Data Modeling using Apache Cassandra and PostgreSQL

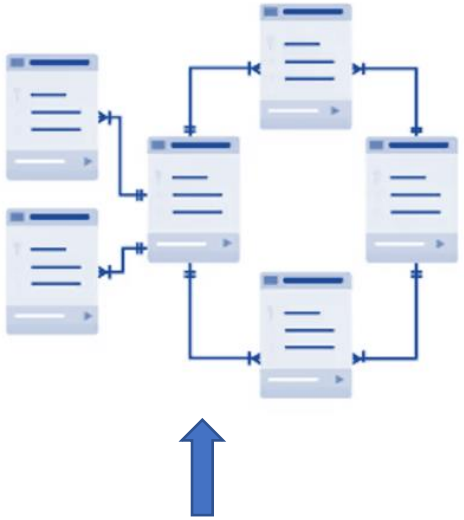
What is a Data Model?

"The **process of creating data models** for an information system."

Data modeling can easily translate to database modeling, as this is the essential **end state**

Note: PostgreSQL is also referred as Postgres in short

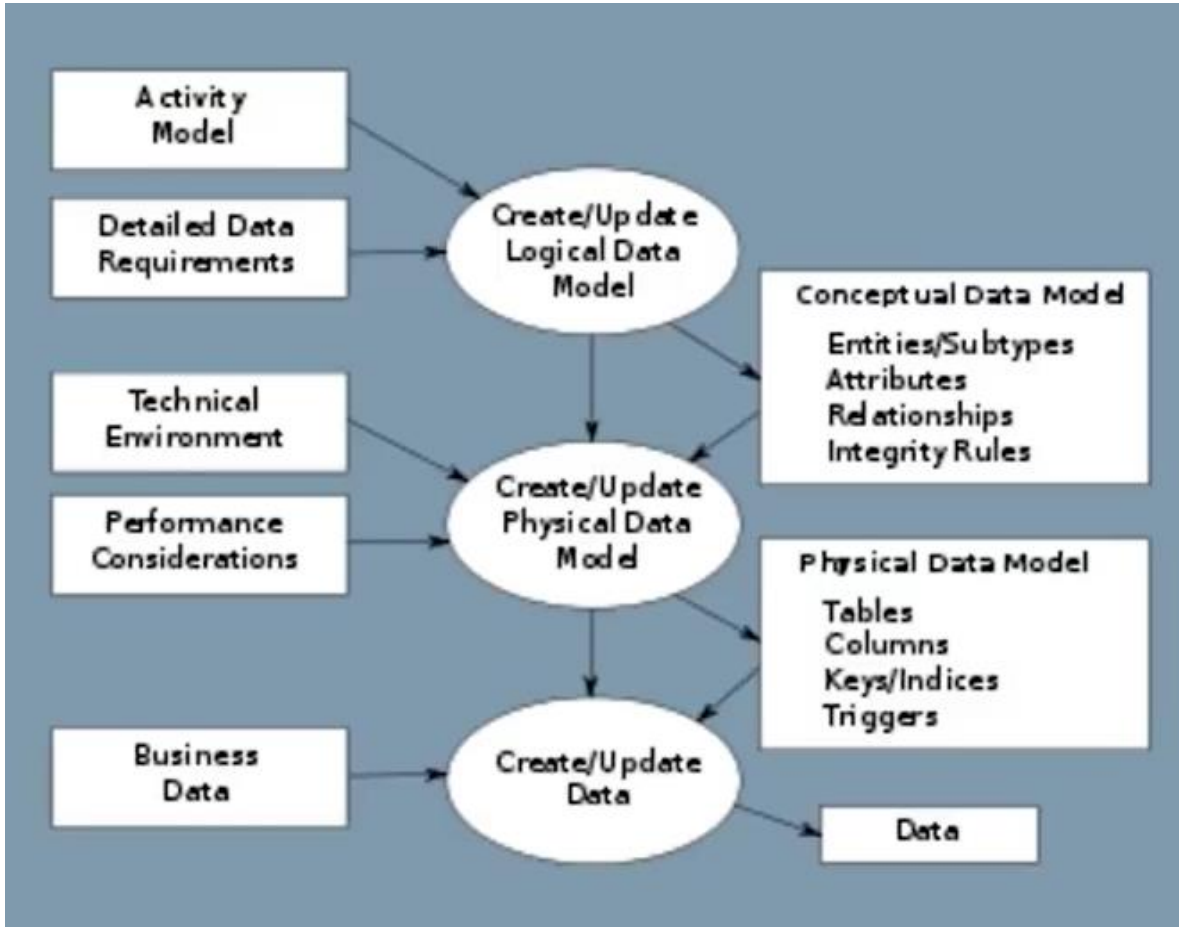
Data Modeling



- Process to support business and user applications
- Gather Requirements
- Conceptual Data Modeling
- Logical Data Modeling
- Physical Data Modeling

This is an Entity Relationship Diagram(ERD)

Note: DDL is Data Definition Language. These are commands that can be used to define the database schema eg CREATE, DROP, etc



Common Questions

Why can't everything be stored in a giant Excel spreadsheet?

