# Apache Cassandra NoSQL Database

by: Panisara Yimcharoen 66011564 & Phatdanai Khemanukul 66011147

## Introduction

Apache Cassandra is a NoSQL database that is designed to handle large amounts of data across multiple servers without a single point of failure. It is highly scalable and works well for applications that require fast data processing and high availability. In this report, information about Cassandra's data structure and key features. It also includes the dataset Presidents from the presidential database to demonstrate how data is stored and queried in Cassandra compared to a relational database.

## Cassandra Database Model

Cassandra follows a column-family-based data model, which differs significantly from relational database models. Here are the key aspects of its structure:

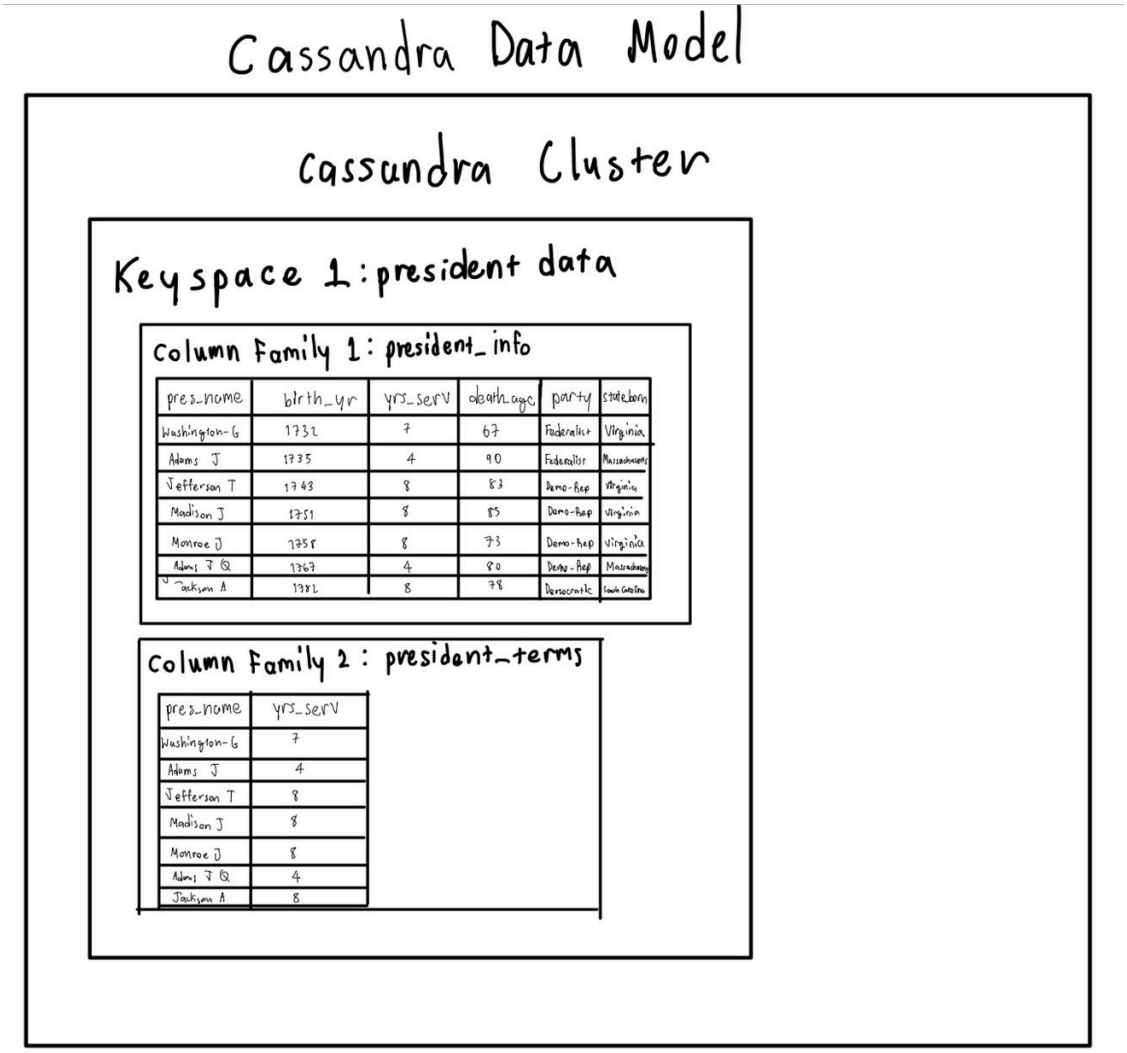
### 1. Logical Data Structure

Cassandra organizes data into keyspaces, which contain column families. Each column family consists of multiple rows, where each row contains a unique primary key and a set of columns.

