NoSQL: Cassandra

Waheed Iqbal

DevOps (Fall 2023)
Department of Data Science, FCIT, University of the Punjab
Lahore, Pakistan

NoSQL

- NoSQL (which stands for "Not Only SQL") is a type of database that uses a non-relational data model.
- Unlike traditional relational databases, NoSQL databases are designed to handle large volumes of unstructured or semi-structured data.
- NoSQL databases are often used for big data applications, where data is stored across multiple servers or nodes in a cluster.
- NoSQL databases are often used in conjunction with traditional relational databases, as part of a hybrid or polyglot persistence strategy.

NoSQL Types

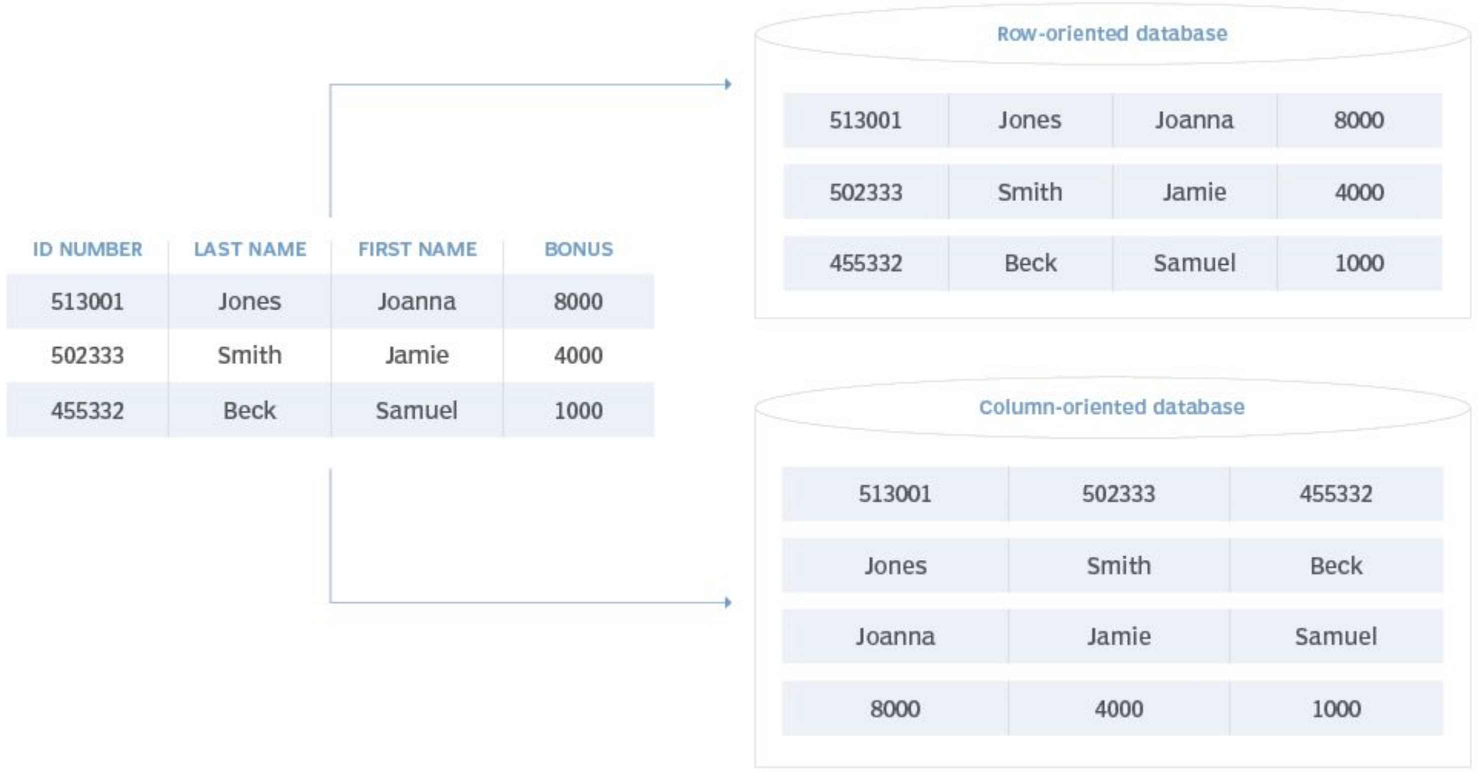
1. **Key-value stores:** In key-value stores, data is stored as key-value pairs. These databases are very simple and fast, and are often used for caching and session management. Examples include Redis, Riak, and Amazon DynamoDB.

