# Cassandra - Column-Oriented or Column-Family Database?

## 1. Introduction

Apache Cassandra is an open-source distributed NoSQL database that is built to handle large amounts of data across multiple data centers. Cassandra's data model is a topic of discussion across multiple documents and papers, often resulting in confusing or contradictory information. This is due to Cassandra's ability to store and access column families separately, which results in a mistaken classification as *column-oriented* rather than *column-family.*

In this article, we'll look at the differences between data models and establish the nature of Cassandra's *partitioned row store* data model.

## 2. Database Data Models

The *README* on the Apache Cassandra git repo states that:

Cassandra is a partitioned row store. Rows are organized into tables with a required primary key.

Partitioning means that Cassandra can distribute your data across multiple machines in an application-transparent matter. Cassandra will automatically repartition as machines are added and removed from the cluster.

Row store means that like relational databases, Cassandra organizes data by rows and columns.

From this, we can conclude that **Cassandra is a *partitioned rowstore***. However, *column-family* or *wide-column* are also suitable names, as we'll find out below.

**A *column-family* data model is not the same as a *column-oriented* model**. A *column-family* database stores a row with all its column families together, whereas a *column-oriented* database simply stores data tables by column rather than by row.

### 2.1. Row-Oriented and Column-Oriented Data Stores

Let's take an *Employees* table as an example:

| ID | Last | First | Age |
|---|---|---|---|
| 1 | Cooper | James | 32 |
| 2 | Bell | Lisa | 57 |
| 3 | Young | Joseph | 45 |

A *row-oriented* database stores the above data as:

1,Cooper,James,32;2,Bell,Lisa,57;3,Young,Joseph,45;

While a *column-oriented* database stores the data as:

1,2,3;Cooper,Bell,Young;James,Lisa,Joseph;32,57,45;

**Cassandra does not store its data like either a *row-oriented* or a *column-oriented* database.**

### 2.2. Partitioned Row Store

**Cassandra uses a *partitioned row store***, which means rows contain columns. A *column-family* database stores data with keys mapped to values and the values grouped into multiple column families.

In a *partitioned row store*, the *Employees* data looks like this:

```
"Employees" : {
        row1 : { "ID":1, "Last":"Cooper", "First":"James", "Age":32},
        row2 : { "ID":2, "Last":"Bell", "First":"Lisa", "Age":57},
        row3 : { "ID":3, "Last":"Young", "First":"Jospeh", "Age":45},
        ...
    }
```

**A *partitioned row store* has rows that contain columns, yet the number of columns in each row does not have to be the same** (like *big-table*). Some rows may have thousands of columns, while some rows could be limited to just one.

We can think of a *partitioned row store* as a *two-dimensional key-value store*, where a row key and a column key are used to access data. **To access the smallest unit of data (a column), we must first specify the row name (key) and then the column name.**

## 3. Conclusion

In this article, we have learned that Cassandra's *partitioned row store* means that it is ***column-family* rather than *column-oriented***. The main characteristic that defines *column-family* is that **column information is part of the data**. This is the main difference between a *column-family* model and both *row-oriented* and *column-oriented* models. The term *wide-column* comes from the idea that tables holding an unlimited number of columns are wide by nature.

We've also explored how rows in a *column-family* datastore don't need to share column names or column numbers. This enables *schema-free* or *semi-structured* tables.