Let's consider the **President Table** dataset from an RDBMS and compare how it would be represented in Cassandra.

| PRES_NAME | BIRTH_YR | YRS_SERV | DEATH_AGE | PARTY | STATE_BORN |
|---|---|---|---|---|---|
| Washington G | 1732 | 7 | 67 | Federalist | Virginia |
| Adams J | 1735 | 4 | 90 | Federalist | Massachusetts |
| Jefferson T | 1743 | 8 | 83 | Demo-Rep | Virginia |
| Madison J | 1751 | 8 | 85 | Demo-Rep | Virginia |
| Monroe J | 1758 | 8 | 73 | Demo-Rep | Virginia |
| Adams J Q | 1767 | 4 | 80 | Demo-Rep | Massachusetts |
| Jackson A | 1767 | 8 | 78 | Democratic | South Carolina |
| Van Buren M | 1782 | 4 | 79 | Democratic | New York |
| Harrison W H | 1773 | 0 | 68 | Whig | Virginia |
| Tyler J | 1790 | 3 | 71 | Whig | Virginia |
| Polk J K | 1795 | 4 | 53 | Democratic | North Carolina |
| Taylor Z | 1784 | 1 | 65 | Whig | Virginia |
| Fillmore M | 1800 | 2 | 74 | Whig | New York |

President Table
In a relational database, each row represents a president, and all records must follow a defined structure with fixed columns according to a predefined schema.




## Cassandra Cluster

This cluster contains all data related to U.S. Presidents.

**Keyspace: presidents_data**

This keyspace organizes all information about presidents.

Column Family 1: presidents_info

- pres_name (Partition Key)
- birth_yr (Clustering Column)
- yrs_serv

- death_age
- party
- state_born

Column Family 2: president_terms

- pres_name (Partition Key)
- yrs_serv (Clustering Column)

This model optimizes queries based on president names and years served, improving lookup speed for historical queries.

Partition Key (`pres_name`) ensures fast lookups.

- Instead of scanning rows in a SQL table, Cassandra distributes data across multiple nodes based on `pres_name`.

Clustering Column (`birth_yr`) organizes data efficiently.

- Sorting by `birth_yr` within each partition makes range queries faster.

Denormalized structure avoids expensive `JOINs`.

- `presidents_info` and `presidents_terms` are separate column families instead of requiring complex joins.

## 2. Integrity Constraint

In traditional relational database management systems (RDBMS), integrity constraints such as entity integrity (primary keys must not be null) and referential integrity (foreign keys must match primary keys or be null) are strictly enforced to maintain data accuracy and consistency. Apache Cassandra, as a NoSQL distributed database, prioritizes availability, scalability, and performance, and thus handles integrity constraints differently.

### i) Primary Key Constraints

Cassandra enforces a primary key constraint to ensure that each row in a table is uniquely identified. The primary key in Cassandra comprises:

- **Partition Key**: Determines data distribution across nodes in the cluster.
- **Clustering Keys**: Define the sorting order of rows within a partition.

The partition key is crucial for data distribution and retrieval, while clustering keys organize data within the partition.

**Example:**

```
CREATE TABLE presidents (
    pres_name TEXT,
    birth_yr INT,
    yrs_serv INT,
    PRIMARY KEY (pres_name, birth_yr)
);
```

In this example:

- pres_name: Serves as the partition key, ensuring that all rows with the same pres_name are stored in the same partition.
- birth_yr: Acts as the clustering key, ensuring uniqueness within the partition and defining the sort order of rows.

**ii) Referential Integrity Constraints**

Unlike relational databases, Cassandra does not enforce referential integrity constraints such as foreign keys. This means that relationships between tables are not automatically maintained by the database. To handle this, developers often use a denormalization approach, storing related data together within the same table to maintain data consistency.