- There are limitations to the amount of data that can be stored in an Excel sheet. So, a database helps organize the elements into tables - rows and columns, etc. Also reading and writing operations on a large scale is not possible with an Excel sheet, so it's better to use a database to handle most business functions.

Does data modeling happen before you create a database, or is it an iterative process?

- It's definitely an iterative process. Data engineers continually reorganize, restructure, and optimize data models to fit the needs of the organization.

How is data modeling different from machine learning modeling?

- Machine learning includes a lot of data wrangling to create the inputs for machine learning models, but data modeling is more about how to structure data to be used by different people within an organization. You can think of data modeling as the process of designing data and making it available to machine learning engineers, data scientists, business analytics, etc., so they can make use of it easily.

Choose the Correct Order of the Data Modeling Process:

- ○ Physical -> Logical -> Conceptual
- ○ Logical -> Physical -> Conceptual
- ○ Conceptual -> Physical -> Logical
- ○ Conceptual -> Logical -> Physical

▼ **SOLUTION:**

Conceptual -> Logical -> Physical

Key points about Data Modeling

- Data organization is critical
- Organized data determines later data use
- Begin prior to building out application, business logic, and analytical models
- Iterative process

Data Modeling is an important skill for:

- data scientist,
- software engineers,
- data engineers,
- anyone involved in the process of using and analyzing data

So really EVERYONE!

Who should focus on learning data modeling? Choose one response.

- ○ Data Scientists
- ○ Data Engineers
- ○ Software Engineers
- ○ Everyone who deals with data!

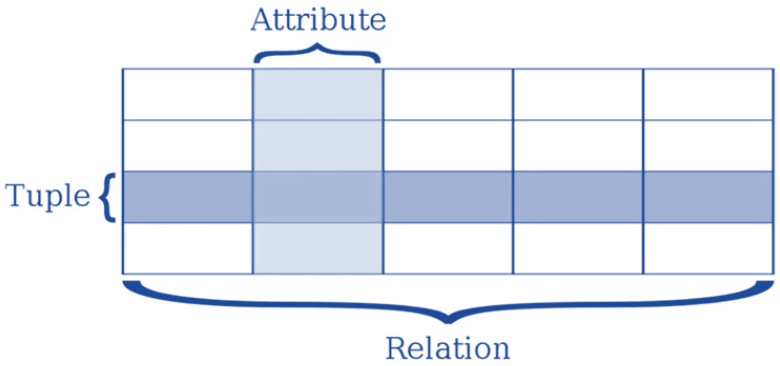
▼ **SOLUTION:**

Everyone who deals with data!

Relational Databases

Relational Model

"This model **organizes data into** one or more tables (or "**relations**") **of columns and rows**, with a **unique key identifying each row**. Generally, each table represents one "entity type" (such as customer or product)."



Relational Database

Invented by Edgar Codd (1970)

"... is a digital database **based on the relational model** of data...a software system used to maintain relational databases is a relational database management system (RDBMS)."

"SQL (Structured Query Language) is the language used across almost all relational database system for querying and maintaining the database."

Common Types of Relational Databases

- Oracle
- Teradata
- MySql
- PostgreSQL
- Sqlite

The Basics

- Database/Schema
  - Collection of Tables
- Tables/Relation
  - A group of rows sharing the same labeled elements
    - Customers

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

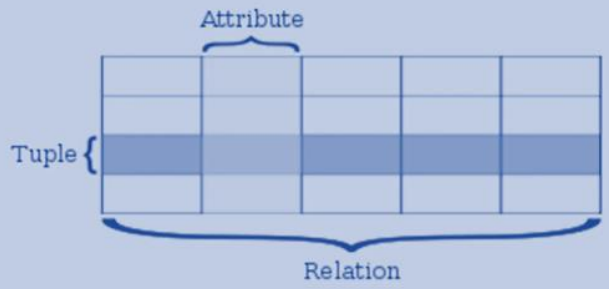
Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Customers

Name	Email	City
Amanda	jdoe@xyz.com	NYC
Toby	n/a	NYC

- Columns/Attribute
  - Labeled element
    - Name, email, city
- Rows / Tuple
  - A single item
    - Amanda, jdoe@xyc.com, NYC



Customers

Name	email	City
Amanda	jdoe@xyz.com	NYC
Toby	n/a	NYC

What is an RDBMS?