2. **Document stores:** Document stores are designed to store and retrieve data in semi-structured formats such as JSON, XML, or BSON. They are often used for managing unstructured or semi-structured data such as social media posts, blog entries, or user profiles. Examples include MongoDB, Couchbase, and Apache CouchDB.

3. **Column-family stores:** Column-family stores store data in columns rather than rows, allowing for faster querying and data retrieval. They are often used for managing large amounts of structured data, such as time series data or log data. Examples include Apache **Cassandra**, Apache HBase, and Amazon SimpleDB.

4. **Graph databases:** Graph databases are designed to store and manage data as nodes and edges, allowing for complex relationships and interconnections between data points. They are often used for managing social networks, recommendation engines, and other types of data where relationships between entities are important. Examples include Neo4j, OrientDB, and Amazon Neptune.

Column vs Row Database



ID NUMBER	LAST NAME	FIRST NAME	BONUS
513001	Jones	Joanna	8000
502333	Smith	Jamie	4000
455332	Beck	Samuel	1000

Row-oriented database

513001	Jones	Joanna	8000
502333	Smith	Jamie	4000
455332	Beck	Samuel	1000

Column-oriented database

513001	502333	455332
Jones	Smith	Beck
Joanna	Jamie	Samuel
8000	4000	1000

Column vs Row Database (Cont.)

- Column-family databases can be faster than row-based databases for certain types of queries because they store data in a columnar format.
- When a query is executed, the database engine only needs to read the columns that are relevant to the query.
- This can be faster than row-based databases, where the entire row of data must be read even if only a small subset of columns are needed for a query.

Data Consistency Consistency

With multiple copies of the same data, we are faced with options on how to synchronize them so clients have a consistent view of the data.

Weak consistency: After a write, reads may or may not see it. A best effort approach is taken.

- This approach is seen in systems such as memcached. Weak consistency works well in real time use cases such as VoIP, video chat, and real time multiplayer games. For example, if you are on a phone call and lose reception for a few seconds, when you regain connection you do not hear what was spoken during connection loss.

Eventual consistency: After a write, reads will eventually see it (typically within milliseconds). Data is replicated asynchronously.

- This approach is seen in systems such as DNS and email. Eventual consistency works well in highly available systems.

Strong consistency: After a write, reads will see it. Data is replicated synchronously.

- This approach is seen in file systems and RDBMSes. Strong consistency works well in systems that need transactions.

