**When Not to Use SQL:**

* **Need high Availability in the data**: Indicates the system is always up and there is no downtime
* **Have Large Amounts of Data**
* **Need Linear Scalability**: The need to add more nodes to the system so performance will increase linearly
* **Low Latency**: Shorter delay before the data is transferred once the instruction for the transfer has been received.
* **Need fast reads and write**

Here is a helpful blog that describes the [**different types of NoSQL databases**](https://www.xenonstack.com/blog/overview-types-nosql-databases/). You can bookmark it to review this later too.

**Eventual Consistency:**

Over time (if no new changes are made) each copy of the data will be the same, but if there are new changes, the data may be different in different locations. The data may be inconsistent for only milliseconds. There are workarounds in place to prevent getting stale data.

**Commonly Asked Questions:**

**What does the network look like? Can you share any examples?**  
In Apache Cassandra every node is connected to every node -- it's peer to peer database architecture.

**Is data deployment strategy an important element of data modeling in Apache Cassandra?**  
Deployment strategies are a great topic, but have very little to do with data modeling. Developing deployment strategies focuses on determining how many clusters to create or determining how many nodes are needed. These are topics generally covered under database architecture, database deployment and operations, which we will not cover in this lesson. Here is a useful link to learn more about it for [**Apache Cassandra**](https://docs.datastax.com/en/dse-planning/doc/).

In general, the size of your data and your data model can affect your deployment strategies. You need to think about how to create a cluster, how many nodes should be in that cluster, how to do the actual installation. More information about deployment strategies can be found on this [**DataStax documentation page**](https://docs.datastax.com/en/dse-planning/doc/)

**Citation for above slides:**

Here is the [**Wikipedia page**](https://en.wikipedia.org/wiki/Eventual_consistency) cited in the slides.

**Cassandra Architecture**

We are not going into a lot of details about the Apache Cassandra Architecture. However, if you would like to learn more about it for your job, here are some links that you may find useful.

**Apache Cassandra Data Architecture:**

* [**Understanding the architecture**](https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archTOC.html)
* [**Cassandra Architecture**](https://www.tutorialspoint.com/cassandra/cassandra_architecture.htm)

The following link will go more in-depth about the Apache Cassandra Data Model, how Cassandra reads, writes, updates, and deletes data.

* [**Cassandra Documentation**](https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlIntro.html)

**CAP Theorem:**

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

**Additional Resource:**

You can also check out this [**Wikipedia page**](https://en.wikipedia.org/wiki/CAP_theorem) on the CAP theorem.

**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.  
To learn more, you may find this discussion useful:

* [**Discussion about ACID vs. CAP**](https://www.voltdb.com/blog/2015/10/22/disambiguating-acid-cap/)

**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.

**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**.

**Note of correction:** At 2:55 of the video, the instructor says "Losing customers to outages or low latency is not [inexpensive]." She should have said "Losing customers to outages or poor *performance* is not [inexpensive]."

**Data Modeling in Apache Cassandra:**

* Denormalization is not just okay -- it's a must
* Denormalization must be done for fast reads
* Apache Cassandra has been optimized for fast writes
* ALWAYS think Queries first
* One table per query is a great strategy
* Apache Cassandra does **not** allow for JOINs between tables

**Commonly Asked Questions:**

* **I see certain downsides of this approach, since in a production application, requirements change quickly and I may need to improve my queries later. Isn't that a downside of Apache Cassandra?**  
  In Apache Cassandra, you want to model your data to your queries, and if your business need calls for quickly changing requirements, you need to create a new table to process the data. That is a requirement of Apache Cassandra. If your business needs calls for ad-hoc queries, these are not a strength of Apache Cassandra. However keep in mind that it is easy to create a new table that will fit your new query.

**Additional Resource:**

Here is a reference to the DataStax documents on [Apache Cassandra].([**https://docs.datastax.com/en/dse/6.7/cql/cql/ddl/dataModelingApproach.html**](https://docs.datastax.com/en/dse/6.7/cql/cql/ddl/dataModelingApproach.html))

**Primary Key**

* Must be unique
* The PRIMARY KEY is made up of either just the PARTITION KEY or may also include additional CLUSTERING COLUMNS
* A Simple PRIMARY KEY is just one column that is also the PARTITION KEY. A Composite PRIMARY KEY is made up of more than one column and will assist in creating a unique value and in your retrieval queries
* The PARTITION KEY will determine the distribution of data across the system

Here is the [**DataStax documentation**](https://docs.datastax.com/en/cql/3.3/cql/cql_using/useSimplePrimaryKeyConcept.html#useSimplePrimaryKeyConcept) on Primary Keys.

**Clustering Columns:**

* The clustering column will sort the data in sorted **ascending** order, e.g., alphabetical order. **Note: this is a mistake in the video, which says descending order.**
* More than one clustering column can be added (or none!)
* From there the clustering columns will sort in order of how they were added to the primary key

**Commonly Asked Questions:**

**How many clustering columns can we add?**  
You can use as many clustering columns as you would like. You cannot use the clustering columns out of order in the SELECT statement. You may choose to omit using a clustering column in your SELECT statement. That's OK. Just remember to use them in order when you are using the SELECT statement.

**Additional Resources:**

1. Here is the [**DataStax documentation**](https://docs.datastax.com/en/cql/3.3/cql/cql_using/useCompoundPrimaryKeyConcept.html) on Composite Partition Keys.
2. This **[Stackoverflow](https://stackoverflow.com/questions/24949676/difference-between-partition-key-composite-key-and-clustering-key-in-cassandra" \t "_blank)** page provides a nice description of the difference between Partition Keys and Clustering Keys.

**WHERE clause**

* Data Modeling in Apache Cassandra is query focused, and that focus needs to be on the WHERE clause
* Failure to include a WHERE clause will result in an error

**Additional Resource**

AVOID using "ALLOW FILTERING": Here is a reference [**in DataStax**](https://www.datastax.com/dev/blog/allow-filtering-explained-2) that explains ALLOW FILTERING and why you should not use it.

**Commonly Asked Questions:**

**Why do we need to use a WHERE statement since we are not concerned about analytics? Is it only for debugging purposes?**  
The WHERE statement is allowing us to do the fast reads. With Apache Cassandra, we are talking about big data -- think terabytes of data -- so we are making it fast for read purposes. Data is spread across all the nodes. By using the WHERE statement, we know which node to go to, from which node to get that data and serve it back. For example, imagine we have 10 years of data on 10 nodes or servers. So 1 year's data is on a separate node. By using the WHERE year = 1 statement we know which node to visit fast to pull the data from.

**What we covered in this lesson:**

* Basics of Distributed Database Design
* Must know your queries and model the tables to your queries
* Importance of Denormalization
* Apache Cassandra is a popular NoSQL database
* CQL and some key differences with SQL
* Primary Key, Partition Key, and Clustering Column
* The WHERE clause