- ○ Relateable Database Management System
- ○ Relational Database Management System
- ○ Readable Data Management System
- ○ Reachable Database Management System

▼ **SOLUTION:**
Relational Database Management System

True or False: An attribute is another name for a column.

- ○ True
- ○ False

▼ **SOLUTION:**
True

True or False: A schema is a collection of tables in some database terminology.

- ○ True
- ○ False

▼ **SOLUTION:**
True

Advantages of using a Relational Database

- Ease of use -- SQL
- Ability to do JOINS
- Ability to do aggregations and analytics
- Smaller data volumes
- Easier to change business requirements
- Flexibility for queries
- Modeling the data not modeling queries
- Secondary Indexes available
- ACID Transactions --data integrity

Secondary Index

Here Country code is the secondary index

City

City	City code	Country code
Paris	048	001
London	097	002
Liverpool	026	002

ACID Properties

"...properties of database transactions intended to **guarantee validity** even **in** the **event of errors, power failures**..."

Atomicity

"...the whole transaction is processed or nothing is processed"

Consistency

"...only transactions that abide by constraints and rules is written into the database otherwise database keeps previous state"

Isolation

"...transactions are processed independently and securely, order does not matter"

Durability

"...completed transactions are saved to database even of cases of system failure"

What are ACID properties?

Atomicity
Consistency
Isolation
Durability

Which of these are benefits of a relational database?

- [ ] ACID Transactions
- [ ] Ability to do JOINS
- [ ] Can handle big data
- [ ] Easy to change business requirements on the data

▼ **SOLUTION:**

- ACID Transactions
- Ability to do JOINS
- Easy to change business requirements on the data

Can you JOIN a table with another table on any column?

- ○ Yes, as long there are matching values in the columns
- ○ No, only on columns with the same name
- ○ Yes, you can join any columns together
- ○ No, only on columns in the same table

▼ **SOLUTION:**

Yes, as long there are matching values in the columns

When to not use a Relational Database

- Large amounts of data
- Need to be able to store different data type formats
- Need high throughput -- fast reads
- Need a flexible schema
- Need high availability
- Need horizontal scalability

When should you use a Relational Database?

- [ ] Small amounts of data
- [ ] Need to be able to do aggregations
- [ ] Need ACID transactions
- [ ] You need to be able to scale out quickly
- [ ] Need to be able to join multiple tables

▼ **SOLUTION:**

- Small amounts of data
- Need to be able to do aggregations
- Need ACID transactions
- Need to be able to join multiple tables

High Availability

... describes a database where there is very little **downtime** of the system, it is **always on** and **functioning"**

Horizontal Scalability

- Ability to add more servers to the system

Horizontal Scalability

True or False: Relational Databases are traditionally horizontally scalable.

- ○ True
- ○ False

▼ **SOLUTION:**

False

NoSQL Database

"...has a simpler design, simpler horizontal scaling, and finer control of availability. Data structures used are different than those in Relational Database are make some operations faster."

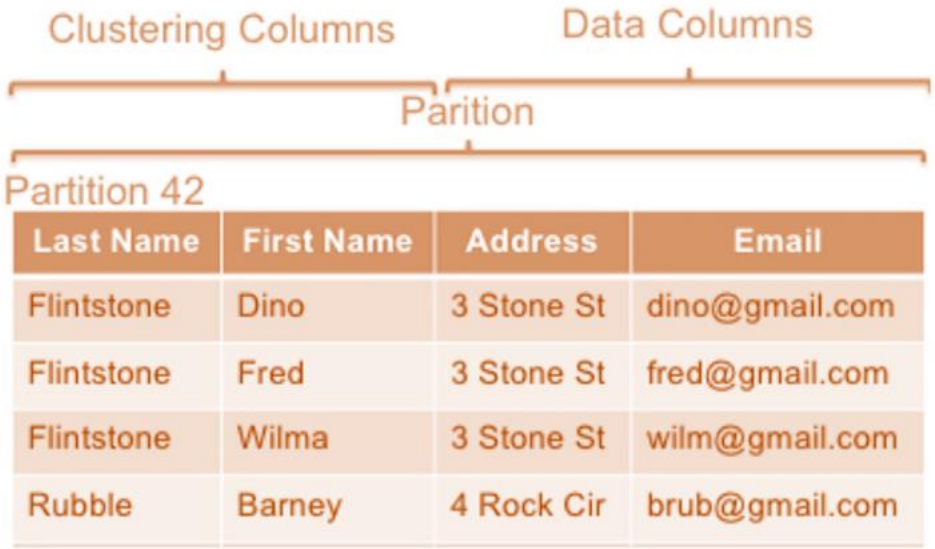
- NoSQL= Not Only SQL; NoSQL and NonRelational are interchangeable terms.
- Various types of NoSQL databases