CAP Theorem

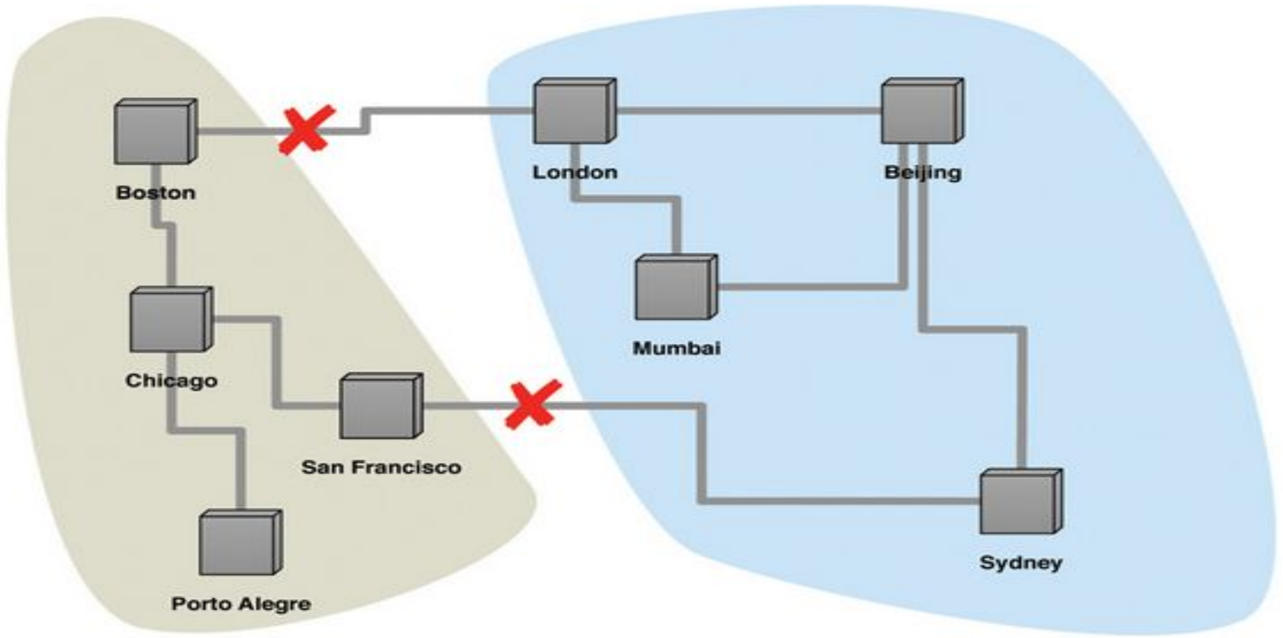
- Consistency is a Good Thing—but, sadly, sometimes we have to sacrifice it.
- In the NoSQL world it’s common to refer to the CAP theorem as the reason to relax consistency.

CAP Theorem (Cont.)

In a distributed system, you can have both **Consistency** and **Availability**, except when there is a **Partition**.

- **Consistency**: all nodes see the same data at the same time
- **Availability**: each client can always read and write any data
- **Partition Tolerance:** the system continues to operate despite failure of part of the system

CAP Theorem (Cont.)



With two breaks in the communication lines, the network partitions into two groups.

Cassandra

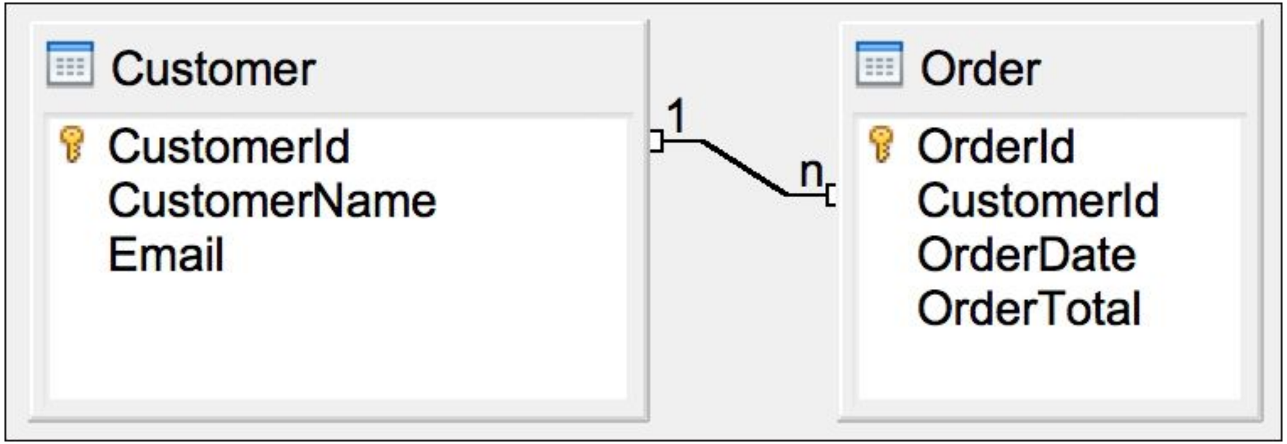
- Cassandra was initially developed at Facebook and is now managed by the Apache Software Foundation.
- It is based on a column-family data model
- Cassandra is used by many large-scale applications, including Twitter, Netflix, and eBay.
- It has a flexible and powerful query language called CQL (Cassandra Query Language) that allows you to interact with the database using SQL-like syntax.
- Cassandra is designed to provide high availability and partition tolerance, which are two of the three guarantees provided by the CAP theorem. Also provide some level of consistency too.
- Cassandra is designed to provide a balanced performance profile that can handle both write-intensive and read-intensive workloads.

