A `SELECT` statement can only query a single table.
- The `WHERE` clause (mostly) supports filtering only on the primary key (i.e., the partition key and clustering columns).[2]
- The `GROUP BY` clause exists but is limited:
  - It can be used only within a single partition (i.e., the partition key must be specified in `WHERE`).
  - It applies only to clustering columns.
- There is no `HAVING` clause.

[2]Filtering on non-key columns requires the `ALLOW FILTERING` keyword, which is discouraged due to performance costs.

DDL Example

```
CREATE TABLE employees (
  department TEXT,
  hire_date DATE,
  name TEXT,
  title TEXT,
  PRIMARY KEY ((department), hire_date, name)
);
```

Note the primary key:

- The partition key is `department` (notice the double parentheses).
- Clustering keys are `hire_date` and `name`.

The data is grouped by department and then sorted by hire date and then name.

Sample Data in Partitions

Partition: `department = 'Parks and Recreation'`

`hire_date`	`name`	`title`
2009-04-09	Leslie Knope	Deputy Director
2007-04-22	Ron Swanson	Director
2009-09-24	Tom Haverford	Administrator
2009-04-16	Donna Meagle	Office Manager

Partition: `department = 'Library'`

`hire_date`	`name`	`title`
2009-11-05	Tammy Swanson Swanson	Deputy Director
2010-02-04	Marci	Librarian

Example CQL Query

Valid CQL query:

```
SELECT hire_date, COUNT(*)
FROM employees
WHERE department = 'Parks and Recreation'
GROUP BY hire_date;
```

Invalid CQL query:

```
SELECT department, COUNT(*)
FROM employees
GROUP BY department;
```

Question: Why is this query invalid?

Comparison Between Cassandra and MongoDB

For a comparison between the syntax of Cassandra and MongoDB queries:

https://phoenixnap.com/kb/cassandra-vs-mongodb

The query languages CQL and MQL are also compared side-by-side.

For example, for a basic query (if `name` was the partition key):

In Cassandra:

```
SELECT *
FROM employees
WHERE department = 'Parks and Recreation';
```

In MongoDB:

```
db.employees.find({ department: 'Parks and Recreation' })
```

Astra DB

Astra DB (by Datastax) is a **Database-as-a-service (DBaaS)** built on Apache Cassandra. Astra DB offers a free $25 credit *per month* (up to 80GB storage and 20 million read/write operations).

Astra DB's website:
https://www.datastax.com/products/datastax-astra

Getting Started with Astra DB

To get started with Astra DB:

- Some instructions on how to get started with Astra DB: https://ostechnix.com/datastax-astra-db-tutorial/
- Datastax hosts free workshops on NoSQL and Cassandra: https://www.eventbrite.com/o/datastax-23222864038
- Slides for the Datastax NoSQL workshop: https://github.com/datastaxdevs/workshop-introduction-to-nosql
- Datastax also offers free interactive tutorials on Cassandra and the basics of Cassandra Query Language (CQL): https://www.datastax.com/learn/cassandra-fundamentals

ScyllaDB

ScyllaDB offers **ScyllabDB Cloud**, a wide-column DBaaS that is compatible with Apache Cassandra and CQL. It is marketed as a "drop-in replacement" to Cassandra with higher performance and scalability.

ScyllaDB Cloud has a free tier: https://cloud.scylladb.com/

- ScyllaDB Quick Start Guide and Documentation: https://cloud.docs.scylladb.com/stable/
- Learning resources: https://resources.scylladb.com/
- NoSQL courses through ScyllabDB University: https://university.scylladb.com/

Where Do We Go From Here?

Graph Databases

A **graph database** is a database that uses a graph structure for storing data.

Rather than tables or documents, data is stored through a network of nodes, edges, and properties, where edges represent relationships between nodes.

Graph databases are particularly useful for large scale scenarios where relationships in the data are important to explicitly model.

Some use cases of graph databases:
https://aws.amazon.com/nosql/graph/#topic-1

A basic tutorial on graph databases:
https://www.datacamp.com/blog/what-is-a-graph-database

Neo4j

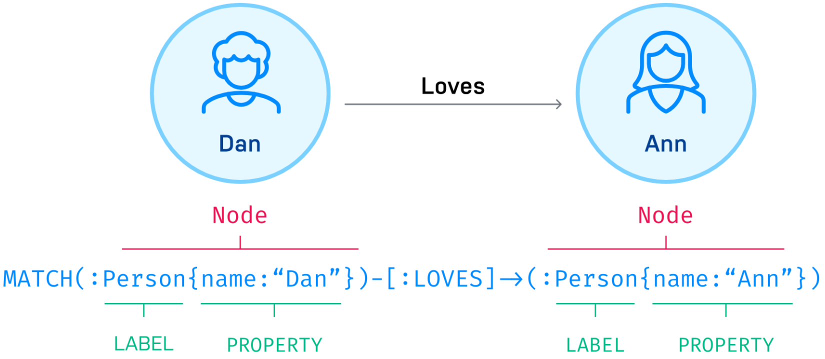
Neo4j (https://neo4j.com/) is a graph database management system.

Neo4j offers **AuraDB**, a fully managed cloud service (DBaaS) for graph databases with a free tier: https://neo4j.com/product/auradb/

Example Relationship in Cypher

Neo4j's graph query language is called **Cypher**, which is conformant with GQL (graph query language), an up and coming international standard query language for graph databases.

An example graphic showing pattern matching and relationships using Cypher.



Source: Neo4j Documentation, "What is a graph database?"

Getting Started with Neo4j and AuraDB

To get started on Neo4j:

- Introduction to the Neo4j Graph Database: https://www.bmc.com/blogs/neo4j-graph-database/
- Getting Started with Aura Free Tier video: https://youtu.be/1Ee242FDFcc
- A Neo4j tutorial by Chris Hay: https://youtu.be/IShRYPsmiR8

Vector Databases

With the rapid ubiquity of AI, a new type of NoSQL database has risen in popularity in the last few years called a **vector database**, which stores data in vectors (similar to vectors in R).

A quick overview by Fireship:
https://www.youtube.com/watch?v=klTvEwg3oJ4

A couple popular vector database management systems are **Pinecone** (https://www.pinecone.io) and **Weaviate** (https://weaviate.io/).

Thank You

Thank you for being the inaugural class of Stats 167!