Common Types of NoSQL Databases

- Apache Cassandra (Partition Row store)
- MongoDB (Document store)
- DynamoDB (Key-Value store)
- Apache HBase (Wide Column Store)
- Neo4J (Graph Database)

The Basics of Apache Cassandra

- Keyspace
  - Collection of Tables
- Table
  - A group of partitions
- Rows
  - A single item
- Partition
  - Fundamental unit of access
  - Collection of row(s)
  - How data is distributed
- Primary Key
  - Primary key is made up of a partition key and clustering columns
- Columns
  - Clustering and Data
  - Labeled element



Last Name	First Name	Address	Email
Flintstone	Dino	3 Stone St	dino@gmail.com
Flintstone	Fred	3 Stone St	fred@gmail.com
Flintstone	Wilma	3 Stone St	wilm@gmail.com
Rubble	Barney	4 Rock Cir	brub@gmail.com

True or False: A Keyspace in Apache Cassandra is similar to a schema in PostgreSQL

○ True

○ False

▼ **SOLUTION:**

True

Which of these are examples of non-relational databases?

☐ SQL

☐ Apache Cassandra

☐ RDBMS

☐ MongoDB

▼ **SOLUTION:**

- Apache Cassandra
- MongoDB

What is Apache Cassandra?

"...**provides scalability** and **high availability** without compromising performance. Linear Scalability and proven **fault-tolerance** on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data."

- Apache Cassandra uses its own query language CQL.

When should you use a NoSQL database?

- [ ] Large amounts of data
- [ ] Need to be able to do aggregations
- [ ] Need high availability
- [ ] You need to be able to scale out quickly
- [ ] Need to be able to join multiple tables

▼ **SOLUTION:**

- Large amounts of data
- Need high availability
- You need to be able to scale out quickly

When to use a NoSQL Database

- Large amounts of data
- Need horizontal scalability
- Need high throughput -- fast reads
- Need a flexible schema
- Need high availability
- Need to be able to store different data type formats
- Users are distributed --low latency

Note: CQL is Cassandra Query Language

Common Questions:

What type of companies use Apache Cassandra?

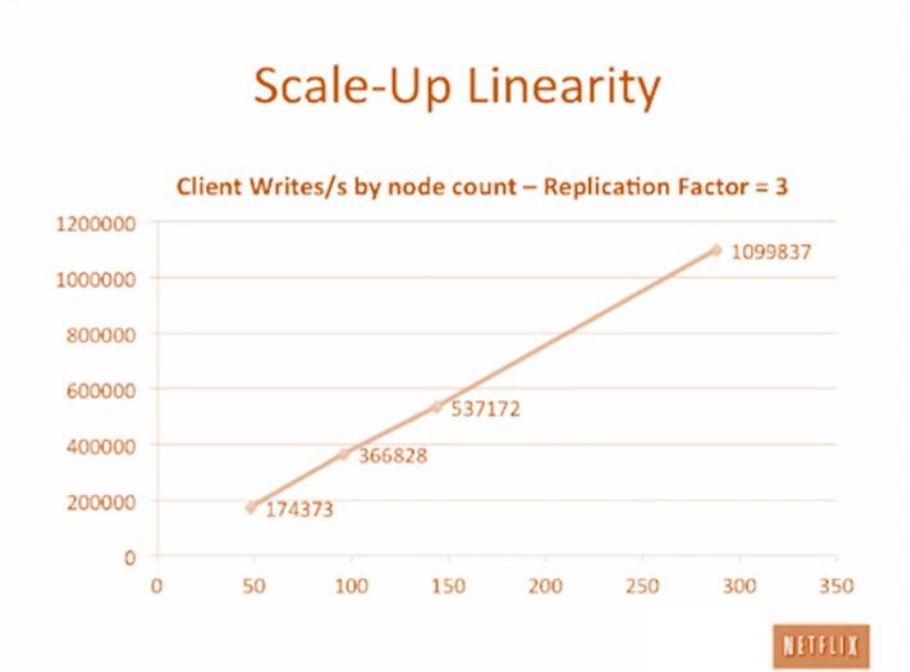
All kinds of companies. For example, Uber uses Apache Cassandra for their entire backend. Netflix uses Apache Cassandra to serve all their videos to customers. Good use cases for NoSQL (and more specifically Apache Cassandra) are :

1. Transaction logging (retail, health care)
2. Internet of Things (IoT)
3. Time series data
4. Any workload that is heavy on writes to the database (since Apache Cassandra is optimized for writes).