Data Modeling in Cassandra

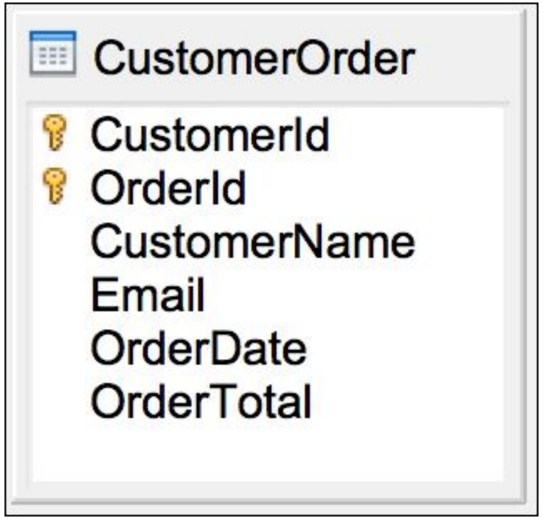
Data modeling in Cassandra is different from traditional relational database modeling because Cassandra is a distributed database with a decentralized architecture. Here are some key considerations when modeling data in Cassandra:

1. Denormalization: Cassandra is optimized for fast writes and denormalized data models that minimize the need for complex joins or table joins. This means that you should denormalize your data model to ensure that data is stored in a way that optimizes read performance.
2. Query-driven data modeling: Because Cassandra is optimized for fast reads, it's important to consider the types of queries that will be used to access the data when designing your data model. This means that you should think carefully about the queries that your application will use and structure your data model accordingly.
3. Partitioning: Cassandra uses partitioning to distribute data across multiple nodes in a cluster. This means that you should design your data model to ensure that data is partitioned in a way that allows for efficient and balanced distribution across the cluster.
4. Data duplication: In order to optimize read performance, it's often necessary to duplicate data across multiple tables or partitions. This means that you should be prepared to duplicate data in your data model in order to ensure that reads can be performed efficiently.
5. Time-series data: Cassandra is often used for storing time-series data such as logs, metrics, and event data. This means that you should consider how time-series data will be stored and queried when designing your data model.

Data Modeling in Cassandra (Cont.)



Normalized Schema



Denormalized Schema: Cassandra Style

Cassandra Terms

- Keyspace: A keyspace is a top-level namespace in Cassandra that defines a logical grouping of related tables. A keyspace is roughly analogous to a database in other systems.
- Table: A table is a collection of rows in Cassandra that share a common set of columns. Each row represents an instance of an entity or object, and each column represents a specific attribute or property of that object.
- Column: A column is a basic unit of data in Cassandra, consisting of a name, a value, and a timestamp. Each row in a table can contain multiple columns.
- Column family: In older versions of Cassandra, the term "column family" was used to refer to a collection of related columns. In more recent versions, this concept has been replaced by the more flexible and powerful concept of user-defined types and collections.
- Partition key: The partition key is the primary component of a row's primary key in Cassandra. The partition key is used to determine which node in the cluster will be responsible for storing and managing the data for that row.
- Clustering key: The clustering key is the secondary component of a row's primary key in Cassandra. The clustering key is used to determine the order in which rows are stored within a partition, and to enable sorting and range queries on the data.