**Data Modeling:**
To keep data organized, developers often store related data in the same table instead of splitting it into multiple tables. This reduces the need for complex queries and makes data retrieval faster.

**Application Logic:**
Since Cassandra does not automatically manage relationships between tables, applications must include rules to keep data connections accurate and consistent.

## 3. Database Language

Cassandra uses Cassandra **Query Language (CQL),** a query language that resembles SQL but is specifically designed for NoSQL environments and distributed databases.

**Example** Query to Retrieve Presidents Who Were Born After 1800:

```
SELECT * FROM presidents WHERE birth_yr > 1800;
```

**Example** Query to Update a President's Party Affiliation:

```
UPDATE presidents
SET party = 'Independent'
WHERE pres_name = 'Washington G';
```

## 4. Features of Cassandra

**Distributed System:**
All nodes in Cassandra have the same role, meaning there is no single point of failure. Data is spread across multiple nodes, but since there is no master node, handling requests can sometimes be challenging.

**Data Replication & Multi-Data Center Support:**
Cassandra automatically makes copies of data to improve reliability. It is designed to work across many data centers, ensuring data is always available, even if some servers fail.

**Scalability:**
Cassandra can easily handle more data and users. When new machines are added, the system grows without stopping or slowing down other applications.

**Fault Tolerance:**
Cassandra keeps multiple copies of data, so if one machine fails, another takes over immediately, preventing data loss.

**Big Data Processing:**
Cassandra works well with big data tools like Hadoop, MapReduce, Apache Hive, and Apache Pig, making it easier to process large amounts of information.

**Query Language:**
Cassandra uses Cassandra Query Language (CQL), which is simple to use and helps developers access data quickly and easily.

## 5. Applications suitable for NoSQL

### 1. Big Data Applications

**Why use NoSQL?**
NoSQL databases are great at handling huge amounts of data. Unlike regular databases that need fixed structures, NoSQL can store all kinds of data—structured, semi-structured, and unstructured—without strict rules.

**Benefits of NoSQL:**

- Can store and manage data across many servers (scales easily).
- Works well with cloud storage and distributed systems.
- Supports different data formats like JSON, key-value pairs, and documents.
- Can quickly handle large amounts of incoming data (useful for logs, sensors, and analytics).

### 2. Real-Time Analytics

Why use NoSQL?
NoSQL databases can process and analyze data instantly without slowing down. Regular databases struggle with this because of their complex queries and rigid structure.

**Benefits of NoSQL:**

- Fast read and write speeds, even with large datasets.
- Can handle data that keeps changing in structure.
- Can add more servers to improve performance (scalable).
- Perfect for dashboards, fraud detection, and recommendation systems.

### 3. Content Management Systems (CMS)

**Why use NoSQL?**
A CMS stores different types of content like text, images, videos, and comments. NoSQL is flexible and doesn't require a fixed structure, making it easier to manage mixed content types.

**Benefits of NoSQL:**

- Can store different kinds of content without needing a fixed structure.
- Allows quick searches and indexing for faster results.
- Can easily adjust to new content types without major changes.
- Can grow easily as the amount of content increases.

**4. IoT (Internet of Things) Applications**

**Why use NoSQL?**
Smart devices (IoT) generate a massive amount of data in real-time, often in different formats. NoSQL databases can process and store this data quickly and efficiently.

**Benefits of NoSQL:**

- Handles high-speed data input without delays.
- Can support millions of IoT devices at the same time.
- Stores time-based data efficiently for analysis.
- Works well in cloud-based and edge computing environments.

**5. Social Networks**

**Why use NoSQL?**
Social media apps need to store complex relationships between users, posts, likes, and comments. NoSQL databases, especially graph databases, are great at handling these connections.

**Benefits of NoSQL:**

- Graph databases can efficiently store and retrieve user relationships.
- Can quickly show posts, likes, and followers without delays.
- Can scale easily as more users join and interact.
- Adapts well when new features are added to the platform.

## Summary of Why NoSQL is Better for These Applications compare to using relational DBMS

**Scalable** – Can handle more data by adding more servers.
**Flexible** – Works with different types of data without needing strict rules.
**Fast** – Reads and writes data quickly, making it ideal for real-time apps.
**Distributed Storage** – Works well in cloud environments and multi-server setups.
**High Availability** – Ensures data is always accessible and backed up.