Would Apache Cassandra be a hindrance for my analytics work? If yes, why?

Yes, if you are trying to do analysis, such as using `GROUP BY` statements. Since Apache Cassandra requires data modeling based on the query you want, you can't do ad-hoc queries. However you can add clustering columns into your data model and create new tables.

Apache Cassandra Performance



When to not use a NoSQL Database?

- Need ACID Transactions
- Need ability to do JOINS
- Ability to do aggregations and analytics
- Have changing business requirements
- Queries are not available and need to have flexibility
- Have a small dataset

Terminology

NoSQL and Non-Relational are interchangeable terms.

NoSQL = Not Only SQL

When to Use NoSQL

- Need High Availability
- Have Large Amounts of Data
- Need Linear Scalability
- Low Latency
- Need fast reads and write
- Using Apache Cassandra -- a NoSQL database

Apache Cassandra

- Open Source NoSQL DB -- go download the code!
- Masterless Architecture
- High Availability
- Linearly Scalable
- Used by Uber, Netflix, Hulu, Twitter, Facebook, etc
- Major contributors to the project: DataStax, Facebook, Twitter, Apple

Basics of NoSQL Database Design

Distributed Databases

In a **distributed database**, in order to have high availability, you will need copies of you data.

Eventual Consistency

A consistency model used in distributed computing to achieve **high availability** that informally guarantees that, **if no new updates** are made to a given data item, eventually all accesses to that item will **return the last updated value**

The CAP Theorem

A theorem in computer science that states it is **impossible** for a distributed data store to **simultaneously provide** more than two out of the following three guarantees of **consistency**, **availability**, and **partition tolerance**.

The CAP Theorem

Consistency
Every read from the database gets the latest (and correct) piece of data or an error

Availability
Every requests is received and a response is given -- without a guarantee that the data is the latest update

Partition Tolerance
The system continues to work regardless of losing network connectivity between nodes.

Commonly Asked Questions:

- **Is Eventual Consistency the opposite of what is promised by SQL database per the ACID principle?**

Much has been written about how *Consistency* is interpreted in the ACID principle and the CAP theorem. Consistency in the ACID principle refers to the requirement that only transactions that abide by constraints and database rules are written into the database, otherwise the database keeps previous state. In other words, the data should be correct across all rows and tables. However, consistency in the CAP theorem refers to every read from the database getting the latest piece of data or an error.

- **Which of these combinations is desirable for a production system - Consistency and Availability, Consistency and Partition Tolerance, or Availability and Partition Tolerance?**

As the CAP Theorem Wikipedia entry says, "The CAP theorem implies that in the presence of a network partition, one has to choose between consistency and availability." So there is no such thing as Consistency and Availability in a distributed database since it must always tolerate network issues. You can only have Consistency and Partition Tolerance (CP) or Availability and Partition Tolerance (AP). Remember, relational and non-relational databases do different things, and that's why most companies have both types of database systems.

Commonly Asked Questions:

Does Cassandra meet just Availability and Partition Tolerance in the CAP theorem?

According to the CAP theorem, a database can actually only guarantee two out of the three in CAP. So supporting Availability and Partition Tolerance makes sense, since Availability and Partition Tolerance are the biggest requirements.

If Apache Cassandra is not built for consistency, won't the analytics pipeline break?

If I am trying to do analysis, such as determining a trend over time, e.g., how many friends does John have on Twitter, and if you have one less person counted because of "eventual consistency" (the data may not be up-to-date in all locations), that's OK. In theory, that can be an issue but only if you are not constantly updating. If the pipeline pulls data from one node and it has not been updated, then you won't get it. Remember, in Apache Cassandra it is about Eventual Consistency.

Normalization

- A simple definition is "organizing your data."
- Reduces data duplication and increases data integrity

Normalization Levels

- There are five common levels of normalization.
- Most DBAs will get to about the third normal form.
- Each additional level adds more considerations around data storage and design of tables.

Note: I have just introduced the topic of normalization here so that we can better understand denormalization in Apache Cassandra. We don't want to get into the details of each normal form at this stage.

First Normal Form (1NF)

- The table contains no repeating groups (columns).
- As an example, consider sales related to a customer. You would not include item name, price, quantity, etc. in the customer table as that would be repeated across multiple sales.

Second Normal Form (2NF)

- All attributes depend on the primary key.
- This means that values in different columns have dependencies on other columns.

Third Normal Form (3NF)

- No attributes in a table that do NOT depend on the primary key
- Consider placing a state attribute in a person table. State does NOT depend on the person primary key.