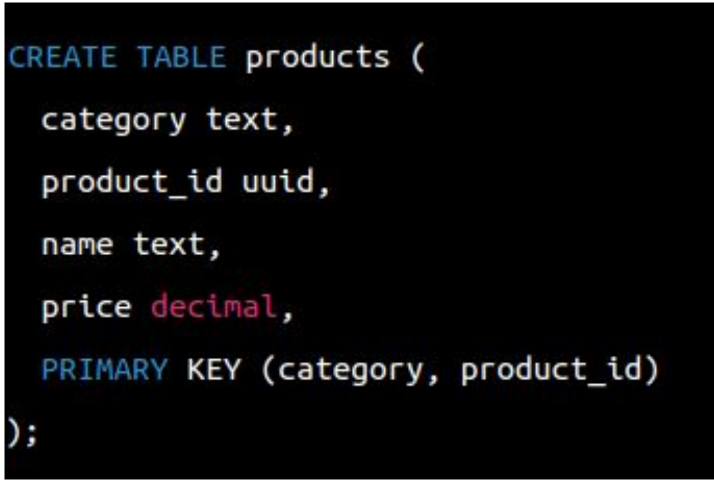
Partition Key and Clustering Key

In this table, we've defined the primary key as a composite key consisting of two components:

- The category column, which will serve as the partition key.
- The product_id column, which will serve as the clustering key.

In this case, we've chosen category as the partition key, which means that all products in the same category will be stored together on the same node. This can help to ensure that queries for products within a particular category can be executed efficiently.

We've chosen product_id as the clustering key, which means that products within a given category will be ordered by their unique product_id. This can enable efficient sorting and range queries on the data.



```
CREATE TABLE products (
  category text,
  product_id uuid,
  name text,
  price decimal,
  PRIMARY KEY (category, product_id)
);
```

Cassandra Architecture

At high level, Cassandra's architecture consists of three main components:

1. Node: A node is a single instance of Cassandra running on a physical or virtual machine. Each node in the cluster is responsible for storing and managing a portion of the data in the system.
2. Datacenter: A datacenter is a logical grouping of one or more nodes that are located in a single physical location or region. Datacenters in Cassandra are used to provide fault tolerance and high availability by replicating data across multiple geographically-distributed locations.
3. Cluster: A cluster is a group of nodes and datacenters that work together to provide a single, unified view of the database. Each node in the cluster is aware of the other nodes and datacenters in the system, and can communicate with them to coordinate data replication, load balancing, and other tasks.

Cassandra Components

- Commit log: is a write-ahead log that is used to ensure data durability in Cassandra. All writes to the database are first written to the commit log before being applied to the database itself, ensuring that data can be recovered in the event of a node failure or other disaster.
- Memtable: is an in-memory data structure that is used to temporarily store recently-written data before it is flushed to disk. This helps to improve write performance by reducing the number of disk operations required.
- SSTable: is a disk file to which all the data from the mem table flushes as it reaches a certain threshold value.
- Gossip protocol: The gossip protocol is used by Cassandra nodes to communicate with each other and share information about the state of the cluster. This includes information about which nodes are currently up or down, which datacenters are available, and other important metadata.
- Partitioner: The partitioner is responsible for dividing the data in the cluster into smaller partitions, each of which is managed by a specific node. Cassandra uses a variety of partitioning strategies, including hashed partitioning.

Memtables

- Memtables are important for Cassandra's.
- When a write operation is performed, the data is first written to the memtable.
- This means that data is immediately available for read operations, but it has not yet been persisted to disk.
- To ensure availability, Cassandra uses a technique called "hinted handoff" to replicate the data to other nodes in the cluster. This means that even if the node that received the write request fails before the data is flushed to disk, the data is still available on other nodes and can be retrieved for read operations.
- Once the data is flushed to disk, it is fully replicated to other nodes in the cluster for durability.