Denormalization

The process of intentionally duplicating information in a table, in violation of normalization rules

Denormalization is done to a previously normalized database.

Orders

OrderID	CustomerID
7	71
8	17
9	51
10	66

OrdersDishes

OrdersDishesID	OrderID	DishID
12	7	11
13	7	10
14	7	3

Dishes

DishID	...	...	Price
10			$9.99
11			$9.99
3			$7.00

Quantity = 3

Total = $26.98

Orders

OrderID	CustomerID	Quantity	Total
7	71	5	$26.98
8	17		
9	51		
10	66		

OrdersDishes

OrdersDishesID	OrderID	DishID
12	7	11
13	7	10
14	7	3

Dishes

DishID	...	...	Price
10			$9.99
11			$9.99
3			$7.00

Denormalization is a trade-off. Gaining speed may reduce consistency.

Denormalization in Apache Cassandra

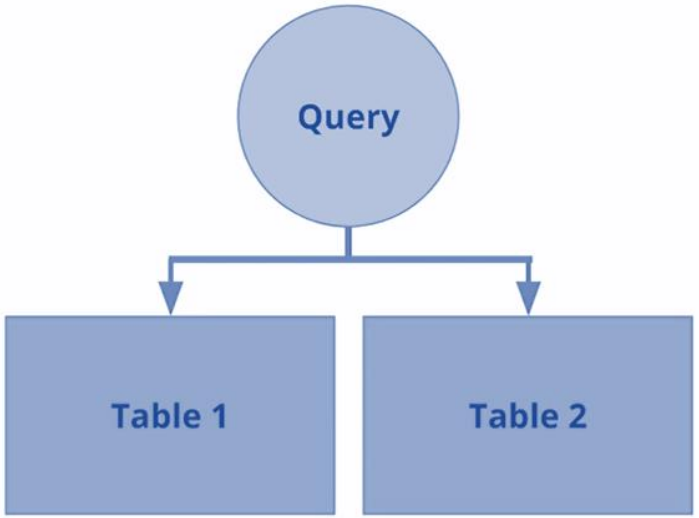
Denormalization of tables in Apache Cassandra is absolutely critical. The biggest take away when doing data modeling in Apache Cassandra is to think about your **queries** first. There are no `JOINS` in Apache Cassandra.

Data Modeling

In Apache Cassandra

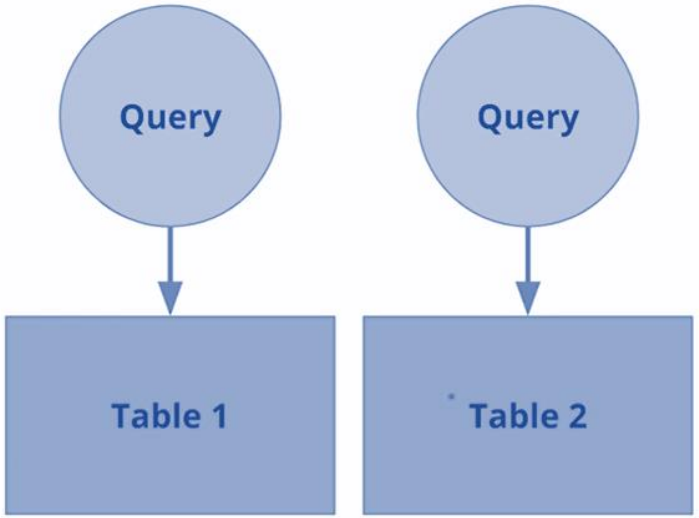
- Denormalization is not just okay -- its a must
- Denormalization must be done for fast reads
- Apache Cassandra has been optimized for fast writes
- Think Queries first

Relational Databases



In a relational database, one query can access and join data from multiple tables

NoSQL Databases



In Apache Cassandra, you cannot join data, queries can only access data from one table

Two Queries Two Tables

- All Albums in a Given Year
- All Albums by a Given Artist

```
SELECT * from music_library
WHERE year = 1965
```

Year	Artist_Name	Album_Name
1965	The Beatles	Rubber Soul
1965	The Who	My Generation

```
SELECT * from album_library
WHERE artist_name = 'The Beatles'
```

Artist_Name	Album_Name	Year
The Beatles	Let is Be	1970
The Beatles	Rubber Soul	1965

Cassandra Query Language: CQL

Cassandra query language is the way to interact with the database and is very similar to SQL. JOINS ,GROUP BY, or subqueries are not in CQL and are not supported by CQL.