SSTable

- In Cassandra, SSTables are created by flushing data from memtables to disk.
- When a memtable reaches a certain size threshold, it is flushed to disk as a new SSTable file.
- Once an SSTable file is created, it is immutable, meaning that its contents cannot be modified. Instead, new data is written to new SSTable files, and older SSTables are eventually merged together in a process called compaction.

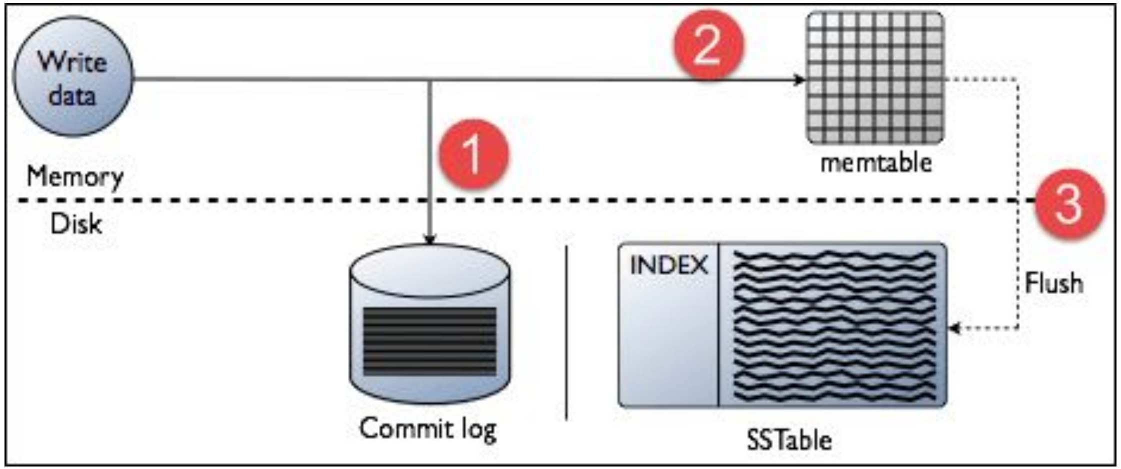
Compaction

- Compaction merges multiple SSTable files into a single file, removing obsolete data and compressing data to optimize storage.
- During compaction, multiple SSTable files are read, and their contents are merged into a new SSTable file.
- Tombstones (markers indicating deleted data) are removed, and the data is compressed to optimize storage.
- The resulting SSTable file is then written to disk, and the original SSTable files are deleted.

Replication Factor

- Replication Factor in Cassandra refers to the number of copies of data stored across the cluster.
- It is used to provide fault tolerance and high availability in Cassandra by ensuring that data is replicated across multiple nodes in the cluster.
- Replication Factor is specified at the keyspace level and can be set when creating a keyspace or updated later using the ALTER KEYSPACE command.
- The minimum recommended Replication Factor is 3, meaning that data will be replicated to at least 3 nodes in the cluster.
- When a write request is received, data is written to the nodes responsible for that data based on the partition key. If the Replication Factor is greater than 1, the data is also replicated to additional nodes.

Write in Cassandra



Read in Cassandra

- When a read request is made to Cassandra, the request is first sent to the coordinating node. The coordinating node is responsible for routing the request to the appropriate replica node(s) and returning the results to the client.
- The coordinating node determines which replica node(s) to query based on the partition key specified in the read request. If a replication factor greater than 1 is configured, the coordinating node will query multiple replicas to ensure that the data is consistent.
- If the requested data exists on multiple replica nodes, the coordinating node will compare the timestamps to determine which data is the most up-to-date. The coordinating node will then return the requested data to the client along with the timestamp.
- If the requested data does not exist on any of the replica nodes, the coordinating node will return an error to the client.

Cassandra Cluster on Kubernetes

https://kubernetes.io/docs/tutorials/stateful-application/cassandra/

References

https://cloudinfrastructureservices.co.uk/cassandra-architecture-with-diagram-components-of-cassandra/

https://cloudinfrastructureservices.co.uk/cassandra-vs-mongodb-whats-the-difference/