Primary Key

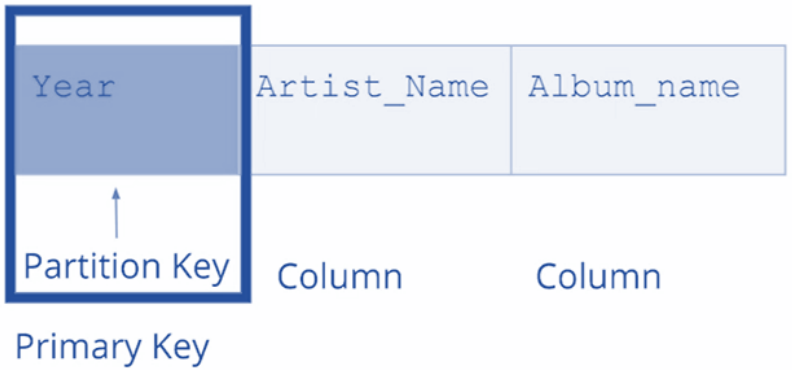
- The **PRIMARY KEY** is how each row can be uniquely identified and how the data is distributed across the nodes (or servers) in our system.
- The first element of the **PRIMARY KEY** is the **PARTITION KEY** (which will determine the distribution).
- The PRIMARY KEY is made up of either just the **PARTITION KEY** or with the addition of **CLUSTERING COLUMNS**.

Partition Key

- The **PRIMARY KEY** is made up of either just the **PARTITION KEY** or with the addition of **CLUSTERING COLUMNS**. The **PARTITION KEY** will determine the distribution of data across the system.
- The partition key's row value will be hashed (turned into a number) and stored on the node in the system that holds that range of values.

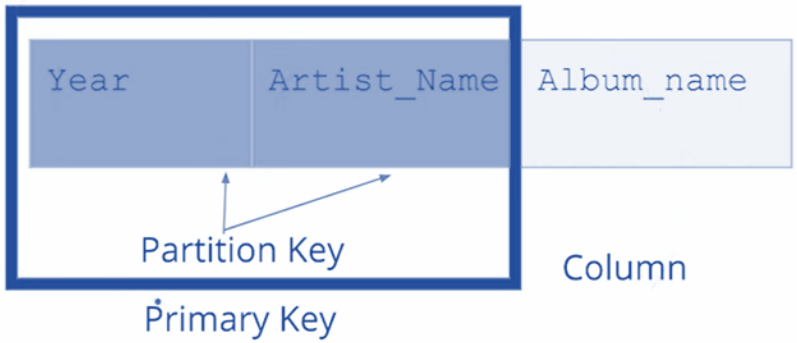
Primary Key Simple

```
CREATE TABLE
music_library
(year int,
artist_name text,
album_name text,
PRIMARY KEY (year)
```



Primary Key Composite

```
CREATE TABLE
music_library
(year int,
artist_name text,
album_name text,
PRIMARY KEY ((year,
artist_name))
```



Primary Key

Key Points

- Must be unique
- Hashing of this value results in placement on a particular node in the system
- Data distributed by this partition key
- Simple or Composite
- May have one or more clustering columns

True or False: Apache Cassandra supports duplicate rows.

- ○ True
- ○ False

▼ **SOLUTION:**
False

Which is better: Simple or Composite Primary Keys?

- ○ It depends on the data you have and the queries you will do
- ○ Simple -- Simple is always better
- ○ Composite
- ○ Neither, it is better to use a Relational Database

▼ **SOLUTION:**
It depends on the data you have and the queries you will do

Some More Notes on Primary Key

A composite Primary Key can be one of:
1.A single-column partition key and a single clustering column
2.A single-column partition key and 2 or more clustering columns
3.A composite partition key and 1 clustering column
4.A composite partition key and 2 or more clustering columns
We will see below that 3. and 4. are essentially the same thing.

Case 1:

```
CREATE TABLE TABLENAME (...,PRIMARY KEY (col1, col2));
```

By default, when you create a Primary key with 2 columns, the first column is the partition key and the second is the clustering column :

Case 2:

```
CREATE TABLE TABLENAME(...,PRIMARY KEY (col1, col2, col3));
OR
CREATE TABLE TABLENAME(...,PRIMARY KEY (col1, (col2, col3)));
```

If you have 3 (or more than 3) columns in your primary key declaration, the first column is the partition key and the remaining columns form the clustering columns. You can include round brackets/parentheses, but they aren't necessary.

Case 3:

```
CREATE TABLE TABLENAME(...,PRIMARY KEY ((col1, col2), col3));
OR
CREATE TABLE TABLENAME(...,PRIMARY KEY ((col1, col2),
(col3)));
```

Case 4:

```
CREATE TABLE TABLENAME(...,PRIMARY KEY ((col1, col2), (col3,
col4)));
OR
CREATE TABLE TABLENAME(...,PRIMARY KEY ((col1, col2), col3,
col4));
```

If the first 2 (or more than 2) columns in the primary key declaration are enclosed in round brackets/parentheses, you have a **composite partition key.** The columns which follow the composite partition key are clustering columns. You can have one clustering column, or two, or more. As before, these clustering columns may or may not be enclosed in parentheses.

Case 3 and Case 4 can really be clubbed together, but I divided them into two cases to provide a clear contrast between the examples in Case 2 and Case 3.

Clustering Columns

The **PRIMARY KEY** is made up of either just the **PARTITION KEY** or with the addition of **CLUSTERING COLUMNS**. The **CLUSTERING COLUMN** will determine the sort order within a Partition.

- More than one clustering column can be added
- Will sort in nested sorted order

```
CREATE TABLE
music_library
(year int,
artist_name text,
album_name text,
PRIMARY KEY ((year),
artist_name, album_name)
```

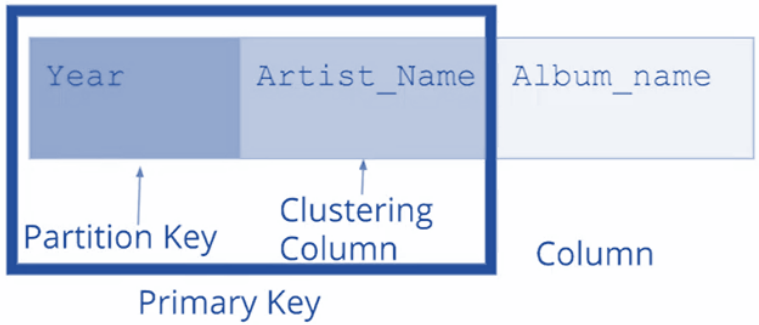
```
cqlsh: select * from music_library;

 year | artist_name | album_name
------+-------------+------------------
 1965 |       Elvis |      Blue Hawaii
 1965 | The Beatles |      Rubber Soul
 1965 | The Beatles |    Showing order
 1965 | The Monkees | Meet the Monkees

(4 rows)
```

Clustering Columns

```
CREATE TABLE
music_library
(year int,
artist_name text,
album_name text,
PRIMARY KEY ((year),
artist_name)
```



The PRIMARY KEY is made up of...

- ○ The composite key, the primary key, and the clustering key
- ○ The composite key, the partition key, and the clustering columns
- ○ The partition key and the clustering columns

▼ **SOLUTION:**
The partition key and the clustering columns

A Clustering Column is required in the Primary Key

- ○ True
- ○ False

▼ **SOLUTION:**
False

WHERE Clause

- Data Modeling in Apache Cassandra is query focused, and that focus needs to be on the **WHERE** clause.
- The **PARTITION KEY** must be included in your query and any **CLUSTERING COLUMNS** can be used in the order they appear in your **PRIMARY KEY**.

Sample Queries

```
cqlsh:> select * from music_library WHERE year=1965;

 year | artist_name | album_name
------+-------------+------------------
 1965 |       Elvis |      Blue Hawaii
 1965 | The Beatles |      Rubber Soul
 1965 | The Beatles |    Showing order
 1965 | The Monkees | Meet the Monkees

(4 rows)
cqlsh:> select * from music_library WHERE year=1965 and artist_name='The Beatles';

 year | artist_name | album_name
------+-------------+---------------
 1965 | The Beatles |   Rubber Soul
 1965 | The Beatles | Showing order

(2 rows)
cqlsh:> select * from music_library WHERE year=1965 and artist_name='The Beatles' and alb
um_name='Rubber Soul';

 year | artist_name | album_name
------+-------------+-------------
 1965 | The Beatles | Rubber Soul

(1 rows)
```

WHERE Clause

```
SELECT * from
music_library
WHERE year = 1965
AND artist_name =
'The Beatles'
AND album_name =
'Rubber Soul`;
```

Year	Artist_Name	Album_name
1965	The Beatles	Rubber Soul

Select * from table

The `WHERE` clause must be included to execute queries. It is recommended that one partition be queried at a time for performance implications. It is possible to do a select * from table if you add a configuration `ALLOW FILTERING` to your query. This is risky, but available if absolutely necessary.

Can you do `SELECT * FROM myTable` in Apache Cassandra?

- ○ Yes
- ○ No
- ○ It is highly discouraged as performance will be slow (or may just fail) but it is possible with a configuration setting
- ○ Yes, and no one should worry about it

▼ **SOLUTION:**

It is highly discouraged as performance will be slow (or may just fail) but it is possible with a configuration setting